OpenMP

Yuri Kayser da Rosa
Daniel Yoshizawa
Everton Coelho

Introdução

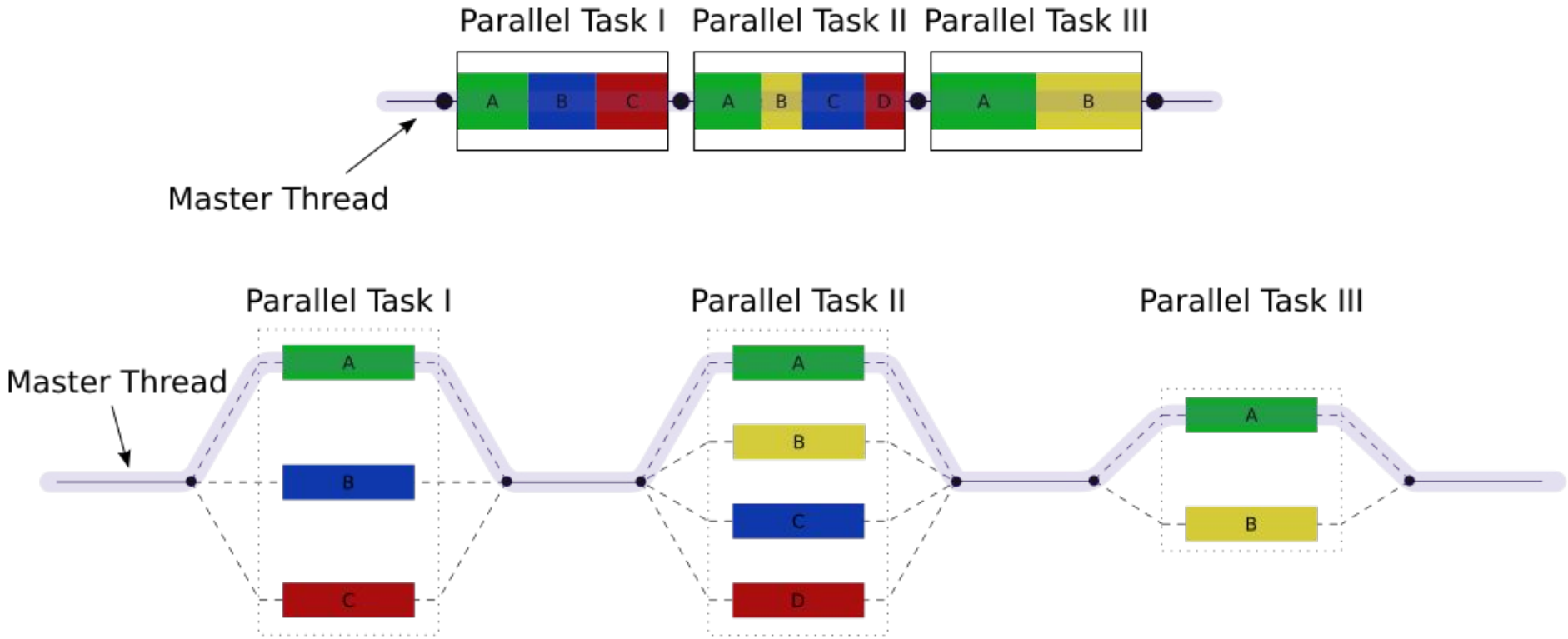
- API para programação paralela em memória compartilhada nas linguagens C/C++ e Fortran.
- Multi-Plataforma
- Mantida pela OpenMP architecture Review Board, uma organização composta por grandes nomes do mercado como IBM e Intel.

História

- OpenMP 1.0 Fortran (1997) C/C++ (1998)
- OpenMP 2.0 (2000) C/C++ (2002)
- OpenMP 3.0 (2008)
- OpenMP 4.0 (2014)
- Atual 4.5

Multithreading

- Para realizar a paralelização, o OpenMP utiliza o modelo de Multithreading, em que uma thread Master se divide em determinados momentos para ser executada em paralelo por outros processadores.
- Fork and Join.


Parallel Task I
Parallel Task II
Parallel Task III
A
B
C
A
B
C
D
A
B
Master Thread
Parallel Task I
Parallel Task II
Parallel Task III
Master Thread
A
B
C
A
B
C
D
A
B

Diretivas

- Mas como indicamos quando um pedaço do nosso código deve ser paralelizado?
- Para isso, utilizamos as Diretivas. Instruções inseridas no código fonte que indicam ao compilador que um determinado trecho deve ser paralelizado.

Diretivas

- As diretivas são sempre indicadas da seguinte forma:
- #pragma omp name [clause…]

Diretiva parallel

- Roda o código em paralelo

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

Diretiva for

- Paraleliza um loop

```
#pragma omp parallel for
for (int i = 1; i < n; i++) {
    a[i] = 2 * i;
}
```

Diretiva barrier

- Threads esperam até que as outras threads cheguem naquele ponto

```
#pragma omp parallel for
for (i=1; i<n; i++) {
    a[i] = 2 * i;
}
#pragma omp barrier
```

Diretiva critical

- Somente uma thread pode acessar a região por vez

```
#pragma omp critical
    a = operacaoCritica();
```

Diretiva sections

- Permite cada thread executar um caminho diferente

```
void sect_example(){
    #pragma omp parallel sections
    {
        #pragma omp section
        XAXIS();
        #pragma omp section
        YAXIS();
        #pragma omp section
        ZAXIS();
    }
}
```

Diretiva atomic

- Protege a região crítica a um update de um endereço de memória

```
#pragma omp atomic read
   value = *p;
   return value;
}
```

Diretiva single

- Trecho é executado por uma única thread. Outras threads aguardam

```
for (i = 0; i < 10; i++) {
  a += i;
}

#pragma omp single
printf ("Sum is %d\n", a);
```

Vantagens

- Simples, não precisa de troca de mensagens como o MPI
- Codigo unificado para programas seriais e paralelos
- Pode ser usado em aceleradores como placas GPGPU

Desvantagens

- Requer um compilador que suporte OpenMP.
- Escalabilidade limitada pela arquitetura de memória
- Falta de mecanismo de baixa-granularidade para lidar com o mapeamento de threads-processador.

MPI

Message Passing Interface

Introdução

- É um padrão para comunicação de dados em computação paralela
- Usado em C, C++ e Fortran, mas pode ser utilizado em qualquer linguagem
- Existem várias versões largamente testadas e geralmente open-source
- Desenvolvido por pesquisadores acadêmicos e da indústria
- O objetivo de MPI é prover um amplo padrão para escrever programas com passagem de mensagens de forma prática, portátil, eficiente e flexível. MPI não é um IEEE ou um padrão ISO, mas chega a ser um padrão industrial para o desenvolvimento de programas com troca de mensagens

- Uma aplicação é constituída por uma ou mais tarefas que se comunicam utilizando funções para envio e recebimento de mensagens entre os processos.
- Mecanismos de comunicação ponto a ponto
- Collective é chamado para executar operações globais
- Comunicação assíncrona e programação modular através do communicator

Motivação

- Portabilidade: MPI permite que você coloque sua aplicação em uma plataforma diferente, mas que suporte o padrão MPI sem necessitar alterar o código de forma que processos em diferentes linguagens podem rodar em plataformas diferentes.
- Funcionalidade: Mais de 300 rotinas são definidas em MPI.
- Disponibilidade: Há uma enorme variedade de implementações disponíveis.

Utilização

- Cada tarefa MPI deve ser identificada por um ID
- Atribui a cada processo um rank e um grupo de comunicação
- Ranks identificam os processos
- Todos os processos são inicializados em um grupo default
- O grupo define quais processos podem chamar comunicação ponto a ponto

Rotinas

- Envio síncrono
- Envio bloqueante/recebimento bloqueante
- Envio não bloqueante /recebimento não bloqueante
- Envio buferizado
- Envio e recebimento combinado

Topologias Virtuais

- Mapeamento de processos MPI em um “forma” geométrica
- Cartesiano e Gráfico (Grafos)
- Construída pelos grupos e comunicadores MPI
- Deve ser implementado pelo desenvolvedor da aplicação

Obrigado!