Arquiteturas Paralelas e Distribuídas

Compiladores paralelizadores

Eduardo Furlan Miranda

Abordagens para Explorar o Paralelismo

- Paralelismo Implícito
  - Mantém a sintaxe da codificação sequencial
  - O compilador traduz o código sequencial para sua forma paralela automaticamente
  - Oculta a complexidade do paralelismo ao programador
  - Compiladores mais complexos
  - Limitações na detecção automática de paralelismo
- Paralelismo Explícito :
  - Linguagens ampliadas com construtores específicos para paralelismo, ex.: OpenMP, MPI
  - Ou criação de novas linguagens, ex.: HPF, Linda
  - O programador especifica o que pode/deve ser executado em paralelo
  - Maior controle sobre o desempenho
  - Exige conhecimento profundo do algoritmo e da arquitetura

Compiladores vs. Paralelização

- Linguagens para Programação Paralela
  - Declarativas (Lógicas e Funcionais):
    - Paralelismo de granulação fina
    - Explorado naturalmente por compiladores específicos
    - Exemplos: Sisal, Val, Haskell
  - Imperativas:
    - Compiladores Paralelizadores:
      - Geram versões paralelas de programas sequenciais
      - Exigem pouco conhecimento extra do usuário
      - Desempenho modesto, mas útil para certas aplicações
      - Exemplos: Oxygen, OSCAR, PARADIGM, SUIF, Parafrase2, Cray
    - Extensões Paralelas (Bibliotecas):
      - Usam primitivas da linguagem hospedeira
      - Não exigem aprendizado de nova linguagem
      - Melhor desempenho que compiladores paralelizadores
      - Exemplos: PVM, MPI

Compiladores Paralelizadores

- Desafio Principal
  - Detectar paralelismo implícito em sequências de instruções
  - Requer análise de dependências de dados entre instruções
- Condições para Paralelismo
  - Se não houver dependências de dados, as instruções podem ser executadas simultaneamente
  - Se houver dependências, o compilador realiza otimizações
    - Rearranjo de índices em loops
    - Remoção de operações desnecessárias dentro de loop

Compiladores Paralelizadores

- Compiladores Paralelizadores
  - Laços de repetição (loops) em códigos sequenciais são o foco principal
- Tarefas do Compilador Paralelizador
  - Identificar regiões com potencial de paralelismo
  - Mapear o paralelismo na arquitetura da máquina alvo
  - Gerar e otimizar código paralelo

Técnicas e Exemplos de Compiladores Paralelizadores

- Arquiteturas e Técnicas
  - Arquiteturas Vetoriais
    - Compiladores vetorizadores convertem loops em instruções vetoriais
    - Operações são executadas em pipeline
    - Desempenho aumenta com o número de loops convertidos
  - Arquiteturas Multiprocessadas
    - Compiladores particionam instruções de loops entre processadores
    - Técnicas de paralelização e vetorização podem ser combinadas
- Exemplos de Compiladores Paralelizadores/Vetorizadores:
  - Oxigen, OSCAR, PARADIGM (Fortran)
  - SUIF (Fortran e C)
  - Parafrase2 (C)

Análise de Dependências de Dados

- Identificar dependências de dados para detectar paralelismo implícito e otimizar programas sequenciais
  - Dependência Verdadeira (RAW - Read After Write)
    - Exemplo: A = B + C → D = A + 2
      - A instrução 2 depende do valor de A gerado na instrução 1
  - Antidependência (WAR - Write After Read)
    - Exemplo: A = B + C → B = D / 2
      - A instrução 1 usa o valor antigo de B antes de ser atualizado na 2
  - Dependência de Saída (WAW - Write After Write)
    - Exemplo: A = B + C → A = E + F
      - A ordem de execução afeta o valor final de A
  - Dependência de Controle:
    - Ocorre em desvios condicionais
      - Exemplo: if (X >= 0) then A = A + 2

Dependências em Laços de Repetição

- Dependência RAW entre as instruções 1 e 2 na mesma iteração

```
do I = 2, N
  A(I) = B(I) + C(I)
  D(I) = A(I)
end do
```

- Dependência RAW entre iterações (valor de A da iteração anterior)

```
do I = 2, N
  A(I) = B(I) + C(I)
  D(I) = A(I-1)
end do
```

- Dependência WAR (antidependência) entre iterações

```
do I = 2, N
   A(I) = B(I) + C(I)
   D(I) = A(I+1)
end do
```

Vetorização de Instruções

- Arquiteturas Vetoriais
  - Operações executadas em pipeline
  - Compiladores vetorizadores convertem loops em instruções vetoriais

- Código original

```
do I = 1, N
   A(I) = B(I+2) + C(I+1)
end do
```

- Código vetorizado
  - A(1:N) = B(3:N+2) + C(2:N+1)

- Dependências de dados exigem reordenação de instruções

```
do I = 1, N
   A(I) = B(I)
   C(I) = A(I) + B(I)
   E(I) = C(I+1)
end do
```

Paralelização de Instruções

- Distribuir iterações de loops entre processadores em arquiteturas multiprocessadas
  - Iterações Independentes. Paralelização possível no laço externo (I)

```
do I = 1, N
  do J = 2, N
    A(I,J) = B(I,J) + C(I,J)
    C(I,J) = D(I,J) / 2
    E(I,J) = A(I,J-1)**2 + E(I,J-1)
  end do
end do
```

- Iterações Dependentes. Sincronização necessária devido à dependência entre iterações

```
do I = 2, N
   A(I) = B(I) + C(I)
   C(I) = D(I) * 2
   E(I) = C(I) + A(I-1)
end do
```

- Escalonamento em Blocos Contínuos
  - as iterações do loop são divididas em blocos contínuos, onde cada processador recebe um conjunto sequencial de iterações

- Auto-Escalonamento (Self-Scheduling)
  - as iterações são atribuídas dinamicamente aos processadores conforme eles ficam disponíveis

Linguagem Linda: Programação Paralela Baseada em Espaço de Tuplas

- Modelo de programação paralela baseado em espaço de tuplas (tuple space)
- Permite comunicação e sincronização entre processos de forma assíncrona
- Não depende de uma linguagem específica; pode ser integrada a várias linguagens (ex.: C, Fortran)
- Insere/remove/lê uma tupla do espaço compartilhado
- Foco em comunicação indireta e coordenação implícita
- Ideal para sistemas distribuídos e ambientes heterogêneos

Linguagem Java: Paralelismo Explícito com Threads e Concorrência

- Java oferece suporte nativo para programação paralela através de threads e bibliotecas de concorrência
- A linguagem é amplamente usada em sistemas distribuídos e multicore
- Possui recursos para threads, concorrência, e memória compartilhada
- Facilidade de uso com APIs bem documentadas
- Grande ecossistema e suporte para frameworks paralelos (ex.: Hadoop, Spark)

Linguagem HPF (High-Performance Fortran)

- Extensão da linguagem Fortran para programação paralela em sistemas de memória distribuída

- Principais Características
  - Diretivas de Paralelismo
    - Anotações no código (ex.: DISTRIBUTE, ALIGN) guiam o compilador na distribuição de dados e tarefas
  - Modelo de Dados Distribuídos
    - Arrays são divididos entre processadores para processamento paralelo
  - Foco em Granularidade Grossa
    - Adequado para problemas que envolvem grandes conjuntos de dado
  - Mantém a sintaxe familiar do Fortran
  - Alta performance em sistemas de memória distribuída