COMP 515: Advanced Compilation for Vector and Parallel Processors

Prof. Krishna Palem
Prof. Vivek Sarkar
Department of Computer Science
Rice University
{palem,vsarkar}@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP515

Acknowledgments

- Slides from previous offerings of COMP 515 by Prof. Ken Kennedy
  - http://www.cs.rice.edu/~ken/comp515/

Enhancing Fine-Grained Parallelism (contd)

Chapter 5 of Allen and Kennedy

Array Renaming

```
     DO I = 1, N
S1       A(I) = A(I-1) + X
S2       Y(I) = A(I) + Z
S3       A(I) = B(I) + C
     ENDDO
```

- $S_1\ \delta_\infty\ S_2$ $\quad$ $S_2\ \delta_\infty^{-1}\ S_3$ $\quad$ $S_3\ \delta_1\ S_1$ $\quad$ $S_1\ \delta_\infty^{0}\ S_3$
- **Rename A(I) to A'(I):**

```
     DO I = 1, N
S1       A'(I) = A(I-1) + X
S2       Y(I) = A'(I) + Z
S3       A(I) = B(I) + C
     ENDDO
```

- **Dependences remaining:** $S_1\ \delta_\infty\ S_2$ **and** $S_3\ \delta_1\ S_1$

Array Renaming: Profitability

- Examining dependence graph and determining minimum set of critical edges to break a recurrence is NP-complete!
- Solution: determine edges that are removed by array renaming and analyze effects on dependence graph
- procedure array_partition:
  - Assumes no control flow in loop body
  - Identifies collections of references to arrays which refer to the same value
  - Identifies deletable output dependences and antidependences
- Use this procedure to generate code
  - Minimize amount of copying back to the "original" array at the beginning and the end

Seen So Far...

- Uncovering potential vectorization in loops by
  - Loop Interchange
  - Scalar Expansion
  - Scalar and Array Renaming
- Safety and Profitability of these transformations

What's next ...

- More transformations to expose more fine-grained parallelism
  - Node Splitting
  - Recognition of Reductions
  - Index-Set Splitting
  - Run-time Symbolic Resolution
  - Loop Skewing

Today's lecture

Next lecture

- Unified framework to generate vector code
- Note: these transformations are useful for generating other forms of parallel code as well (beyond vector)

Node Splitting

- Sometimes Renaming fails

```
DO I = 1, N
  S1:       A(I) = X(I+1) + X(I)
  S2:       X(I+1) = B(I) + 32
ENDDO
```

- Recurrence kept intact by renaming algorithm

Node Splitting

```
DO I = 1, N
S1:   A(I) = X(I+1) + X(I)

S2:   X(I+1) = B(I) + 32
ENDDO
```

- Break critical antidependence
- Make copy of read from which antidependence emanates

```
DO I = 1, N
S1': X$(I) = X(I+1)

S1: A(I) = X$(I) + X(I)

S2: X(I+1) = B(I) + 32
ENDDO
```

- Recurrence broken
- Vectorized to

```
S1':  X$(1:N) = X(2:N+1)
S2:   X(2:N+1) = B(1:N) + 32
S1:   A(1:N) = X$(1:N) + X(1:N)
```

Node Splitting Algorithm

- Takes a constant loop independent antidependence D
- Add new assignment x: T$=source(D)
- Insert x before source(D)
- Replace source(D) with T$
- Make changes in the dependence graph

Node Splitting: Profitability

- Not always profitable
- For example

```
DO I = 1, N
S1: A(I) = X(I+1) + X(I)

S2: X(I+1) = A(I) + 32
ENDDO
```

- Node Splitting gives

```
DO I = 1, N
S1':X$(I) = X(I+1)

S1:     A(I) = X$(I) + X(I)

S2:     X(I+1) = A(I) + 32
ENDDO
```

- Recurrence still not broken
- Antidependence was not critical

Node Splitting

- Determining minimal set of critical antidependences is NP-hard
- Perfect job of Node Splitting difficult
- Heuristic:
  - Select antidependences
  - Delete it to see if acyclic
  - If acyclic, apply Node Splitting

Recognition of Reductions

- Sum Reduction, Min/Max Reduction, Count Reduction
- Vector ---> Single Element

```
S = 0.0
DO I = 1, N
    S = S + A(I)
ENDDO
```

- Not directly vectorizable

Recognition of Reductions

- Assuming commutativity and associativity

```
S = 0.0
DO k = 1, 4
  SUM(k) = 0.0
  DO I = k, N, 4
    SUM(k) = SUM(k) + A(I)
  ENDDO
  S = S + SUM(k)
ENDDO
```

- Distribute `k` loop

```
S = 0.0
DO k = 1, 4
   SUM(k) = 0.0
ENDDO
DO k = 1, 4
   DO I = k, N, 4
        SUM(k) = SUM(k) + A(I)
   ENDDO
ENDDO
DO k = 1, 4
   S = S + SUM(k)
ENDDO
```

Recognition of Reductions

- **After Loop Interchange**

```
DO I = 1, N, 4
  DO k = I, min(I+3,N)
    SUM(k-I+1) = SUM(k-I+1) + A(I)
  ENDDO
ENDDO
```

- **Vectorize**

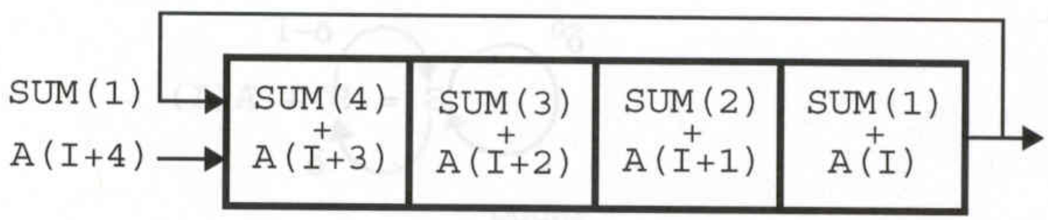
```
DO I = 1, N, 4
  SUM(1:4) = SUM(1:4) + A(I:I+3)
ENDDO
```

Recognition of Reductions

- Useful for vector machines with 4 stage pipeline, and fine-grain SIMD parallelism on modern processors (MMX, Altivec)

- Recognize Reduction and Replace by the efficient version

Pipeline for Sum Reduction.



Recognition of Reductions

- Properties of Reductions
  - Reduce Vector/Array to one element
  - No use of Intermediate values
  - Reduction operates on vector and nothing else
- Reduction recognized by
  - Presence of self true, output and anti dependences
  - Absence of other true dependences

Index-set Splitting

- **Subdivide loop into different iteration ranges to achieve partial parallelization**
  - **Threshold Analysis** [Strong SIV, Weak Crossing SIV]
  - **Loop Peeling** [Weak Zero SIV]
  - **Section Based Splitting** [Variation of loop peeling]

Threshold Analysis

```
DO I = 1, 20
     A(I+20) = A(I) + B
ENDDO
```

Vectorize to..

```
A(21:40) = A(1:20) + B
```

```
DO I = 1, 100
     A(I+20) = A(I) + B
ENDDO
```

Strip mine to..

```
DO I = 1, 100, 20
   DO i = I, I+19
     A(i+20) = A(i) + B
   ENDDO
ENDDO
```

Vectorize this

Threshold Analysis

- **Crossing thresholds**

```
DO I = 1, 100
     A(100-I) = A(I) + B
ENDDO

Strip mine to…
DO I = 1, 100, 50
   DO i = I, I+49
     A(101-i) = A(i) + B
   ENDDO
ENDDO

Vectorize to…
DO I = 1, 100, 50
     A(101-I:51-I) = A(I:I+49) + B
ENDDO
```

Loop Peeling

- **Source of dependence is a single iteration**

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

Loop peeled to..
A(1) = A(1) + A(1)
DO I = 2, N
    A(I) = A(I) + A(1)
ENDDO

Vectorize to..
A(1) = A(1) + A(1)
A(2:N)= A(2:N) + A(1)
```

Section-based Splitting

```
DO I = 1, N
   DO J = 1, N/2
S1:  B(J,I) = A(J,I) + C
   ENDDO
   DO J = 1, N
S2:  A(J,I+1) = B(J,I) + D
   ENDDO
ENDDO
```

– J Loop bound by recurrence due to B
– Only a portion of B is responsible for it

- Partition second loop into loop that uses result of S1 and loop that does not

```
DO I = 1, N
   DO J = 1, N/2
S1: B(J,I) = A(J,I) + C
   ENDDO
   DO J = 1, N/2
S2: A(J,I+1) = B(J,I) + D
   ENDDO
   DO J = N/2+1, N
S3: A(J,I+1) = B(J,I) + D
   ENDDO
ENDDO
```

Section-based Splitting

```
DO I = 1, N
   DO J = 1, N/2
S1:  B(J,I) = A(J,I) + C
   ENDDO
   DO J = 1, N/2
S2:  A(J,I+1) = B(J,I) + D
   ENDDO
   DO J = N/2+1, N
S3:  A(J,I+1) = B(J,I) + D
   ENDDO
ENDDO
```

- S3 now independent of S1 and S2

- Loop distribute to

```
DO I = 1, N
    DO J = N/2+1, N
S3: A(J,I+1) = B(J,I) + D
    ENDDO
ENDDO
DO I = 1, N
    DO J = 1, N/2
S1: B(J,I) = A(J,I) + C
    ENDDO
    DO J = 1, N/2
S2: A(J,I+1) = B(J,I) + D
    ENDDO
ENDDO
```

Section-based Splitting

```
DO I = 1, N
  DO J = N/2+1, N
S3:  A(J,I+1) = B(J,I) + D
  ENDDO
ENDDO
```

- Vectorized to

```
A(N/2+1:N,2:N+1) = B(N/2+1:N,1:N) + D
```

```
DO I = 1, N
  DO J = 1, N/2
S1:  B(J,I) = A(J,I) + C
  ENDDO
  DO J = 1, N/2
S2:  A(J,I+1) = B(J,I) + D
  ENDDO
ENDDO
```

```
DO I = 1, N
 B(1:N/2,I) = A(1:N/2,I) + C
 A(1:N/2,I+1) = B(1:N/2,I) + D
ENDDO
```

Homework #4 (Written Assignment)

1. **Solve exercise 5.6 in book**
   – Your solution should be legal for all values of K (note that the value of K is invariant in loop I)

- **Due in class on Thursday, Oct 17th**

- Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.