Sistemas Operacionais 1

Threads – Parte 2 - OpenMP

Prof. Leandro Marzulo

março de 2012

Histórico

- Escassez de padrões para compartilhamento de memória
- Fórum Open MP iniciado por IBM, Digital, Intel, SGI e KAY
- Open MP para FORTRAN – 1997
- Open MP para C/C++ – 1998
- Open MP 2.0 para FORTRAN – 2000
- Open MP 2.0 para C/C++ – 2000

Compilador

- Diretiva
  - Comando que faz sentido apenas para alguns compiladores
- Sentinela
  - Caracter(es) que distingue(m) uma diretiva
  - FORTRAN – !OMP, COMP, *$OMP
  - C/C++ – #pragma omp
- Ignora a diretiva caso não reconheça a sentinela
- *Parallel OMP* se refere ao bloco (em C/C++)

Hello World

```
#include <stdio.h>
int main (void)
{
#pragma omp parallel
   {
      printf("Hello, world!\n");
    }
     return 0;
 }
```

Multiplicação de vetor por escalar

```
int main(int argc, char **argv) {
    const int N = 100000;
    int i, a[N];
    #pragma omp parallel for
    for (i = 0; i < N; i++)
        a[i] = 2 * a[i];
    return 0;
}
```

Clausulas de atributos de compartilhamento de dados

- shared(var)
- private(var) – não inicializado
- default(shared | private | none)
- firstprivate(var) – como *private* exceto ao inicializar pelo valor original
- lastprivate(var) – como *private* exceto que o valor original é atualizado depois da construção
- reduction(operation:var)

Clausulas de sincronização

- *critical section*: o código incluso será executado por somente um thread por vez e não simultaneamente executado por múltiplos threads. É frequentemente usado para proteger os dados compartilhados das condições de corrida.
- *atomic*: semelhante à *critical section*, mas informamos ao compilador para usar instruções especias de hardware para um melhor desempenho. Os compiladores podem optar por ignorar essa sugestão dos usuários e usar a *critical section* ao invés da *atomic*.
- *ordered*: o bloco estruturado é executado na ordem em que as iterações seriam executadas em um loop sequencial.
- *barrier*: cada thread espera até que todos os outros threads de um grupo tenham alcançado este ponto. Uma construção de partilha tem uma barreira de sincronização implícita no final.
- *nowait*: especifica quais threads podem completar sua instrução sem esperar todos os outros threads do grupo para concluir. Na ausência de tal cláusula acontece o mesmo da *barrier*.

Clausulas de escalonamento

- schedule(type, chunk)
  - static
  - dynamic
  - guided – chunk diminui exponencialmente

Controle do Número de Threads

- Variável de ambiente OMP_NUM_THREADS

Exemplo

```
#include <omp.h>
int main (int argc, char *argv[]) {
  int th_id, nthreads;
  #pragma omp parallel private(th_id)  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);
    #pragma omp barrier
    if ( th_id == 0 ) // #pragma omp master{
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
} } }
```