Introdução a Programação

Paralela com OpenMP

## Aloísio Felipe

O OpenMP (do inglês Open Multi-Processing, ou Multi-processamento aberto) é uma interface de programação de aplicativo(API) para a programação multi-processo de memória compartilhada em múltiplas plataformas. Permite acrescentar simultaneidade aos programas escritos em C, C++ e Fortran sobre a base do modelo de execução fork-join. Está disponível em muitas arquiteturas, incluindo as plataforma Unix e Microsoft Windows. É constituída por um conjunto de diretivas de compilador, rotinas de biblioteca, e variáveis de ambiente que influenciam o comportamento do tempo de execução. Definido em conjunto por um grupo principal de fornecedores de hardware e de software, o OpenMP é um modelo de programação portável e escalável que proporciona aos programadores uma interface simples e flexível para o desenvolvimento de aplicações paralelas para as plataformas que vão dos computadores de escritório até os supercomputadores. Uma aplicação construída com um modelo de programação paralela híbrido pode ser executado em um cluster de computadores utilizando tanto os OpenMP e MPI, ou mais transparentemente através das extensões do OpenMP para os sistemas de memória distribuída.

Aprendendo OpenMP

Como fazer um mesmo código executar em paralelo.

Como dividir as interações de um loop entre threads.

Como replicar uma variável para várias threads.

contando de 1 a 4

O programa sequencial abaixo que conta de 1 a 4 é executado por apenas uma thread.

#include <stdio.h>  
int main()   
{  
 int i;  
  
 printf("Vamos contar de 1 a 4\n");  
   
 for(i = 1; i <= 4; i++)  
 printf("%d\n",i);  
  
}

Primeiro vou copiar o código acima para um arquivo chamado contar.c.

Vamos então compilar e executar o programa com os seguintes comandos:

$ gcc contar.c -o contar

$ ./contar

A saída esperada para esse programa é a seguinte:

Vamos contar de 1 a 4  
1  
2  
3  
4

Não importa a quantidade de núcleos do seu processador, este código sempre será executado uma vez em apenas um núcleo.

replicando o código entre múltiplas threads

Para replicar o código abaixo, podemos usar o OpenMP. Ele é uma ferramenta que auxilia na paralelização automática de código. Para fazer o mesmo trecho de código executar em todos os núcleos do seu processador, basta adicionar a diretiva **#pragma omp parallel {...}** no respectivo trecho de código. O OpenMP vai automaticamente criar várias threads (ex: em um quad-core, ele cria 4 threads) que executam exatamente o mesmo código.

#include <stdio.h>  
int main()   
{  
 #pragma omp parallel  
 {  
 int i;  
 printf("Vamos contar de 1 a 4\n");  
  
 for(i = 1; i <= 4; i++)  
 printf("%d\n",i);  
 }  
}

Vamos compilar e executar o programa com os seguintes comandos, note que é necessário adicionar uma diretiva **-fopenmp** para habilitar o uso do OpenMP:

$ gcc contar.c -o contar -fopenmp

$ ./contar

Uma saída possível seria a a seguinte:

Vamos contar de 1 a 4  
1  
2  
3  
4  
Vamos contar de 1 a 4  
1  
2  
3  
4

Ao acrecentar um parallel, o que vc espera na saída do programa? Provavelmente que ele replique o código pra todas as threads. Se vc tem um dual core, então que a contagem de 1 a 4 fosse feita duas vezes. Mas foi isso que aconteceu?

Execute o código mais 10 vezes.

É possível notar que ele gerou outras saídas diferentes. Porque? Como as duas threads executam ao mesmo tempo, **não tem como garantir qualquer ordem de execução.**

**Regra 1 do Paralelismo: Não se pode garantir a ordem de execução de threads em paralelo.**

**replicando apenas parte do código**

**Para replicar apenas a contagem de 1 a 4,basta adicionar a diretiva #pragma omp parallel antes da estrutura de repetição (for). Quando as chaves {}, ou seja, o escopo do bloco de código não é especificado, o OpenMP automaticamente paraleliza o comando logo abaixo da diretiva.**

**#include <stdio.h>  
int main()   
{  
 int i;  
  
 printf("Vamos contar de 1 a 4\n");  
  
 #pragma omp parallel  
 for(i = 1; i <= 4; i++)  
 printf("%d\n",i);  
}**

**Vamos compilar e executar o programa com os seguintes comandos, note que é necessário adicionar uma diretiva -fopenmp para habilitar o uso do OpenMP:**

**$ gcc contar.c -o contar -fopenmp**

**$ ./contar**

**Uma saída possível seria a seguinte:**

**Vamos contar de 1 a 4  
1  
2  
3  
4  
1**

**Ao acrecentar um parallel, o que vc espera na saída do programa? Provavelmente que ele imprimisse a contagem de 1 a 4 duas vezes, no caso de um dual core. Mas não foi isso que aconteceu, por que?**

**Execute o código mais 10 vezes.**

**Por que estão faltando números?**

**Quando duas ou mais threads compartilham uma mesma variável, neste exemplo a variável "i", o valor da variável pode ficar imprevisível. No exemplo acima, a thread 0 leu o valor 1 de "i" ao mesmo tempo que a thread 1, mas a thread 0 foi mais rápida e executou todas as iterações incrementando o contador "i" até 5, terminando sua execução. Quando a thread 1 decide continuar sua execução, ela imprime o valor 1 e quando vai conferir o valor de "i" na condição do loop, ela descobre que ele vale 5, também saindo do loop.**

**Regra 2 do Paralelismo: Quando threads compartilham uma mesma variável,**

**o valor da variável pode ficar imprevisível.**

**lidando com variáveis compartilhadas**

**Para resolver o problema de ter apenas um contador "i" para as threads, basta replicar o contador em cada thread. Para isso é necessário declarar o contador dentro da região paralela, ou seja, dentro da estrutura for.**

**#include <stdio.h>  
int main()   
{  
 printf("Vamos contar de 1 a 4\n");  
  
 #pragma omp parallel  
 for(int i = 1; i <= 4; i++)  
 printf("%d\n",i);  
}**

**Vamos compilar e executar o programa com os mesmos comandos de antes.**

**Uma saída possível seria a seguinte:**

**Vamos contar de 1 a 4  
1  
2  
3  
4  
1  
2  
3  
4**

**Ao declara o contador "i" dentro da região paralela, cada thread agora tem a sua cópia privada. Ou seja, não existe mais o compartilhamento de variáveis.**

**Outra forma de resolver este problema, é usar o comando private que transforma variáveis compartilhadas em variáveis privadas, como no exemplo abaixo.**

**#include <stdio.h>  
int main()   
{  
 int i;  
  
 printf("Vamos contar de 1 a 4\n");  
  
 #pragma omp parallel private(i)  
 for(i = 1; i <= 4; i++)  
 printf("%d\n",i);  
}**

**distribuindo o trabalho**

**Mas se ao invés de replicar o código, eu queira dividir o trabalho entre as threads? Para isso existe o comando #pragma omp parallel for especificamente para distribuir as iterações do for entre as threads. Ou seja, cada iteração do for é executada apenas uma vez, mas cada thread pega uma iteração diferente.**

**#include <stdio.h>  
int main()   
{  
 printf("Vamos contar de 1 a 4\n");  
  
 #pragma omp parallel for  
 for(int i = 1; i <= 4; i++)  
 printf("%d\n",i);  
}**

**Vamos compilar e executar o programa com os mesmos comandos de antes.**

**Uma saída possível seria a seguinte:**

**Vamos contar de 1 a 4  
1  
3  
4  
2**

**Execute várias vezes e você notará que a ordem pode sempre variar, mas necessariamente cada iteração ocorrerá uma única vez.**

**A maioria dos programas paralelos são feitos através da paralelização de um for, que geralmente é onde o programa gasta mais tempo e portanto deve ser paralelizado.**

**Regra 3 do Paralelismo: Estruturas de repetição são os melhores candidatos para a paralelização.**