Computação Paralela e Distribuída

Ano lectivo 2023-24

Rascunho - Será reorganizado e actualizado

Tema#03

Programação de Sistemas de Memória Partilhada

João José da Costa
joao.costa@isptec.co.ao

Coordenação de Engenharia Informática
Departamento de Engenharias e Tecnologias
Instituto Superior Politécnico de Tecnologias e Ciências

Optimização

Análise de Desempenho e Depuração de Programas Paralelos

Objectivos

Instrutivo

- Testar e analisar o programa paralelo baseado em arquitectura UMA.

Educativo

- Sentir a necessidade de testar e analisar o programa paralelo para correcção e melhor desempenho.

Tópicos

- Desempenho
- Depuração

Factores que impactam o desempenho

- **Desempenho do algoritmo e sua implementação**
- **Fracção do Código que é executado em paralelo**
- **Gestão de threads**
  - Overhead para criar, resumir, gerir, suspender e destruir
  - Quantidade de sincronização
- **Organização de dado**
  - Localidade de dado
  - Dependência de dado
  - Conflitos de memória
  - Partilha de dado efectiva
- **Balanceamento de carga**

Regras de eficiência OpenMP

Optimização para escalabilidade e desempenho:

- ✓ **Minimizar join/forks**
- ✓ **Minimizar sincronização**
- ✓ **Maximizar dado privado/independente**

Dado partilhado e privado

- ✓ **Utilizar dado privado ou dado independente para cada thread, onde possível**
- ✓ **Independência de dado não é um problema para dado apenas de leitura.**
- ✓ **Não é suficiente que duas threads acessem a mesma posição de memória para garantir independência de dado!**
- ❖ **Linhas de cache**

Fusão de ciclos

```
for(i = 0; i < n; i++)
  a[i] = sqrt(1.0 + i*i);
      sum = 0.0;
for(i = 0; i < n; i++)
     sum + = a[i];
```

- ✓ Barreira implícita no fim das secções paralelas
- ✓ Melhor programa serial: fusão promove pipelining de software e reduz a frequência de ramificações
- ✓ Melhor programa OpenMP: fusão reduz a sincronização e overhead de escalonamento
- ✓ Promover fusão: reordenar o Código para obter ciclos que não estão separados por declarações que criam dependência de dados

Multiplicação de matriz

```
for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            for(k = 0; k < n; k++)
                c[i][j] += a[i][k] * b[k][j];
```

Implementação paralela correcta e eficiente??????

Multiplicação de matriz

```
#pragma omp parallel private(i,j,k)
for(i = 0; i < n; i++)
          for(j = 0; j < n; j++)
                #pragma omp for
               for(k = 0; k < n; k++)
                    c[i][j] += a[i][k] * b[k][j];
```

Solução incorrecta! Corrida de dado em c

Multiplicação de matriz

```
#pragma omp parallel private(i,j,k) reduction(+:c)
for(i = 0; i < n; i++)
          for(j = 0; j < n; j++)
              #pragma omp for
              for(k = 0; k < n; k++)
                  c[i][j] += a[i][k] * b[k][j];
```

Correcto, mas extremamente ineficiente!

Enorme desperdício de memória, c é replicado.

Multiplicação de matriz

```
for(i = 0; i < n; i++)
          for(j = 0; j < n; j++)
              #pragma omp parallel for reduction(+:c[i][j])
               for(k = 0; k < n; k++)
                  c[i][j] += a[i][k] * b[k][j];
```

Correcto, mas baixa eficiência, muitos fork-joins.

Multiplicação de matriz

```
#pragma omp parallel private(i,j,k)
for(i = 0; i < n; i++)
          for(j = 0; j < n; j++) {
              acc = 0;
              #pragma omp for reduction(+:acc)
               for(k = 0; k < n; k++)
                   acc += a[i][k] * b[k][j];
          #pragma omp single
              c[i][j] = acc;
     }
```

Correcto, mas continua com baixa eficiência.
Também muitos pontos de sincronização.

Multiplicação de matriz

```
#pragma omp parallel for private(i,j,k)
for(i = 0; i < n; i++)
          for(j = 0; j < n; j++)
                    for(k = 0; k < n; k++)
                              c[i][j] += a[i][k] * b[k][j];
```

Correcto e bom desempenho!

Compensações

Compensações na paralelização

- ✓ para aumentar a fração paralela de trabalho ao paralelizar loops, é melhor paralelizar o loop mais externo de um loop aninhado;

- ✓ fazer isso pode exigir transformações de loop, como intercâmbios de loop, que podem destruir a eficiência da cache devido ao padrão de memória de acesso;

- ✓ escalonamento de loop estático em grandes pedaços por thread promove localidade de cache e página, mas pode não atingir balanceamento de carga;

- ✓ escalonamento dinâmico e intercalado alcança bom balanceamento de carga, mas causa baixa localidade de referências de dados.

Profiling

Criação de perfil

- ✓ Fornece estatísticas resumidas de métricas de desempenho
- - número de vezes que uma rotina foi invocada, tempo exclusivo e inclusivo, estatísticas de desempenho de hardware, número de rotinas filhas invocadas, árvore de chamadas e gráfico de chamadas, etc.
- ✓ Ajuda a expor gargalos (bottlenecks) de desempenho e pontos de acesso (hotspots)
- - Optimize o que importa, não perca tempo optimizando coisas que têm influência geral insignificante no desempenho.

Profiling

Criação de perfil

Implementações

- ✓ **Sampling/Amostragem**: interrupções periódicas do sistema operativo ou armadilhas (traps) do contador de hardware
- Construa um histograma de valores de contador de programa amostrados.

- ✓ **Medição**: inserção directa do código de medição
- As medições são feitas no início e no final das regiões de interesse e, em seguida, calculam a diferença.

```
---------------------------------------------------------------------
----     ompP Flat Region Profile (inclusive data)     ---------------
---------------------------------------------------------------------
R00010 rndcnf.F (48-52) PARALLEL LOOP
 TID      execT      execC      bodyT   exitBarT   startupT   shutdwnT
   0       0.33          1       0.33       0.00       0.00       0.00
   1       0.33          1       0.32       0.01       0.00       0.00
   2       0.33          1       0.33       0.00       0.00       0.00
   3       0.33          1       0.32       0.01       0.00       0.00
 SUM       1.32          4       1.29       0.02       0.01       0.00
...
```

Desafios

Teste de programas

Para programas sequenciais:

- criar várias entradas de teste com respostas conhecidas
- execute o código em cada entrada de teste
- se todos os testes derem saída correcta, tenha alguma confiança no programa
- ter intuição sobre quais “casos extremos” testar

Mas para programas paralelos:

- cada execução testa apenas um único escalonamento
- como podemos testar vários escalonamentos diferentes?
- quão confiantes podemos estar quando nossos testes passarem?

Desafios

Depuração do programa

Um programa paralelo produz respostas correctas se satisfizer dois critérios:

- **ausência de não determinismo** - sempre produz as mesmas respostas nas mesmas entradas
- **equivalência à versão sequencial** - produz as mesmas respostas que o programa sequencial no qual se baseia

Assumindo:

- programa paralelo desenvolvido pela produção e depuração da versão sequencial
- não determinismo não é uma propriedade desejável

Tipos de erro

- Uso incorreto de recursos de linguagem
- Decomposição Espacial
- Sincronização
- Efeito colateral da paralelização

Tipos de erro

Uso incorreto de recursos de linguagem

Sintomas:

- Erro de compilação (fácil de corrigir)
- Alguns defeitos podem surgir apenas sob condições específicas
  - Número de processadores, valor da entrada, problemas de alinhamento

Causa:

- Falta de experiência com a sintaxe e semântica de novos recursos de linguagem

Prevenção:

- Verifique cuidadosamente os recursos desconhecidos da linguagem

Tipos de erro

Decomposição Espacial

Sintomas:

- Falha de segmentação (se o índice da matriz estiver fora do intervalo)
- Saída incorrecta

Causa:

- O mapeamento na versão paralela pode ser diferente da versão serial
- A origem do array é diferente em cada processador
- Variáveis privadas x partilhadas

Prevenção:

- Valide a alocação de memória com cuidado ao paralelizar o código

Tipos de erro

Sincronização

Sintomas:

- Programa trava
- Saída incorreta/não determinística

Causa:

- Alguns defeitos podem ser muito sutis
- Bloqueios de dados aninhados

Prevenção:

- Certifique-se de que a comunicação da thread esteja coordenada correctamente

Tipos de erro

Efeito colateral da paralelização

Sintomas:

- Vários problemas de correção e desempenho

Causa:

- Parte sequencial do código é ignorada
- Programas paralelos típicos contêm apenas algumas primitivas paralelas e o restante do código é um programa sequencial executado várias vezes

Prevenção:

- Não se concentre apenas no código paralelo
- Verifique se o código serial está a funcionar em um processador, mas lembre-se de que o defeito pode surgir apenas em um contexto paralelo

Tipos de erro

Efeito colateral da paralelização

Inicialização da seed aleatória:

```
srand(time(NULL));
    for (x = 0; x < nlocal; x++)
        buffer[x+1] = rand() % 10;
```

Qual é o risco?

Solução:

```
srand(time(NULL) + omp_get_thread_num());
    for (x = 0; x < nlocal; x++)
        buffer[x+1] = rand() % 10;
```

Ferramentas

Ferramentas para depuração e criação de perfil

Depuradores comerciais:

- TotalView, Distributed Debugging Tool (DDT)

GDB

printf

Escalonamento

Testes de estresse

Execute cada teste várias vezes:

- não é um bom teste: não há muita aleatoriedade no escalonamento do sistema operativo

Uma solução: subscrever a máquina

- em um sistema de 4 núcleos, execute com 8 ou 16 threads
- executar várias instâncias do programa ao mesmo tempo
- aumentar o tamanho para estourar o cache/memória

Assim, o tempo das threads mudará, dando diferentes escalonamento de threads

Escalonamento

Produção de Ruído / Escalonamento Aleatório

usar escalonamento de thread aleatórios:

- por exemplo, insira um código como:

```
if (rand() < 0.01) usleep(100);
if (rand() < 0.01) yield();
```

adicioná-lo manualmente ao código "suspeito" ou

algumas ferramentas fazem isso automaticamente

- IBM’s ConTest, Thrille, CalFuzzer, ...

Corrida de dados

Detecção/previsão de corrida de dados

Uma corrida de dados ocorre quando duas threads acessam simultaneamente a mesma memória e pelo menos uma é uma escrita.

Happens-Before Race Detection [Schonberg ’89]:

- Ocorrem dois acessos a uma variável, pelo menos um por escrita, sem nenhuma sincronização interveniente?
- Sem avisos falsos

Lockset Race Prediction [Savage, et al., '97]:

- Todo acesso a uma variável contém um bloqueio comum?
- Eficiente, mas muitos falsos avisos

Hybrid Race Prediction [O’Callahan, Choi, 03]

- Combina Lockset com Happens-Before para melhor desempenho e menos avisos falsos vs Lockset

Corrida de dados

Cobertura x Avisos Falsos

Falso aviso: a ferramenta relata uma corrida de dados, mas a corrida não pode acontecer em uma corrida real

Cobertura: quantas corridas de dados reais uma ferramenta relata?

Previsão de corrida híbrida:

- Melhor cobertura, mas mais falsos avisos

Acontece antes da detecção de corrida:

- Menos falsos avisos (ainda alguns, na prática) e menos cobertura

Corrida de dados

Ferramentas de corrida de dados dinâmicos

Intel Thread Checker para pthreads:

- Detecção de corrida Happens-Before

Ferramentas baseadas em Valgrind para pthreads:

- Helgrind e DRD (Happens-Before)
- ThreadSanitizer (Hybrid)

CHESS realiza detecção de corrida para .NET

Revisão

- Desempenho
- Depuração

Bibliografia

- Consulte dentro da subpasta “references” no repositório da disciplina.

Próxima aula

Mais sobre programação de sistemas de memória partilhada:

- Monitores vs Mutexes
- Programação paralela em Java
- Memória transacional de software