CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 25: Desempenho em OpenMP

Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

Mais Efeitos do Compartilhamento

- Exemplo: Busca pelo Máximo de um Array
  - Encontrar o valor máximo, e sua posição, num array *X[0:n-1]*

Código Serial:

```
for(i=0; i<n; i++) {
    if (x[i] > maxval ) {
        maxval = x[i];
        maxloc = i;
}
```

Saídas:
- *maxval:* valor máximo
- *maxloc:* posição do máximo

Execução Sequencial no S.Dumont

- n = 10.000.000 : *t = 11,3ms*

Código Serial:

```
for(i=0; i<n; i++) {
    if (x[i] > maxval ) {
        maxval = x[i];
        maxloc = i;
}
```

Tempo aproximado por iteração:

- *1,13 ηs*

Relógio da CPU: 2,4 GHz

~ 2,7 ciclos / iteração

Paralelização Inicial

- Loop Normal: paralelização trivial com OpenMP

```
#pragma omp parallel for
for(i=0; i<n; i++) {
    if (x[i] > maxval ) {
        maxval = x[i];
        maxloc = i;
}
```

Problema:
maxval e *maxloc* podem ser escritas por vários threads!
→ *Race condition*

Possível erro nos resultados encontrados → Paralelização inválida!

Correção da Paralelização

- Idéia: eliminar *race conditions*
  - Proteger acessos a *maxval* e *maxloc*: diretiva "*critical*"

```
#pragma omp parallel for
for(i=0; i<n; i++) {
#pragma omp critical
 {
    if (x[i] > maxval ) {
            maxval = x[i];
            maxloc = i;
  }
}
```

Efeitos no Desempenho

- Idéia: eliminar *race conditions*
  - Proteger acessos a *maxval* e *maxloc*: diretiva "*critical*"

```
#pragma omp parallel for
for(i=0; i<n; i++) {
#pragma omp critical
  {
     if (x[i] > maxval ) {
                maxval = x[i];
                maxloc = i;
   }
}
```

Tempos no Santos Dumont:

Número de Threads	Tempo (ms)
1	395
2	3.664
4	4.203

Efeitos no Desempenho (cont.)

- Overheads de OpenMP/*critical*
  - T_{serial} = 11,3ms
  - Overhead(*critical*)=395-11,3=383,7 ms
    - ~38*ηs* por iteração !
  - T crescente:
    - *critical* → serialização
  - Resultados corretos…
    - Desempenho sofrível!

Número de Threads	Tempo (ms)
1	395
2	3.664
4	4.203

Problema de Desempenho

- Raiz do problema: serialização
  - Será mesmo necessário proteger todos os acessos a *maxval* e *maxloc* , mantendo tais valores sempre atualizados?
    - Basta que sejam válidos ao final da execução!

```
#pragma omp parallel for
for(i=0; i<n; i++) {
#pragma omp critical
 {
      if (x[i] > maxval ) {
                maxval = x[i];
                maxloc = i;
  }
}
```

Idéia de Otimização:

Replicar *maxval* e *maxloc*

Uma para cada thread,

combinar resultados depois

Otimização de Desempenho

- Novo código: uma instância de *maxval* e *maxloc* por thread

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  mval[id]=0.0;
#pragma omp for
  for(i=0; i<n; i++) {
  {
      if (x[i] > mval[id] ) {
                  mval[id] = x[i];
                  mloc[id] = i;
      }
  }
```

Otimização de Desempenho

- Fase Final: combinação dos resultados de cada thread

```
#pragma omp flush(mloc,mval)
#pragma omp master
  {
      maxloc=mloc[0]; maxval=mval[0];
      for(i=1; i<nt; i++) {
              if (mval[i] > maxval) {
                      maxval = mval[i];
                      maxloc = mloc[i];
              }
      }
  }
}
```

Apesar de serializada, fase final é curta: custo reduzido

Comparação de Desempenho

- Versão Serial: 11.3 ms

Versão OMP otimizada:

Número de Threads	Tempo (ms)
1	16,7
2	9,9
4	6,0

Versão OMP com *critical*:

Número de Threads	Tempo (ms)
1	395
2	3.664
4	4.203

Exemplo-1: Sumário

- Acesso disciplinado a variáveis compartilhadas
  - Resultados gerados são corretos
  - Desempenho pode ser ruim, pela serialização
- Otimizações possíveis
  - Verificar qual/quais dado(s) de fato precisam ser compartilhados
  - Alterar ligeiramente o algoritmo se necessário
  - Ponto chave: minimizar serializações