Programando Aplicações com Diretivas Paralelas

Natiele Lucca e Claudio Schepke

Universidade Federal do Pampa
Campus Alegrete
natielelucca@gmail.com
claudioschepke@unipampa.edu.br

15 de Abril de 2020

Sumário

Introdução

- Diferenciar OpenMP e OpenACC.
- Praticar programação paralela usando `pragmas`.
- Analisar o desempenho de aplicações em arquiteturas paralelas.
- Analisar o impacto da estratégia de paralelização.

Arquitetura Multi-Core e GPU

- Lei de Moore - dobra a quantidade de transistores a cada dois anos (MOORE, 1965).
- Melhora a capacidade dos computadores e mantém o custo.
- Consumo elevado de energia e problemas com a dissipação de calor nos processadores.
- A solução - processadores compostos por dois ou mais *cores*.
- Processadores mais simples e com menor frequência de *clock* (KIRK; WEN-MEI, 2016).

Arquitetura Multi-Core e GPU

Multi-core

- Processadores que contém múltiplas unidades de processamento, permitindo assim, a execução de instruções de forma paralela.
- Arquiteturas *multi-core* foram desenvolvidas para manter a velocidade de programas sequenciais enquanto faziam a transição para múltiplos processadores (SUTTER; LARUS, 2005).
- Utilizada para a performance de código sequencial, contendo uma lógica de controle sofisticada que permite a execução em paralelo de instruções de uma *thread*.

Arquitetura Multi-Core e GPU

Manycore

- Arquitetura *manycore* são voltados para a execução de dezenas, centenas ou até milhares de instruções simultâneas (SUTTER; LARUS, 2005).
- Possui um *hardware* especificamente projetado para suportar um grande número de *threads*, com memórias *cache* de pequeno porte, visando uma redução ao acesso à memória principal nos casos em que múltiplas *threads* acessam os mesmo dados.

OpenMP

- OpenMP é uma API para programação paralela de memória compartilhada e multiplataforma disponível em C/C++ e Fortran (OPENMP, 2020).
- A API é fundamentada no modelo de execução *fork-join*.
- Esse modelo possui uma *thread* mestre que inicia a execução e gera *threads* de trabalho para executar as tarefas em paralelo (CHAPMAN; MEHROTRA; ZIMA, 1998).

C++

```
#pragma omp [diretiva] [atributos]
```

Fortran

```
!$OMP [diretiva] [atributos]
!$omp end [diretiva]
```

OpenACC

- É uma API que contém um conjunto de diretivas de compilação para GPUs.
- Enquanto uma área paralela no OpenMP é executada por *threads* de trabalho localizadas na CPU o OpenACC especifica instruções que criam *threads* localizadas na GPU (OPENACC, 2019).

C++

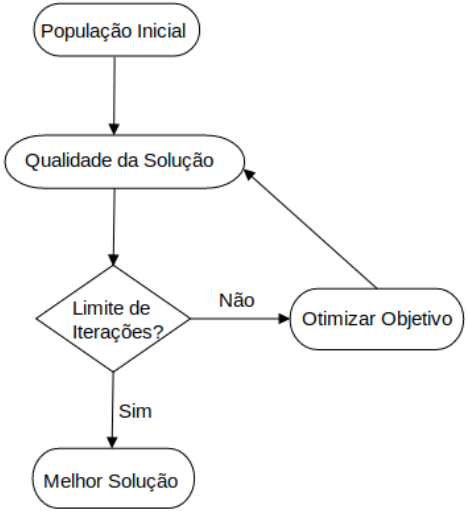
```
#pragma acc [diretiva] [clausulas]
```

Fortran

```
!$acc [diretiva] [clausulas]
!$acc end [diretiva]
```

Algoritmos Bioinspirados

- Estratégia de otimização que aplica conceitos da natureza na solução de problemas complexos (CASTRO et al., 2004).
- Objetivo de encontrar as melhores soluções para um problema.
- Simulam a forma com que o enxame interage e que tomam decisões.
- Aspectos particulares ao meio.



Algoritmo PSO

Algoritmo 3: Pseudocódigo PSO

```
1  início
2      Inicializar as partículas ;                          // Soluções iniciais
3      Inicializar a velocidade das partículas ;
4      enquanto critério de parada não for satisfeito faça
5          Calcular a aptidão das soluções;
6          Memorizar a melhor solução local ;                    // Atualizar p_B
7          Memorizar a melhor solução global ;                   // Atualizar g_B
8          Atualizar a velocidade das partículas ;
9          Atualizar a posição das partículas ;      // De acordo com a velocidade
10     fim
11     retorna Melhor Solução Encontrada
12 fim
```

Tarefa 1

Link para acesso ao código

https://github.com/NatiLucca/MinicursoERAD2020

Tarefa PSO

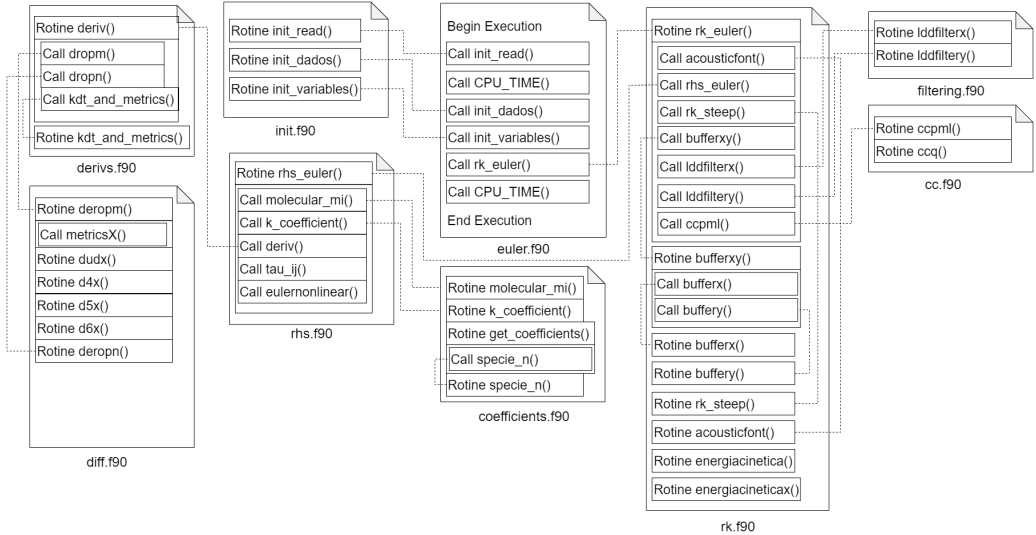
Identificar e paralelizar trechos do código com OpenMP e OpenACC.

Simulação de uma Câmera de Combustão

Descrição

- Um processo de combustão ocorre quando um material combustível reage com um material reagente.
- A camada de mistura compressível serve como um modelo para a análise de problemas de propulsão considerando a passagem de ar em alta velocidade, como a mistura de reagentes em uma câmara de combustão ou a geração de ruído nos bicos de exaustão.

Estrutura do Algoritmo



Tarefa 2

Link para acesso ao código

https://github.com/NatiLucca/MinicursoERAD2020

Tarefa

Identificar e paralelizar trechos do código com OpenMP e/ou OpenACC.

Análise Experimental

Tabela: Ambiente de execução: CPU

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

Tabela: Ambiente de execução: GPU

Características	**Quadro M5000**
Frequência	1.04 GHz
CUDA *cores*	2048
Cache L1	64 KB
Cache L2	2 MB
Memória Global	8 GB

Tempos de Execução PSO

Tabela: Casos de Teste - Melhores Resultados.

Função	PSO			
	Melhor Solução	T. Sequencial (Desvio Padrão)	T. Paralelo (Desvio Padrão)	Taxa de Desempenho (%)
1. Alpine	1,95E+00	0,370068 (0,0253)	0,091106 (0,0113)	306,19
2. Booth	1,47E-05	0,012690 (0,0014)	0,001806 (0,0002)	602,84
3. Easom	-9,64E-01	0,104991 (0,0070)	0,026976 (0,0013)	289,20
4. Griewank	1,20E+02	3,792227 (0,3100)	1,167276 (0,1064)	224,88
5. Rastrigin	3,05E+02	3,947903 (0,3863)	1,0478761 (0,1123)	276,75
6. Rosenbrock	1,30E+05	2,737766 (0,2457)	0,129914 (0,0711)	2007,38
7. Sphere	1,31E-02	0,085171 (0,0115)	0,006861 (0,0027)	1141,37

% do tempo de execução usando GProf - Câmera de Combustão

Nome da Função	Porcentagem do Tempo Total de Execução
deropn	24,11 %
rhs_euler	19,24 %
deropm	17,46 %
lddfilterx	7,92 %
lddfiltery	7,04 %
rk_euler	5,23 %
eulernonlinear	4,68 %
metricsx	3,72 %
metricsy	2,71 %
molecular_mi	1,93 %
bufferxy	1,48 %
k_coefficient	1,19 %
deriv	1,11 %

Compilar com -pg $\|executar\|gprof\ executavel\ gmon.out > saida.txt$

Tempos de Execução Simulação de uma Câmera de Combustão

Tipo	Tempo (s)	Ganho de Desemp (%)
Sequencial	126,6606	-
OpenACC	113,0066	12,08

Tabela: Taxa de ganhos de desempenho.

Tarefas

- **Objetivo: desafiar a otimização de códigos**.
- Acesse um servidor remoto:
- ssh arquitetura@200.132.136.208
- passwd: Erad2020
- Crie um diretório com seu nome: mkdir MeuNome
- Copie as pastas de exemplo para seu diretório: cp -r Exemplo1 Exemplo2 MeuNome
- Para compilar para Multicore lembrar de usar -fopenmp
- Para compilar para GPU: pgf90 -acc -Minfo-accel -O3

Considerações Finais

- Paralelizar uma aplicação possui desafios.
- É preciso garantir a equivalência numérica dos resultados, ou seja, uma versão paralela não pode resultar em valores inconsistentes da solução do problema.
- Nem sempre é possível obter ganho de desempenho com a execução paralela de trechos de código.
- A comparação realizada nas atividades permite perceber o impacto da granularidade das aplicações.
- Identificar a viabilidade do uso da GPU e custo de alterações.

Referências Bibliográficas

CASTRO, L. N. de et al. Computação natural: Uma breve visão geral. In: **Workshop em Nanotecnologia e Computação Inspirada na Biologia**. Santos, São Paulo: Universidade Católica de Santos (UniSantos), 2004.

CHAPMAN, B.; MEHROTRA, P.; ZIMA, H. Enhancing OpenMP with features for locality control. In: CITESEER. **Proc. ECWMF Workshop" Towards Teracomputing-The Use of Parallel Processors in Meteorology**. Austrian: PSU, 1998.

KIRK, D. B.; WEN-MEI, W. H. **Programming Massively Parallel Processors: a Hands-on Approach**. [S.l.]: Morgan kaufmann, 2016.

MOORE, G. E. **Cramming More Components Onto Integrated Circuits, Electronics Magazine**. 1965.

OPENACC. **What is OpenACC?** 2019. [Online; acesso 20 Fevereiro 2020]. Disponível em: ⟨https://www.openacc.org/⟩.

OPENMP. **The OpenMP API specification for parallel programming**. 2020. [Online; accessed at January, 15 2020]. Disponível em: ⟨https://www.openmp.org/⟩.

SUTTER, H.; LARUS, J. Software and the Concurrency Revolution. **Queue**. ACM