Compiling for High Performance

### Forms of Parallelism

- **Instruction-level parallelism**
  - Very fine grain parallelism
  - Evaluate dependences between statements
  - For superscalar and VLIW architectures

- **Task-level parallelism**
  - Very coarse grain parallelism
  - Evaluate dependences between tasks
  - For multiprocessors

- **Loop-level parallelism**
  - Evaluate dependences between loop iterations
  - For vector machines and multiprocessors

- **Parallelism is not scalable**

- **Issue**
  - Dependence analysis
  - Loop transformations
  - Memory access patterns
  - Array variables
  - Structured code

- **Loop-level parallelism**
  - Focus on aggregate operations (loops)

- **High-performance compilation**
  - Unstructured code
  - Flow of values
  - Scalar variables
  - Focus on individual operations

- **Task-level parallelism**
  - Locality
  - Parallelism

- **Instruction-level parallelism**
  - Locality
  - Parallelism

Compiling for High Performance

```plaintext
V(1) = V(1) + 1
do i = 1, 10
    A = A + 1
    B = B + 1
end do
```
Loop-level Parallelism

Basic approach

- execute loop iterations in parallel
- safe if no loop-carried data dependences (i.e., no accesses to same memory location)
  
  \[
  \begin{align*}
  &\text{do } i = 1,10 &\text{do all } i = 1,10 \\
  &A(i) = A(i+1) &A(i) = A(i+10)
  \end{align*}
  \]

Several parallel architectures

- vector processors
  \[
  \]

- multiprocessors
  
  \[
  \begin{align*}
  &\text{do all } i = 1,10 \\
  &A(i) = B(i+1)
  \end{align*}
  \]

- message-passing machines
  
  \[
  \begin{align*}
  &\text{if } \ldots \text{ send } B(1) \\
  &\text{if } \ldots \text{ recv } B(11) \\
  &\text{do } i = L,B \\
  &A(i) = B(i+1)
  \end{align*}
  \]

Which Loops are Parallel?

\[
\begin{align*}
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_1 & A(I,J) = A(I,J-1) \\
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_2 & A(I,J) = A(I-1,J-1) \\
\text{do } I = 1, N \\
\text{do } J = 1, N \\
S_3 & B(I,J) = B(I-1,J+1)
\end{align*}
\]

- a dependence \( D = (d_1, \ldots, d_k) \) is carried at level \( i \), if \( d_i \) is the first nonzero element of the distance vector

- a loop \( l \) is parallel, if \( \forall \) a dependence \( D_j \) carried at level \( i \)

\[
\begin{array}{c|c}
\text{distance vector} \\
\forall D_j & d_1, \ldots, d_{i-1} > 0 \\
\text{OR} & d_1, \ldots, d_i = 0
\end{array}
\]
Exploiting Parallelism

Data
- Convert conditionals into explicit control flow
- Coalesce loops to reduce overhead

Scalar Analysis
- Improve precision of dependence tests
- Eliminate unnecessary scalar statements

Solution Techniques
- Forward propagation
- Constant propagation
- Induction variable recognition

Vectorization
- Express in vector languages (APL, Fortran 90)
- Express in vector language

Execution Model
- Single thread of control
- Single instruction, multiple data (SIMD)

Issues
- High-level operations on vectors of data
- Multithreading and parallelism

Vector Processors
- Overlap iterations of inner loop
- Operations on vectors of data

Vectorization

C MSC

Lecture 22, Page 5
Parallelization

Multithreading

High latency event

Multithreading

Execution model:

\[ \forall (i, j) \in \mathbb{A}, \quad 0 \leq t = i \]
\[ 0 \leq d = j \leq 10, \quad \nu = \text{assign iterations to different processors (MIMD)} \]

Hardware support

- requires threads, efficient context switch
- overlaps computation with event
- switch to new thread after event

Multiple threads of execution

- cache miss
- page miss
- interprocessor communication
- I/O

Software support

- HEF, Tier
- shared caches
- switch sets of registers

Issues

- granularity - larger computation partitions to examine
- scheduling - policy for assigning iterations to processors
- reduce overhead

Multithreading

- uncover parallelism for multiple threads
- reduce context switch overhead

Software support

- HEF, Tier

Execution model:

\[ \forall (i, j) \in \mathbb{A}, \quad 0 \leq t = i \]
\[ 0 \leq d = j \leq 10, \quad \nu = \text{assign iterations to different processors (MIMD)} \]

Multithreading

- uncover parallelism for multiple threads
- reduce context switch overhead

Software support

- HEF, Tier

Execution model:

\[ \forall (i, j) \in \mathbb{A}, \quad 0 \leq t = i \]
\[ 0 \leq d = j \leq 10, \quad \nu = \text{assign iterations to different processors (MIMD)} \]

Multithreading

- uncover parallelism for multiple threads
- reduce context switch overhead

Software support

- HEF, Tier

Execution model:

\[ \forall (i, j) \in \mathbb{A}, \quad 0 \leq t = i \]
\[ 0 \leq d = j \leq 10, \quad \nu = \text{assign iterations to different processors (MIMD)} \]

Multithreading

- uncover parallelism for multiple threads
- reduce context switch overhead

Software support

- HEF, Tier

Execution model:

\[ \forall (i, j) \in \mathbb{A}, \quad 0 \leq t = i \]
\[ 0 \leq d = j \leq 10, \quad \nu = \text{assign iterations to different processors (MIMD)} \]

Multithreading

- uncover parallelism for multiple threads
- reduce context switch overhead

Software support

- HEF, Tier