Continuação do Hands-on Pthreads e OpenMP

Lucas de Sousa Rosa e Alfredo Goldman

MAC0219 - Programação Concorrente e Paralela

10 de setembro de 2024

Mini EP4 - Problemas de Concorrência

- Vimos que a falta de atomicidade em acessos a dados compartilhados pode gerar resultados incorretos.
- Emulamos o incremento de uma variável do tipo `long` de forma não atômica.
  - Um problema real que acontece em Java.
  - Incremento de `longs` e `doubles` são tratados como duas escritas separadas de 32 bits.
- Vamos testar nossa sorte e tentar observar esse fenômeno na aula de hoje.
  - Programa `NonAtomicLong.java`.

Calculando o Produto Interno com Pthreads

- Implementação sequencial é trivial (vamos ver).
  - Implementação ingênua: `0_inner_product_seq.c`
  - Implementação útil para paralelizar: `1_inner_product_seq.c`
- Paralelizando o código com Pthreads.
  - Implementação em Pthreads (alocação estática): `2_inner_product_pth.c`
  - Implementação em Pthreads (alocação dinâmica): `3_inner_product_pth.c`

$$u \cdot v = u^T v = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \dots u_n v_n$$

O que é OpenMP?

Uma API usada para criar programas multithreaded de memória compartilhada em C/C++ e Fortran.

- Fornece capacidade para paralelizar gradualmente um programa serial.
- A API OpenMP é composta por três componentes:
  - Diretivas de compilação
  - Runtime Library Routines
  - Variáveis de ambiente.
- Disponível no Ubuntu através do pacote `libomp-dev`.
- Escrevemos bem menos código 😀 (versus Pthreads).

Escrevendo diretivas e construtor parallel

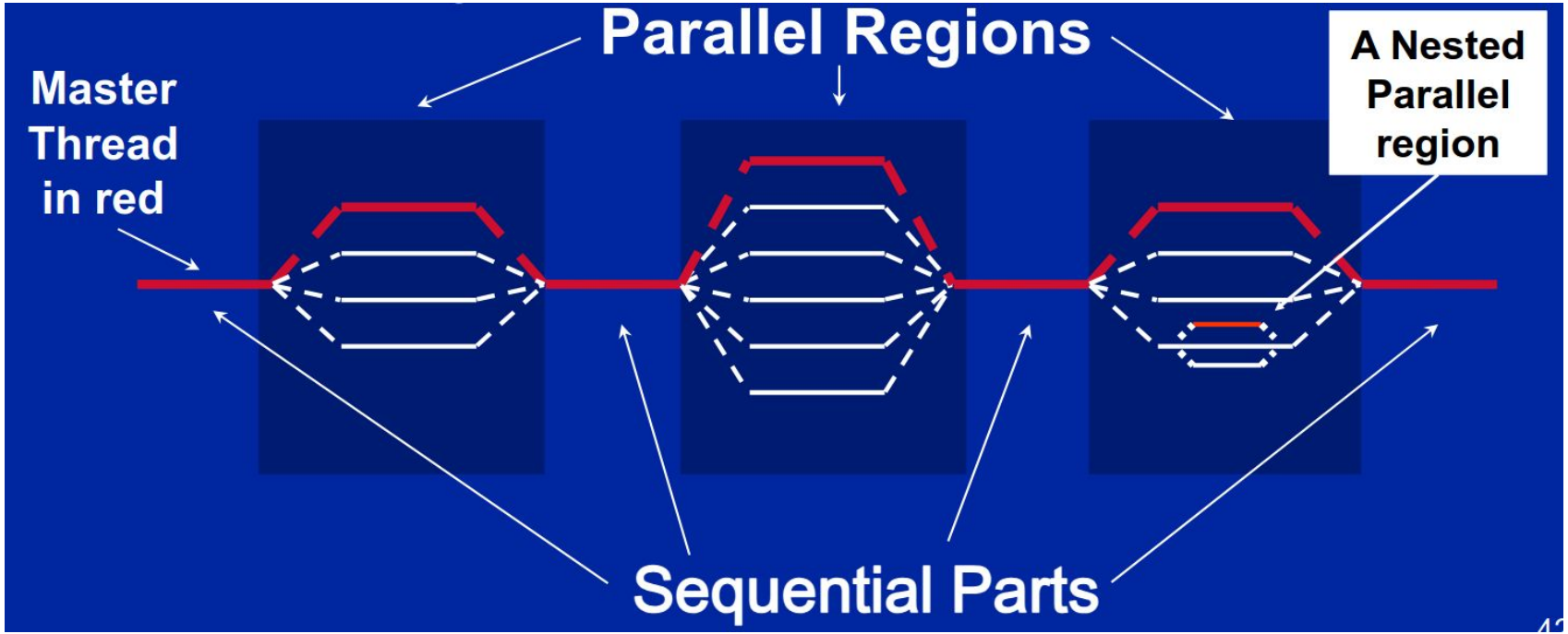
Formato: `#pragma omp directive-name [clause, ...] newline`

1. `#pragma omp`: necessário para todas as diretivas OpenMP C/C++.
2. `directive-name`: uma diretiva válida.
3. `[clause, ...]`: cláusulas são opcionais.
4. `newline`: bloco encapsulado pela diretiva (obrigatório).

Região paralela, cláusula private e número de threads (`omp_exemplo00.c`)

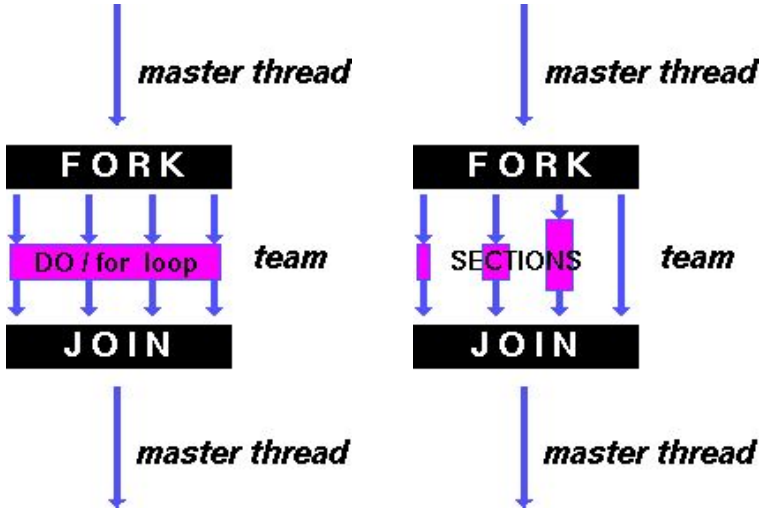
- Uma região paralela é um bloco de código que será executado por várias threads.
  - Há uma barreira implícita no final de uma seção paralela.
- Definindo o número de threads:
  - Variável de ambiente `OMP_NUM_THREADS`.
  - Função `omp_set_num_threads()`.

Construtor parallel



Construtores work-sharing e de sincronização

- Divide a execução da região de código entre as threads.
  - Não cria novas threads.
  - Não tem barreira para entrar, mas tem para sair.
- Diretiva **do/for**:
  - Compartilha iterações de um loop pela equipe (paralelismo de dados).
- Diretiva **sections**:
  - Divide o trabalho em seções separadas e discretas (paralelismo funcional).
  - Cada seção é executada por um thread.
- Eu posso combinar o construtor parallel com os work-sharing.



Construtores work-sharing e de sincronização

Algumas diretivas de sincronização são:

- `Master`: só a thread master pode executar a região.
- `Barrier`: sincroniza todas as threads
- `Critical`: especifica uma região crítica.

Diretiva do/for, cláusulas shared, schedule e nowait (`omp_exemplo01.c` e `omp_exemplo02.c`)

- `schedule`
  - Descreve como as iterações do loop são divididas entre as threads.
- `nowait`
  - Remove a sincronização no final do bloco.

Construtores work-sharing e de sincronização

Produto interno e as cláusulas default e reduce

- Implementação com região crítica: `4_inner_product_omp.c`
- Implementação com reduction (uma linha): `5_inner_product_omp.c`

Referências

- Essa aula foi preparada com base nos slides do **Pedro Bruel** (Github @phrb), nos slides do **Tim Mattson** "*Introduction to OpenMP*" nos tutoriais "*POSIX Threads Programming*" e "*OpenMP Tutorial*" do **LLNL** e no livro "*Programação Paralela e Distribuída*" do **Gabriel**, **Calebe** e **Evaldo**.
- Os códigos-fonte utilizados e outras aulas estão disponíveis no repositório do Github **fredgrub/aulas-PPD**.