Arquiteturas Paralelas e Distribuídas

Ferramentas para programação paralela: OpenMP

Eduardo Furlan Miranda

Baseado em: OPENMP. The OpenMP API specification for parallel programming. 2025. https://www.openmp.org/

OpenMP (Open Multi-Processing)

- API que oferece suporte à programação multiprocessamento de memória compartilhada multiplataforma em C , C++ e Fortran, em muitas plataformas, arquiteturas de conjunto de instruções e sistemas operacionais, incluindo Linux, MacOS e Windows

- Consiste em
  - um conjunto de diretivas do compilador
  - rotinas de biblioteca
  - variáveis de ambiente

que influenciam o comportamento em tempo de execução

- Desenvolvido por Arm, AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Red Hat, Texas, e Oracle

OpenMP

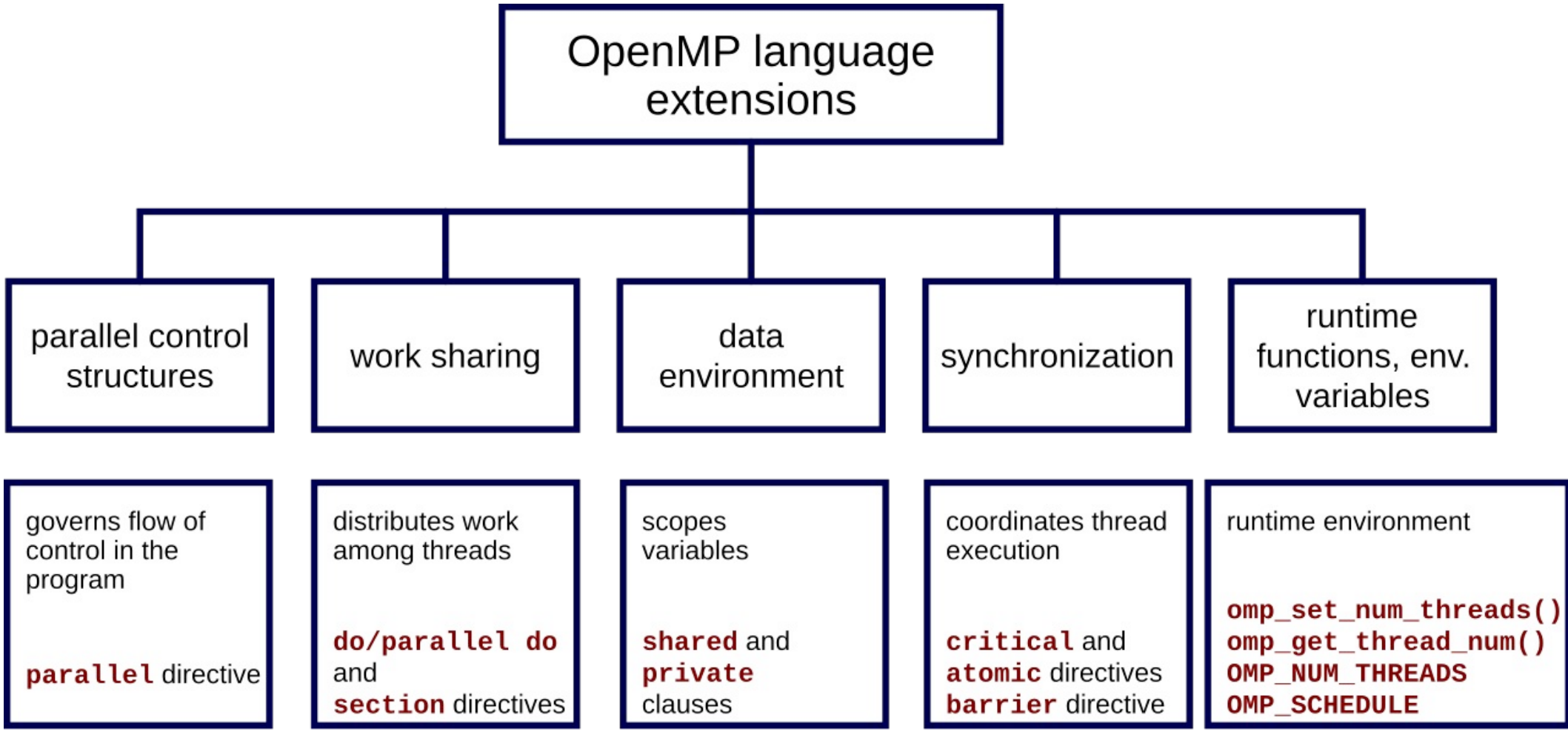
- Modelo portátil e escalável que oferece aos programadores uma interface simples e flexível para desenvolver aplicativos paralelos para plataformas que vão do computador desktop padrão ao supercomputador
- Geralmente usado dentro de um nó (multi-core) enquanto o MPI é usado entre nós
- Implementação de multithreading, um método de paralelismo pelo qual um thread primário bifurca um número especificado de sub-threads e o sistema divide tarefas entre eles
  - As threads então são executados simultaneamente, com o ambiente de tempo de execução alocando threads para diferentes processadores (em um nó)

- Cada thread tem um ID anexado a ele que pode ser obtido usando uma função (chamada omp_get_thread_num())
- O ID do thread é um inteiro, e o thread primário tem um ID=0
- Após a execução do código paralelizado, os threads se juntam novamente ao thread primário, que continua até o final do programa
- Por padrão, cada thread executa a seção paralelizada do código de forma independente

- Construções de compartilhamento de trabalho podem ser usadas para dividir uma tarefa entre as threads para que cada thread execute sua parte alocada do código

- Tanto o paralelismo de tarefas quanto o paralelismo de dados podem ser alcançados usando OpenMP dessa forma

- O ambiente de tempo de execução aloca threads para processadores dependendo do uso, carga da máquina e outros fatores
  - Pode atribuir o número de threads com base em variáveis de ambiente, ou o código pode fazer isso usando funções
  - As funções OpenMP são incluídas em um arquivo de cabeçalho rotulado omp.h em C/C++


OpenMP language extensions
parallel control structures
work sharing
data environment
synchronization
runtime functions, env. variables
governs flow of control in the program
parallel directive
distributes work among threads
do/parallel do and section directives
scopes variables
shared and private clauses
coordinates thread execution
critical and atomic directives barrier directive
runtime environment
omp_set_num_threads()
omp_get_thread_num()
OMP_NUM_THREADS
OMP_SCHEDULE

Criação de thread

- O #pragma omp parallel é usado para bifurcar threads adicionais para executar o trabalho incluído na construção em paralelo

- A thread original será denotada como thread mestre com ID 0

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

- Compila com: $ gcc -fopenmp hello.c -o hello -ldl
  - O pacote gcc já inclui a biblioteca OpenMP

- Saída:
  Hello, world.
  Hello, world.

- Pode ocorrer race condition:
  Hello, wHello, woorld.
  rld.

a função printf pode ou não ser atômica, dependendo da implementação

Construções de compartilhamento de trabalho

- Usado para especificar como atribuir trabalho independente a um ou todos os threads
  - omp for ou omp do : usado para dividir iterações de loop entre os threads, também chamados de construções de loop.
  - seções : atribuição de blocos de código consecutivos, mas independentes, a diferentes threads
  - single : especificando um bloco de código que é executado por apenas um thread, uma barreira é implícita no final
  - master : semelhante ao single, mas o bloco de código será executado apenas pelo thread master e não haverá nenhuma barreira implícita no final

Exemplo

- Inicializar o valor de uma grande matriz em paralelo, usando cada thread para fazer parte do trabalho

```
int main(int argc, char **argv)
{
   int a[100000];

   #pragma omp parallel for
   for (int i = 0; i < 100000; i++) {
      a[i] = 2 * i;
   }

   return 0;
}
```

- Este exemplo é embaraçosamente paralelo e depende apenas do valor de i

- O sinalizador parallel for do OpenMP diz para dividir esta tarefa entre seus threads de trabalho
- Cada thread receberá uma versão única e privada da variável
- P. ex., com dois threads de trabalho, um thread pode receber uma versão de i que vai de 0 a 49999, enquanto o segundo recebe uma versão que vai de 50000 a 99999

Diretivas variantes

- Permitem a adaptação de pragmas e código de usuário em tempo de compilação

- Define características para descrever construções OpenMP ativas, dispositivos de execução e funcionalidade fornecidos por uma implementação, seletores de contexto com base nas características e condições definidas pelo usuário, e diretivas metadirective e declare directive para que os usuários programem a mesma região de código com diretivas variantes

```
// code adaptation using preprocessing directives

int v1[N], v2[N], v3[N];
#if defined(nvptx)
#pragma omp target teams distribute parallel for map(to:v1,v2)
map(from:v3)
for (int i= 0; i< N; i++)
   v3[i] = v1[i] * v2[i];
#else
#pragma omp target parallel for map(to:v1,v2) map(from:v3)
for (int i= 0; i< N; i++)
   v3[i] = v1[i] * v2[i];
#endif

// code adaptation using metadirective in OpenMP 5.0

int v1[N], v2[N], v3[N];
#pragma omp target map(to:v1,v2) map(from:v3)
#pragma omp metadirective \
when(device={arch(nvptx)}: target teams distribute parallel for)\
default(target parallel for)
for (int i= 0; i< N; i++)
   v3[i] = v1[i] * v2[i];
```

Cláusulas de atributo de compartilhamento de dados

- shared: dados fora de região paralela são visíveis e acessíveis por todos os threads. Iteradores de loop são privados por padrão
- private: dados dentro de região paralela são privados e não inicializados. Iteradores de loop são privados
- default: define escopo padrão como shared ou none em C/C++, e shared, firstprivate, private, ou none em Fortran
- firstprivate: como private, mas inicializado com valor original
- lastprivate: como private, mas valor original atualizado após construção
- reduction: forma segura de unir trabalho dos threads após construção

Cláusulas de sincronização

- crítico: bloco de código executado por um thread de cada vez, usado para proteger dados compartilhados de condições de corrida
- atomic: a próxima instrução de atualização de memória (write, read-modify-write) é executada atomicamente, melhorando desempenho
- ordenado: bloco estruturado executado na ordem sequencial das iterações de loop
- barrier: cada thread espera até todas as threads da equipe alcançarem o ponto de sincronização
- nowait: threads que concluem o trabalho podem prosseguir sem esperar a sincronização de barreira no final

Cláusulas de agendamento

- schedule (type, chunk): define método de agendamento para do-loop ou for-loop
- static: iterações divididas igualmente entre threads, chunks opcionais
- dynamic: iterações alocadas dinamicamente, reassign após conclusão
- guiado: pedaços de iterações contíguas, tamanho diminui exponencialmente

Controle e Inicialização

- if: paraleliza tarefa se condição atendida, caso contrário serial
- firstprivate: dados privados, inicializados com valor mestre
- lastprivate: dados privados, copiados para variável global fora da região
- threadprivate: dados globais privados em regiões paralelas

Cópia de Dados e Outros

- copyin: inicializa variáveis threadprivate com valores globais
- copyprivate: copia dados privados de thread único para outros
- redução: variáveis locais resumidas em variável compartilhada global
- flush: valor restaurado do registro para memória fora de região paralela
- master: executado somente pelo thread master, sem barreira implícita

Compiladores com Suporte OpenMP

- Visual C++ (2005-2013): OpenMP 2.0 em edições profissionais
- Intel Parallel Studio: Suporta vários processadores
- Oracle Solaris Studio: Suporte OpenMP atualizado para Solaris e Linux
- The Portland Group: OpenMP 2.5 para Fortran, C e C++
- GCC: Suporte OpenMP desde versão 4.2

Compiladores OpenMP 3.0 e 3.1

- GCC 4.3.1, 4.7
- Mercury Compiler
- Intel Fortran e C/C++: v11.0, 11.1, 12.1
- IBM XL Compiler
- Sun Studio 12 Update 1
- LLVM/Clang 3.7
- Absoft Fortran v19

Compiladores OpenMP 4.0 a 5.0

- GCC 4.9.0, 4.9.1
- Intel Fortran e C/C++: v15.0, 17.0, 18.0, 19.0, 19.1
- IBM XL C/C++ e Fortran: V13.1, V15.1
- LLVM/Clang 12

Compiladores Automáticos e Ferramentas

- Automáticos: iPat/OMP, Parallware, PLUTÃO, ROSE, S2P, ComPar, PragFormer
- Ferramentas: Intel VTune Profiler, Intel Advisor, Allinea DDT e MAP, TotalView, ompP, VAMPIR