OpenMP (OMP): Introdução e Primeiros Programas

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP

OMP: Introdução

- **O que é o OpenMP?**
  - ***Open-Multi-Processing (OMP)***
  - Especificação (API) para um conjunto de diretivas de compilação, funções e variáveis de ambiente
    - Lembrando diretivas
      - Em C: #include, #define, #ifdef, #endif, ...
      - Em Fortran: $omp
    - Lembrando variáveis de ambiente
      - Variáveis definidas no ambiente de execução de um processo
      - Passadas aos processos por herança, vindas do processo pai, normalmente o shell.
        - Ex: $PATH no Linux e %CD% no Windows, ...
  - OMP permite determinar paralelismo de alto nível em programas Fortran C C++
    - Aplicações *multi-threaded* e com memória compartilhada

https://www.openmp.org/about/openmp-faq/#WhatIs

OMP: Introdução

- **Qual o problema que o OMP resolve?**
  - Paralelizar um programa requer programador procure por porções paralelas/independentes
    - Muitas vezes o foco é distribuir a computação de *loops* aninhados aos elementos de processamento (processadores/núcleos)

  - Computações necessitam trocar dados, que podem vir de uma mem compartilhada
    - Requer sincronização para garantir semântica
    - Processadores são assíncronos: sincronização cabe ao software

  - Fabricantes de SMPs na década de 80
    - Desenvolveram notações específicas para paralelização de alto nível
      - Como dividir a carga de trabalho & como sincronizar a computação
      - Surgiram novas funções e diretivas de compilação para Fortran
      - Compilador usava essas informações/recursos para paralelizar o código
      - Funcionava, mas **faltava portabilidade** entre plataformas diferentes

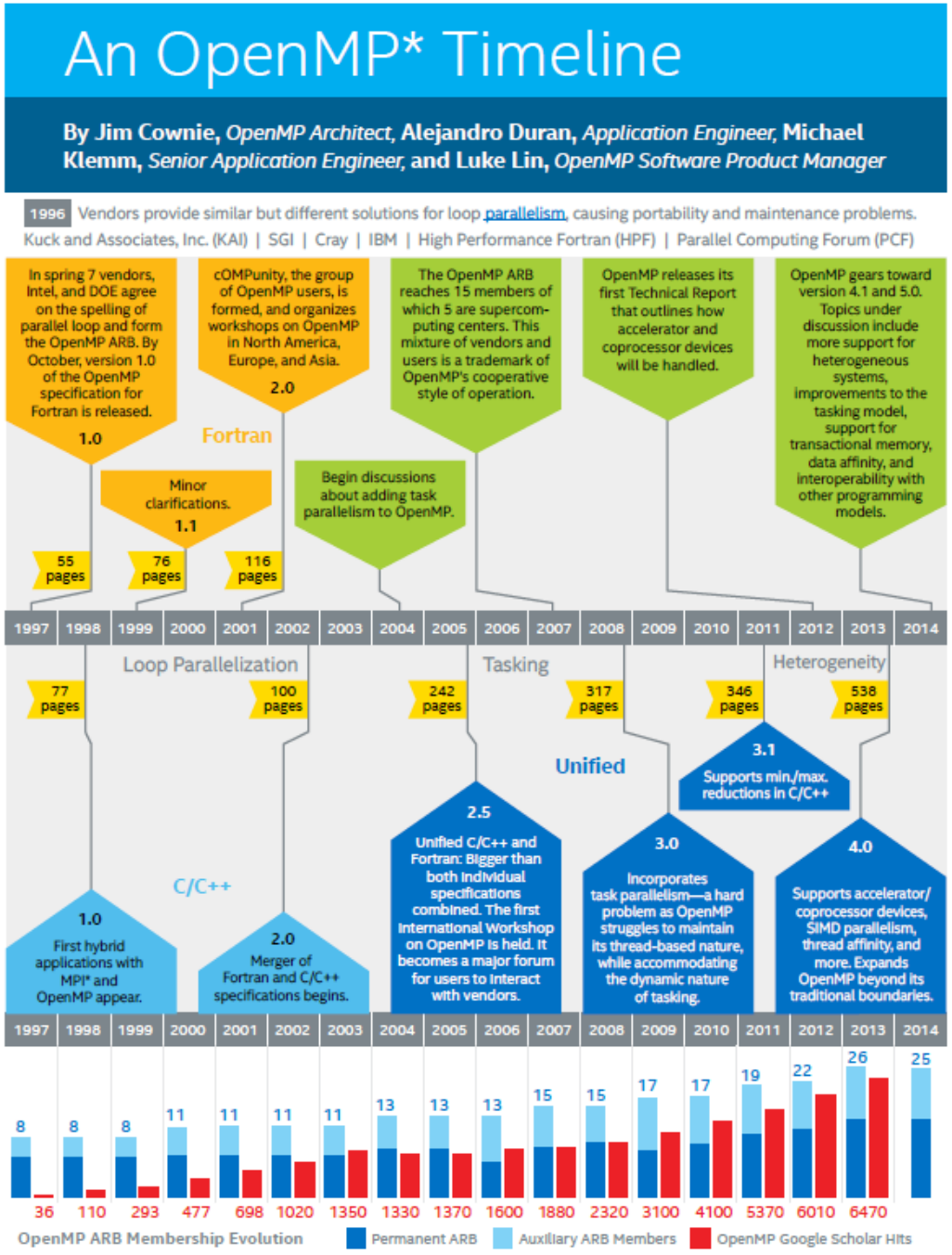
  - OpenMP ARB (*OpenMP **A**rchitecture **R**eview **B**oard*)
    - Definiu o OpenMP para solucionar o problema de **portabilidade** em 1997
    - Especificação 5.0 do OMP é de novembro de 2018

https://www.openmp.org/about/openmp-faq/#WhatIs

OMP: Quem e Quando

- Membros OpenMP ARB:
  - Compac/Digital
  - HP
  - Intel
  - IBM
  - Kuck & Associates, Inc (KAI)
  - Silicon Graphics, Inc.
  - Sun Microsystems, Inc.
  - US Dept of Energy ASCI Prog

- Desenvolvedores de Aplicativos
  - MACROS –
    - Automação industrial
  - GenASiS
    - Gen Astrophysical Sim Syst
  - Altair RADIOSS
    - Análise de impacto e *crash-test*
  - Altair OptiStruct
    - Durabilidade, vibração, ...
  - MATLAB NaN e TSA toolboxes
    - *Machine learning*, estat.
  - ...



Reinders (2014)

OMP: Alinhando Expectativas

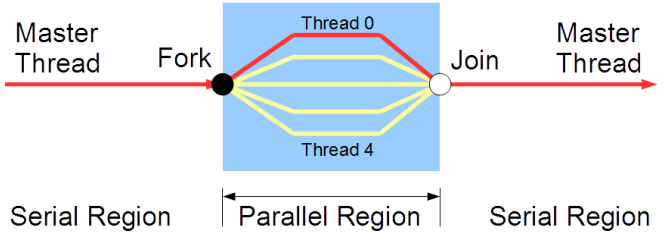
- **O que o OMP não faz ...**
  - Atuar em sistemas paralelos com memória distribuída
  - Ter implementações idênticas
  - Garantir o uso mais eficiente da memória compartilhada
  - Verificar dependências de dados, conflitos de dados, condições de disputa ou *deadlocks*
  - Verificar códigos que não se adequam ao paralelismo pretendido
  - Usar a paralelização automática gerada pelo compilador
    - *flags* de compilação
  - Garantir entrada e saída síncrona para o mesmo arquivo pelas *threads* paralelas

- **Algumas metas do OMP**
  - Oferecer portabilidade para várias plataformas MIMD com memória compartilhada
    - C, C++ e Fortran
  - Padrão simplificado com o uso de poucas diretivas
  - Fácil de usar em um paralelismo incremental sobre o código com granulação grossa ou fina
  - Fórum público para a API e participação de membros da academia, indústria e desenvolvedores

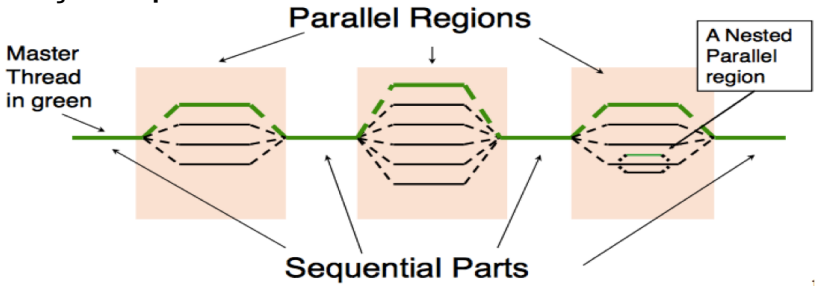
https://www.openmp.org/about/openmp-faq/#WhatIs

OMP: Modelo de Programação

- Identificação explícita do paralelismo
  - OMP oferece controle da paralelização ao programador
  - Baseado em diretivas de compilação
- Usa o **modelo de execução paralela *FORK-JOIN***
  - Programas OMP iniciam com uma thread mestre (**região sequencial**)
    - executa sequencialmente até a primeira diretiva para criar uma região paralela
  - A thread mestre criar um time (*team*) de threads paralelas (***FORK***)
    - Instruções nessas threads são executadas em paralelo (**região paralela**)
  - Há uma sincronização no final das execuções das threads paralelas, incluindo a thread mestre (***JOIN***)
    - Apenas a *thread* mestre continua executando
    - **Nova região sequencial**



- Permite paralelismo aninhado (***nested parallelism***)
  - Construções paralelas dentro de outras construções paralelas

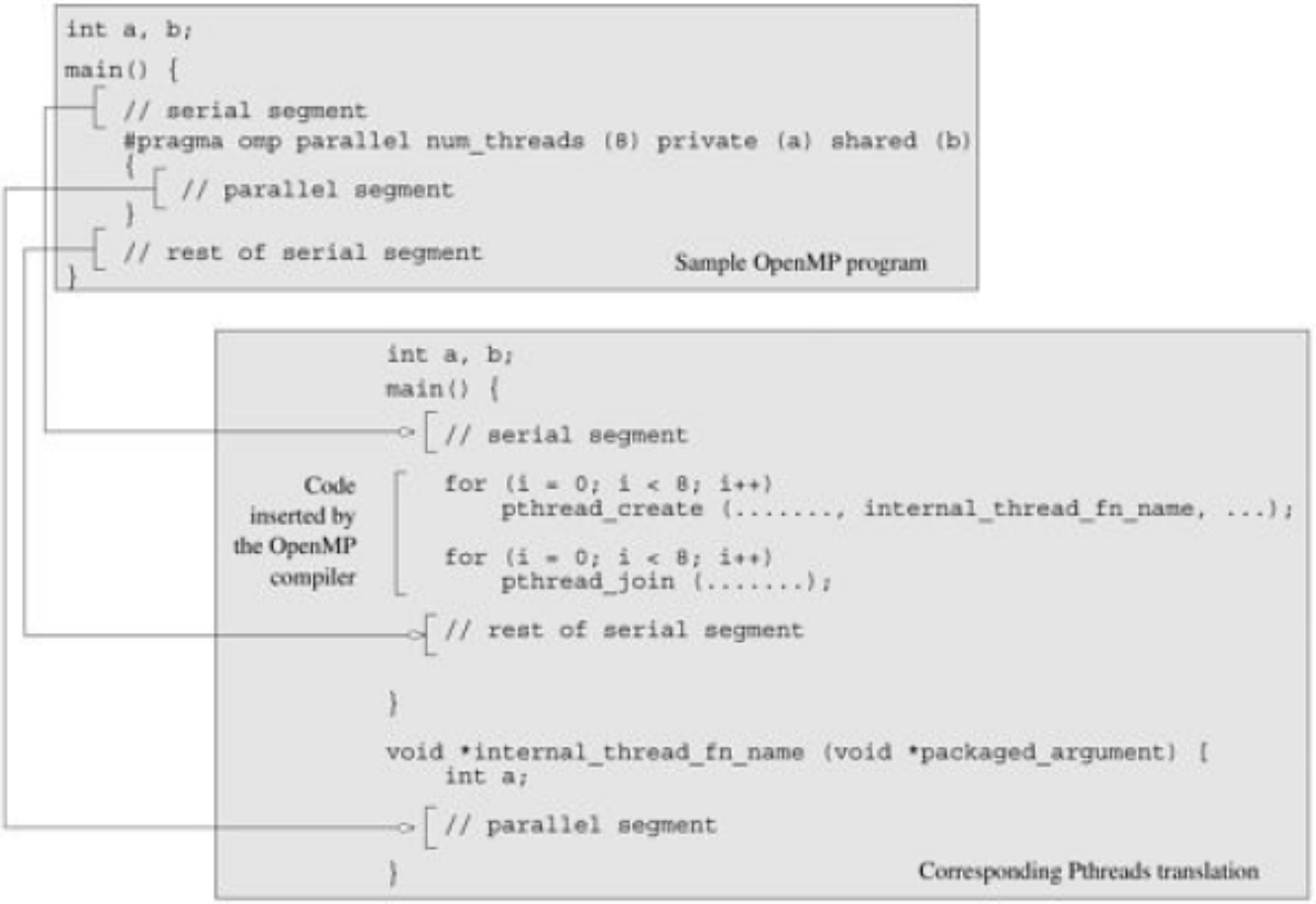


https://docs.nersc.gov/development/programming-models/openmp/openmp-resources/

OMP: Modelo de Programação

- Uma comparação com o modelo de programação *Pthreads*

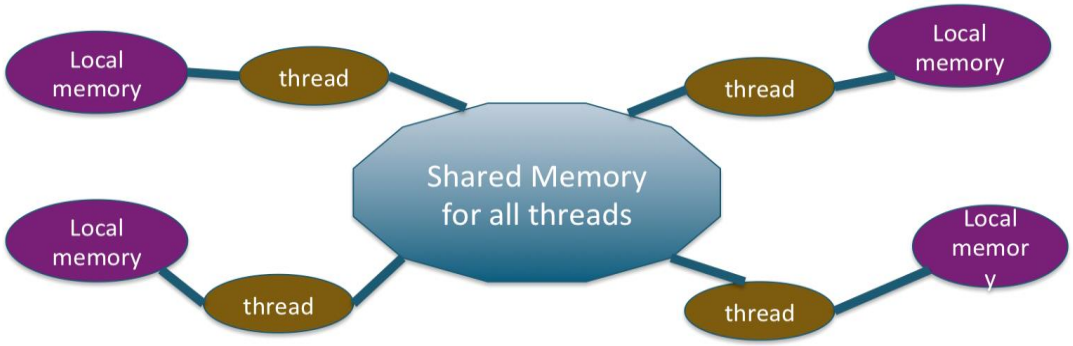
Figure 7.4. A sample OpenMP program along with its Pthreads translation that might be performed by an OpenMP compiler.



Grama et al. (2003)

OMP: Modelo de Memória

- **Memória Compartilhada com paralelismo de threads**
  - Memória global compartilhada (precisa proteger regiões críticas)
  - Memória local não compartilhada entre as *threads*



  - Consistência de Memória Relaxada e visão temporária da memória da *thread*
    - *Threads* armazenam seus dados em cache
    - Não mantêm consistência estrita com a memória física todo o tempo

  - Quando a consistência é necessária, programador usa a diretiva ***flush*** do OMP

https://docs.nersc.gov/development/programming-models/openmp/openmp-resources/

OMP: Modelo de E/S

- **OMP não especifica como a E/S paralela deve ser feita**
  - Se múltiplas *threads* acessam ao mesmo arquivo
    - Acessos concorrentes podem gerar conflito
    - Programador deve garantir o acesso correto pelas *threads*
  - Se múltiplas *threads* acessam arquivos distintos
    - Em tese não deveria haver problema com condições de disputa nos arquivos

OMP: Estrutura de um programa C

- Incluir *header*
  - ***#include <omp.h>***

- Diretivas & Cláusulas
  - ***#pragma omp <directive> [clause list]***
    - Diretivas básicas: ***parallel***, ***for***, ***sections***, ***single***, ***critical***, ...

- Funções OMP
  - ***omp_set_num_theads(), omp_get_num_threads(), omp_get_thread_num(), omp_in_parallel(), omp_get_wtime ( ),***
  - ***omp_init_lock(), omp_init_net_lock(),***
  - ***omp_set_lock, omp_unset_lock(),***
  - ***omp_set_nest_lock(), omp_unset_nest_lock,***
  - ***omp_test_lock(), omp_test_nest_lock(), ...***

- Variáveis de ambiente OMP
  - ***OMP_NESTED, OMP_NUM_THREADS, OMP_SCHEDULE, ...***

- Compilação no Linux com GCC
  - ***gcc fonte.c –o binary –fopenmp***

  - ***No Linux, basta instalar o pacote de desenvolvimento com o gcc que o OMP vem junto!***

OMP: Estrutura de um programa C

- Dicas para programas em C/OpenMP
  - Siga as convenções de diretivas para a linguagem C
  - *Case sensitive*
  - Cada diretiva refere-se ao comando abaixo dela
    - Esse comando abaixo pode ser um bloco estruturado de comandos
  - Coloque a abertura de bloco “{“ na linha abaixo da diretiva; não na mesma linha
  - Linhas de código com diretivas longas (várias cláusulas), podem ser particionadas em várias linhas, desde que use “\”

OMP: Criando Regiões Paralelas

- **Diretiva *parallel***

#pragma omp parallel *[clause[[,]clause] ...]*
structured-block
clause:
if(*scalar-expression*)
num_threads(*integer-expression*)
default(**shared** | **none**)
private(*list*)
firstprivate(*list*)
shared(*list*)
copyin(*list*)
reduction(*reduction-identifier*: *list*)
4.0 **proc_bind**(**master** | **close** | **spread**)

  - Construtor paralelo fundamental em OMP

  - Permite criar regiões paralelas

    - Times (*teams*) de *threads*

    - A *thread* já existente torna-se a mestre
      - *Thread* de número zero
      - *Threads* geradas são numeradas sequencialmente a partir de zero

    - O bloco estruturado de comandos (ou um único comado terminado com “;”)
      - Será replicado nas *threads* geradas
      - Haverá uma barreira implícita no fim do bloco estruturado replicado
        - só a *thread* mestre continua após o término da região paralela

    - Se uma *thread* terminar de maneira anormal na região paralela
      - Todas as *threads* terminam também
      - A situação final da computação feita nas *threads* é indefinido

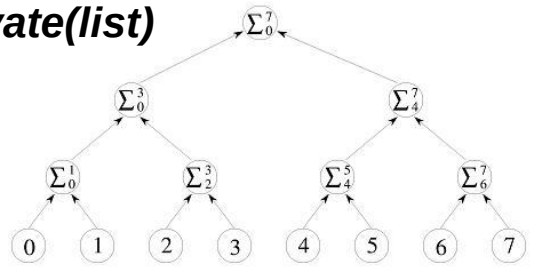
OMP Quick Reference OMP (2018)

OMP: Criando Regiões Paralelas

- Principais cláusulas da diretiva ***parallel***:
  - ***if(scalar-expression)***
    - Condiciona criação da região paralela
  - ***num_threads***
    - Número de *threads* para serem geradas.
    - Alternativas:
      - ***int omp_set_num_threads(int nthreads)***
      - ***OMP_NUM_THREADS***
      - Precedência: *if()*, *num_threads()*, *fnct*, *var_amb*, padrão da implementação
  - ***default(shared | none)***
  - ***private(list)***
  - ***firstprivate(list)***
  - ***shared(list)***
  - ***copyin(list)***
    - Copia o valor da variável *threadprivate* na *thread* mestre para as demais *threads*
    - Usada em conjunto com a ***#pragma omp threadprivate(list)***
  - ***reduction(reduction-identifier:list)***
    - ***+, *, -, &, |, ^, &&, ||, max, min***

#pragma omp parallel *[clause[[,]clause] ...]*
structured-block

clause:
if(*scalar-expression*)
num_threads(*integer-expression*)
default(**shared** | **none**)
private(*list*)
firstprivate(*list*)
shared(*list*)
copyin(*list*)
reduction(*reduction-identifier*: *list*)
4.0 **proc_bind**(**master** | **close** | **spread**)



OMP Quick Reference OMP (2018)

Foster et al. (1994)

OMP: Criando Regiões Paralelas

- Diretivas ***parallel*** aninhadas (*nested*)
  - Novas regiões paralelas podem ser geradas dentro de regiões paralelas
  - O OMP não permite geração de *threads* aninhadas por padrão
  - Deve-se habilitar regiões paralelas aninhadas usando uma dessas possibilidades
    - Função ***omp_set_nested()*** passando TRUE
    - Variável de ambiente ***OMP_NESTED*** com TRUE
  - A função ***omp_get_nested()*** informa se o aninhamento está habilitado ou não
  - Se o aninhamento não estiver habilitado
    - Uma nova diretiva ***parallel*** cria o novo time, composto de apenas uma *thread*

```
1   #pragma omp parallel for default(private) shared (a, b, c, dim) \
2                            num_threads(2)
3       for (i = 0; i < dim; i++) {
4       #pragma omp parallel for default(private) shared (a, b, c, dim) \
5                            num_threads(2)
6           for (j = 0; j < dim; j++) {
7               c(i,j) = 0;
8               #pragma omp parallel for default(private) \
9                            shared (a, b, c, dim) num_threads(2)
10              for (k = 0; k < dim; k++) {
11                  c(i,j) += a(i, k) * b(k, j);
12              }
```

Grama et al. (2003)

OMP: Distribuindo a carga de trabalho nas *threads*

- **Construtores de compartilhamento de trabalho**
  - Dividem a carga de trabalho da região paralela pelas *threads*
  - Não criam novas *threads*, usam as já existentes
    - Esses construtores devem pertencer a uma região paralela para usarem múltiplas *threads*
  - Não há uma sincronização no início da execução
    - mas há uma barreira no fim desses construtores
  - Todas as *threads* da região paralela devem encontrar os construtores de compartilhamento (ou nenhuma deve encontrar)
    - Devem encontrá-los na mesma ordem se houver mais deles na região paralela

#pragma omp for *[clause[[,]clause] ...]*
for-loops
clause:
private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*reduction-identifier*: *list*)
schedule(*kind*[, *chunk_size*])
collapse(*n*)
ordered
nowait

Exemplos de construtores de compartilhamento de trabalho:

- ***for***
  - Compartilha iterações de um loop for (paralelismo de dados)
- ***sections***
  - Divide computações distintas em threads distintas (paralelismo funcional)
- ***single***
  - Indica que o bloco estruturado será executado por uma única *thread*

OMP Quick Reference OMP (2018)

OMP: Diretiva For

- Diretiva ***for***
  - Compartilha iterações de um for (paralelismo de dados)

- O ***for*** deve estar na **forma canônica**
  - Perfeitamente aninhados
  - Número de iterações deve ser conhecido antes da execução
  - Número de iterações não pode mudar ao longo da execução
  - Não há sincronizações implícitas entre as iterações
  - Há restrições para expressões de teste e incremento da iteração

#pragma omp for *[clause[[,]clause] ...]*
for-loops
clause:
private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*reduction-identifier*: *list*)
schedule(*kind*[, *chunk_size*])
collapse(*n*)
ordered
nowait

- Cláusulas:
  - ***collapse***
    - Determina quantos loops aninhados devem ser unidos em uma mesma iteração para ser então distribuída entre as *threads*
    - Execução sequencial das iterações determina a ordem das iterações unidas por ***collapse***
  - ***ordered***
    - Especifica que as iterações devem ser executadas na mesma ordem da execução sequencial
  - ***nowait***
    - Se presente, as *threads* não sincronizarão na barreira no fim do ***for***

OMP Quick Reference OMP (2018)

OMP: Escalonamento de Iterações do *for* às *Threads*

- Escalonamento das iterações do ***for*** nas ***threads***

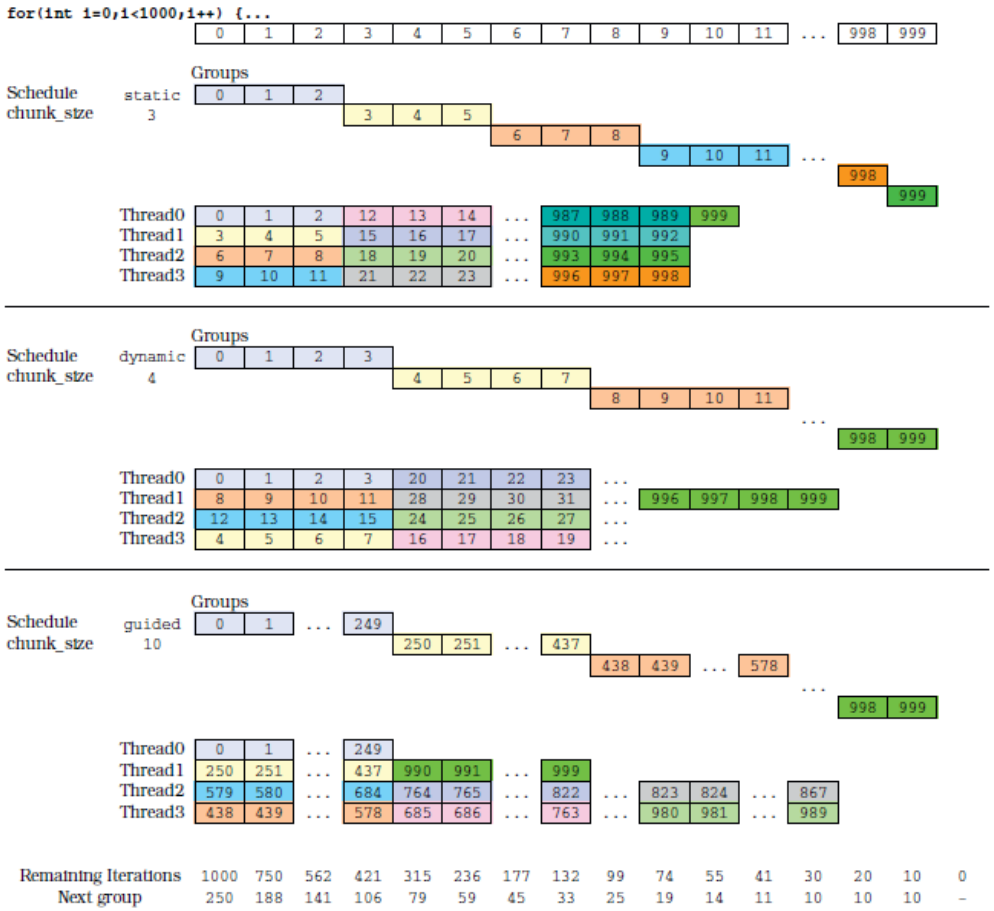


FIGURE 4.7

Examples of resulting iteration partitioning and assignment based on the scheduling scheme for a 1000-iteration `for` loop. We end up with 334 groups for `static`, 250 groups for `dynamic`, and 15 groups for `guided` scheduling for the given `chunk_size` parameters. Only for the `static` scheme do we have *a priori* knowledge of the iteration group assignment to threads.

Static: iterações divididas em porções de *chunk_size* elementos e atribuídas às threads de maneira *Round-Robin*. Padrão para *chunk_size* é ***Nr_Iter/Nr_Threads***.

Dynamic: cada *thread* executa *chunk_size* iterações e requisita mais até finalizar. Padrão para *chunk_size* = 1.

Guided: Cada *thread* executa *chunk_size* iterações, sendo que a cada novo pedido por requisições, o número de iterações é recalculado usando ***Nr_Threads Restantes*** / ***Nr_Threads***. *Chunk_size* padrão é 1.

Auto: Decisão pelo escalonamento é feita pelo compilador ou em tempo de execução.

Runtime: Escalonamento decidido pela variável de ambiente.

Barlas et al. (2015)

OMP: Diretiva *section*

- Diretiva ***sections***
  - Divide computações distintas em *threads* distintas (paralelismo funcional)

```
#pragma omp sections [clause[ [, ] clause] ...]
    {
    [#pragma omp section]
        structured-block
    [#pragma omp section
structured-block]
    ...
    }
clause:
    private(list)
    firstprivate(list)
    lastprivate(list)
    reduction(reduction-identifier: list)
    nowait
```

```
1       #pragma omp parallel
2       {
3           #pragma omp sections
4           {
5               #pragma omp section
6               {
7                   taskA();
8               }
9               #pragma omp section
10              {
11                  taskB();
12              }
13              #pragma omp section
14              {
15                  taskC();
16              }
17          }
18      }
```

Grama et al. (2003)

OMP: Diretiva *single*

- Diretiva ***single***
  - Indica que o bloco estruturado será executado por uma única *thread*

#pragma omp single *[clause[[,]clause] ...]*
structured-block
clause:
private(*list*)
firstprivate(*list*)
copyprivate(*list*)
nowait

- Diretiva ***critical***
  - Identifica uma região crítica que precisa ser protegida
  - Código estruturado terá acesso a apenas uma *thread* por vez

#pragma omp critical (name)
structured block

  - Argumento (***name***) é opcional e permite diferenciar regiões críticas

  - Há outras maneiras de proteger regiões críticas:
    - ***atomic, locks***, uso de semáforos de *Pthreads*, ...

- Diretiva ***atomic***
  - Garante atomicidade a operações de *load/store* em uma posição de memória
    - x++, ++x, x--, --x, e mais operações binárias com expressões

OMP Quick Reference OMP (2018)

OMP: Exemplos de Programas

- Hello World
  - Exemplo de uso da diretiva *parallel* com a cláusula *reduction*

- Soma Elementos de um Vetor de Números Inteiros
  - Analise a facilidade para o desenvolvimento de código paralelo
  - Veja o desempenho ao usar *critical* e *reduction*

- Números perfeitos até N
  - Analise as diferenças de desempenho com escalonamentos diferentes do for

- Multiplicação de Matrizes com a cláusula *collapse*
  - Veja o impacto no desempenho com o uso de *collapse*

- O algoritmo exemplo para *Nested-Parallel*
  - Veja os números das *threads*

- O algoritmo *Nested-Sections* exemplifica o paralelismo funcional com aninhamento
  - Verifique os números das *threads* usadas pelas seções.

Referências

Barlas G.; Multicore and GPU Programming: an integrated approach. Elsevier, 2015.

FOSTER, I. Designing and Building Parallel Programs, Addison-Wesley Publishing Company, 1994.

GRAMA,A.; KUMAR, U.; GUPTA,A.; KARYPIS, G. Introduction to Parallel Computing, 2nd Edition, 2003.

OpenMP API 5.0 C/C++ Syntax Quick Reference Card. 2018. Disponível em https://www.openmp.org/wp-content/uploads/OpenMPRef-5.0-111802-web.pdf . Último acesso em 28/09/2020.

OpenMP Application Program Inferface, Version 5.0 – 2018, Disponível em https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf. Último acesso em 28/09/2020.

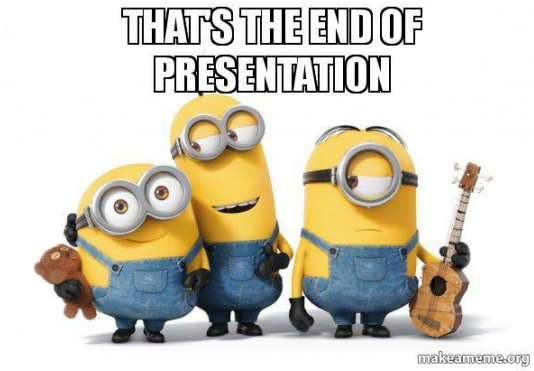
Reinders, James (Ed.) The Parallel Universe: flow Graphs, Speculative Locks, and Tasks Arenas. Intel. 2014. Disponível em https://www.openmp.org/about/whos-using-openmp/ Útlimo acesso em 28/09/2020

OpenMP Resources. NERSC Documentation (National Energy Research Scientific Computing Center. Berkeley Lab. Disponível em: https://docs.nersc.gov/development/programming-models/openmp/openmp-resources/. Último acesso em 28/09/2020.

OpenMP: Introdução e Primeiros Programas

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente



CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP