Enhancing Fine-Grained Parallelism

Chapter 5 of Allen and Kennedy
Fine-Grained Parallelism

Techniques to enhance fine-grained parallelism:

• Loop Interchange
• Scalar Expansion
• Scalar Renaming
• Array Renaming
• Node Splitting
Can we do better?

- Codegen: tries to find parallelism using transformations of loop distribution and statement reordering
- If we deal with loops containing cyclic dependences early on in the loop nest, we can potentially vectorize more loops
- Goal in Chapter 5: To explore other transformations to exploit parallelism
Motivational Example

DO J = 1, M
  DO I = 1, N
    T = 0.0
    DO K = 1, L
      T = T + A(I,K) * B(K,J)
    ENDDO
    C(I,J) = T
  ENDDO
ENDDO

`codegen` will not uncover any vector operations. However, by scalar expansion, we can get:

DO J = 1, M
  DO I = 1, N
    T$(I) = 0.0
    DO K = 1, L
      T$(I) = T$(I) + A(I,K) * B(K,J)
    ENDDO
    C(I,J) = T$(I)
  ENDDO
ENDDO
Motivational Example

DO J = 1, M
  DO I = 1, N
    T$(I) = 0.0
    DO K = 1, L
      T$(I) = T$(I) + A(I,K) * B(K,J)
    ENDDO
    C(I,J) = T$(I)
  ENDDO
ENDDO
Motivational Example II

• Loop Distribution gives us:

\[
\begin{align*}
\text{DO } & J = 1, M \\
& \text{DO } I = 1, N \\
& \quad T$(I) = 0.0 \\
& \text{ENDDO} \\
& \text{DO } I = 1, N \\
& \quad \text{DO } K = 1, L \\
& \quad \quad T$(I) = T$(I) + A(I,K) \times B(K,J) \\
& \quad \text{ENDDO} \\
& \text{ENDDO} \\
& \text{DO } I = 1, N \\
& \quad C(I,J) = T$(I) \\
& \text{ENDDO} \\
& \text{ENDDO}
\end{align*}
\]
Motivational Example III

Finally, interchanging $I$ and $K$ loops, we get:

\[
\begin{align*}
&\text{DO } J = 1, M \\
&T$(1:N) = 0.0 \\
&\quad \text{DO } K = 1, L \\
&T$(1:N) = T$(1:N) + A(1:N,K) \times B(K,J) \\
&\quad \text{ENDDO} \\
&C(1:N,J) = T$(1:N) \\
&\text{ENDDO}
\end{align*}
\]

- A couple of new transformations used:
  - Loop interchange
  - Scalar Expansion
Loop Interchange

DO I = 1, N
    DO J = 1, M
        S       A(I,J+1) = A(I,J) + B
        ENDDO
    ENDDO

• Applying loop interchange:

    DO J = 1, M
        DO I = 1, N
            S       A(I,J+1) = A(I,J) + B
             ENDDO
        ENDDO

• leads to:

    DO J = 1, M
        S       A(1:N,J+1) = A(1:N,J) + B
        ENDDO

• DV: (=, <)

• DV: (<, =)

Optimizing Compilers for Modern Architectures
Loop Interchange

• Loop interchange is a reordering transformation

• Why?
  — Think of statements being parameterized with the corresponding iteration vector
  — Loop interchange merely changes the execution order of these statements.
  — It does not create new instances, or delete existing instances

DO J = 1, M
  DO I = 1, N
    S <some statement>
  ENDDO
ENDDO

• If interchanged, S(2, 1) will execute before S(1, 2)
Loop Interchange: Safety

• Safety: not all loop interchanges are safe

```
DO I = 1, M
    DO J = 1, N
        A(I,J+1) = A(I+1,J) + B
    ENDDO
ENDDO
```

• Direction vector (<, >)

• If we interchange loops, we violate the dependence
Loop Interchange: Safety

• Theorem 5.1 Let $D(i,j)$ be a direction vector for a dependence in a perfect nest of $n$ loops. Then the direction vector for the same dependence after a permutation of the loops in the nest is determined by applying the same permutation to the elements of $D(i,j)$.

• The direction matrix for a nest of loops is a matrix in which each row is a direction vector for some dependence between statements contained in the nest and every such direction vector is represented by a row.
DO I = 1, N
    DO J = 1, M
        DO K = 1, L
            A(I+1,J+1,K) = A(I,J,K) + A(I,J+1,K+1)
        ENDDO
    ENDDO
ENDDO

• The direction matrix for the loop nest is:

\[
\begin{bmatrix}
< & < & = \\
< & = & > \\
\end{bmatrix}
\]

• Theorem 5.2 A permutation of the loops in a perfect nest is legal if and only if the direction matrix, after the same permutation is applied to its columns, has no "\(>\)" direction as the leftmost non-"\(=\)" direction in any row.

• Follows from Theorem 5.1 and Theorem 2.3
DO I = 1, N
S1  T = A(I)
S2  A(I) = B(I)
S3  B(I) = T
ENDDO

• Scalar Expansion:
  DO I = 1, N
  S1  T$(I) = A(I)
  S2  A(I) = B(I)
  S3  B(I) = T$(I)
  ENDDO
  T = T$(N)

• leads to:
  S1  T$(1:N) = A(1:N)
  S2  A(1:N) = B(1:N)
  S3  B(1:N) = T$(1:N)
  T = T$(N)
Scalar Expansion: Safety

- Scalar expansion is always safe
- When is it profitable?
  - Naïve approach: Expand all scalars, vectorize, shrink all unnecessary expansions.
  - However, we want to predict when expansion is profitable
Scalar Expansion: Drawbacks

- Expansion increases memory requirements

- Solutions:
  - Expand in a single loop
  - Forward substitution:
    
    \[
    \text{DO } I = 1, N \\
    T = A(I) + A(I+1) \\
    A(I) = T + B(I) \\
    \text{ENDDO}
    \]

    \[
    \text{DO } I = 1, N \\
    A(I) = A(I) + A(I+1) + B(I) \\
    \text{ENDDO}
    \]
Scalar Renaming

DO I = 1, 100
S1  T = A(I) + B(I)
S2  C(I) = T + T
S3  T = D(I) - B(I)
S4  A(I+1) = T * T
ENDDO

• Renaming scalar T:
DO I = 1, 100
S1  T1 = A(I) + B(I)
S2  C(I) = T1 + T1
S3  T2 = D(I) - B(I)
S4  A(I+1) = T2 * T2
ENDDO
Scalar Renaming

• will lead to:

S₃ \[ T₂$(1:100) = D(1:100) - B(1:100) \]

S₄ \[ A(2:101) = T₂$(1:100) \times T₂$(1:100) \]

S₁ \[ T₁$(1:100) = A(1:100) + B(1:100) \]

S₂ \[ C(1:100) = T₁$(1:100) + T₁$(1:100) \]

T = T₂$(100)