
COMP 515: Advanced Compilation for Vector and Parallel Processors

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu



Acknowledgments

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

Loop Skewing

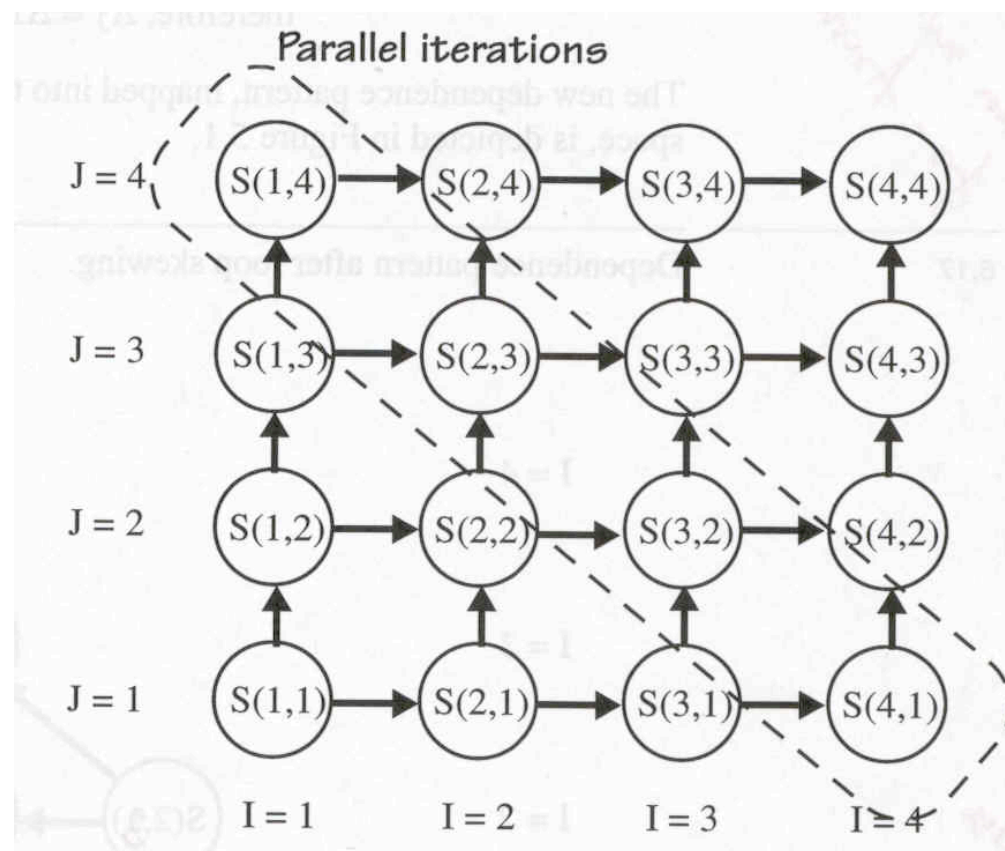
- Reshape Iteration Space to uncover parallelism

```
DO I = 1, N
  DO J = 1, N
    (=,<)
S:  A(I,J) = A(I-1,J) + A(I,J-1)
    (<,&=)
  ENDDO
ENDDO
```

Parallelism not apparent

Loop Skewing

- Dependence Pattern before loop skewing



Loop Skewing

- Do the following transformation called loop skewing

$j=J+I$ or $J=j-I$

DO I = 1, N

DO j = I+1, I+N

(=, <)

S: $A(I, j-I) = A(I-1, j-I) + A(I, j-I-1)$

(<, <)

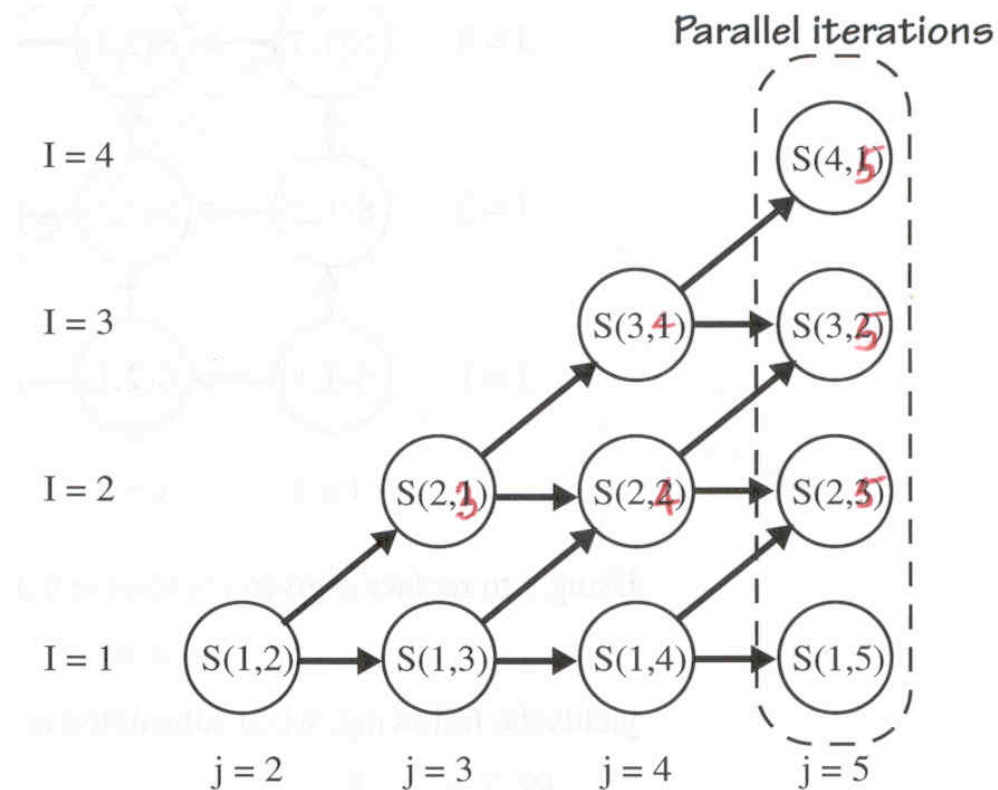
ENDDO

ENDDO

Note: Direction Vector Changes

Loop Skewing

- Dependence pattern after loop skewing



Loop Skewing

```
DO I = 1, N
  DO j = I+1, I+N
S:      A(I,j-I) = A(I-1,j-I) + A(I,j-I-1)
  ENDDO
```

```
ENDDO
```

Loop interchange to..

```
DO j = 2, N+N
  DO I = max(1,j-N), min(N,j-1)
S:      A(I,j-I) = A(I-1,j-I) + A(I,j-I-1)
  ENDDO
```

```
ENDDO
```

Vectorize to..

```
DO j = 2, N+N
  FORALL I = max(1,j-N), min(N,j-1)
S:      A(I,j-I) = A(I-1,j-I) + A(I,j-I-1)
  END FORALL
ENDDO
```

Loop Skewing

- Disadvantages:
 - Varying vector length
 - Not profitable if N is small
 - If vector startup time is more than speedup time, this is not profitable
 - Vector bounds must be recomputed on each iteration of outer loop
 - Apply loop skewing if everything else fails
-

Chapter 5: Putting It All Together

- **Good Part**
 - Many transformations imply more choices to exploit parallelism
 - **Bad Part**
 - *Choosing the right transformation*
 - How to automate transformation selection process?
 - Interference between transformations
-

Putting It All Together

- Example of Interference

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

Sum Reduction gives..

```
DO I = 1, N
  S(I) = S(I) + SUM (A(I,1:M))
ENDDO
```

While Loop Interchange and Vectorization gives..

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

Putting It All Together

- Any algorithm which tries to tie all transformations must
 - Take a global view of transformed code
 - Know the architecture of the target machine
- Goal of our algorithm
 - Finding ONE good vector loop [works well for most vector register architectures]

Unified Framework

- *Detection*: finding ALL loops for EACH statement that can be run in vector
 - *Selection*: choosing *best* loop for vector execution for EACH statement
 - *Transformation*: carrying out the transformations necessary to vectorize the selected loop

 - See Section 5.10 for details
-

Performance on Benchmark

Vectorizing Compiler	Total			Dependence			Vectorization			Idioms			Completeness		
	V	P	N	V	P	N	V	P	N	V	P	N	V	P	N
PFC	70	6	24	17	0	7	25	4	5	5	0	10	23	2	2
Alliant FX/8, Fortran V4.0	68	5	27	19	0	5	20	5	9	10	0	5	19	0	8
Amdahl VP-E, Fortran 77	62	11	27	16	1	7	21	8	5	11	1	3	14	1	12
Ardent Titan-1	62	6	32	18	0	6	19	5	10	9	0	6	16	1	10
CDC Cyber 205, VAST-2	62	5	33	16	0	8	20	5	9	7	0	8	19	0	8
CDC Cyber 990E/995E	25	11	64	8	0	16	6	8	20	3	1	11	8	2	17
Convex C Series, FC 5.0	69	5	26	17	0	7	25	4	5	11	0	4	16	1	10
Cray series, CF77 V3.0	69	3	28	20	0	4	18	3	13	9	0	6	22	0	5
CRAX X-MP , CFT V1.15	50	1	49	16	0	8	12	1	21	10	0	5	12	0	15
Cray Series, CFT77 V3.0	50	1	49	17	0	7	8	1	25	7	0	8	18	0	9
CRAY-2, CFT2 V3.1a	27	1	72	5	0	19	3	1	30	8	0	7	11	0	16
ETA-10, FTN 77 V1.0	62	7	31	18	0	6	18	7	9	7	0	8	19	0	8
Gould NP1, GCF 2.0	60	7	33	14	0	10	19	7	8	8	0	7	19	0	8
Hitachi S-810/820,	67	4	29	14	0	10	24	4	6	14	0	1	15	0	12
IBM 3090/VF, VS Fortran	52	4	44	12	0	12	19	3	12	5	1	9	16	0	11
Intel iPSC/2-VX, VAST-2	56	8	36	15	0	9	17	8	9	6	0	9	18	0	9
NEC SX/2, F77/SX	66	5	29	17	0	7	21	5	8	12	0	3	16	0	11
SCS-40, CFT x13g	24	1	75	7	0	17	6	1	27	5	0	10	6	0	21
Stellar GS 1000, F77	48	11	41	14	0	10	20	9	5	4	1	10	10	1	16
Unisys ISP, UFTN 4.1.2	67	13	20	21	3	0	19	8	7	10	2	3	17	0	10

PFC = Parallel Fortran Converter tool developed at Rice by Allen & Kennedy

Test 171: One example that PFC was unable to vectorize

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

Coarse-Grain Parallelism

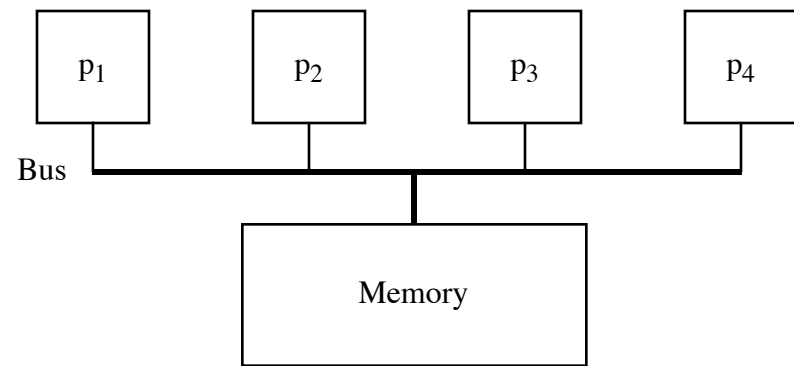
Chapter 6 of *Allen and Kennedy*

Introduction

- Previously, our transformations targeted vector and superscalar architectures.
 - In this lecture, we worry about transformations for symmetric multiprocessor machines.
 - The difference between these transformations tends to be one of granularity.
-

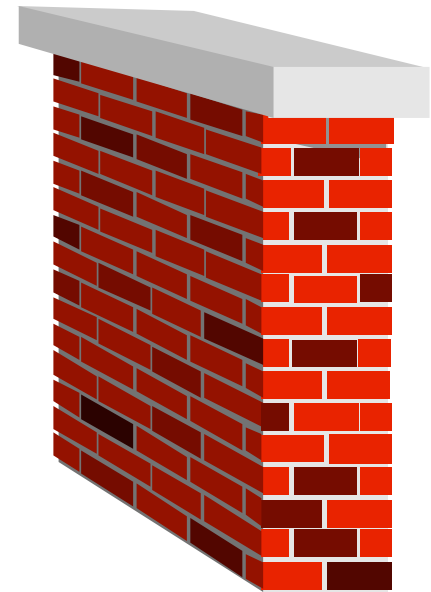
Review

- **SMP machines have multiple processors all accessing a central memory.**
- **The processors are unrelated, and can run separate processes.**
- **Starting processes and synchronization between processes is expensive.**



Synchronization

- A basic synchronization element is the barrier.
- A barrier in a program forces all processes to reach a certain point before execution continues.
- Bus contention can cause slowdowns.



Single Loops

- The analog of scalar expansion is privatization.
- Temporaries can be given separate namespaces for each iteration.

```
      DO I == 1,N  
S1      T = A(I)  
S2      A(I) = B(I)  
S3      B(I) = T  
      ENDDO
```

```
      PARALLEL DO I = 1,N  
          PRIVATE t  
S1      t = A(I)  
S2      A(I) = B(I)  
S3      B(I) = t  
      ENDDO
```

Privatization

Definition: A scalar variable x in a loop L is said to be *privatizable* if every path from the loop entry to a use of x inside the loop passes through a definition of x .

Privatizability can be stated as a data-flow problem:

$$up(x) = use(x) \cup (!def(x) \cap \bigcup_{y \in succ(x)} up(y))$$

$$private(L) = !up(entry) \cap (\bigcup_{y \in L} def(y))$$

We can also do this by declaring a variable x private if its SSA graph doesn't contain a phi function at the entry.

Array Privatization

We need to privatize array variables.

For iteration J , upwards exposed variables are those exposed due to loop body without variables defined earlier.

```
DO I = 1,100
S0   T(1)=X
L1   DO J = 2,N
S1       T(J) = T(J-1)+B(I,J)
S2       A(I,J) = T(J)
           ENDDO
ENDDO
```

$$up(L_1) = \bigcup_{J=2}^N (\{T(J-1)\} \setminus \{T(n) : 2 \leq n \leq j\})$$

So for this fragment, $T(1)$ is the only exposed variable.

Array Privatization

- Using this analysis, we get the following code:

```
      PARALLEL DO I = 1,100
          PRIVATE t
S0      t(1) = X
L1      DO J = 2,N
S1          t(J) = t(J-1)+B(I,J)
S2          A(I,J)=t(J)
          ENDDO
      ENDDO
```

Loop Distribution

- Loop distribution eliminates carried dependencies.
 - Consequently, it often creates opportunity for outer-loop parallelism.
 - However, we must add extra barriers to keep dependent loops from executing out of order, so the overhead may override the parallel savings.
-

Loop Alignment

- Many carried dependencies are due to array alignment issues.
- If we can align all references, then dependencies would go away, and parallelism is possible.
- This is also related to Software Pipelining

```
DO I = 2,N
```

```
  A(I) = B(I)+C(I)
```

```
  D(I) = A(I-1)*2.0
```

```
ENDDO
```



```
DO I = 1,N
```

```
  IF (I .GT. 1) A(I) = B(I)+C(I)
```

```
  IF (I .LT. N) D(I+1) = A(I)*2.0
```

```
ENDDO
```

Alignment

- There are other ways to align the loop:

```
DO I = 2,N
  J = MOD(I+N-4,N-1)+2
  A(J) = B(J)+C
  D(I)=A(I-1)*2.0
ENDDO
```

```
D(2) = A(1)*2.0
DO I = 2,N-1
  A(I) = B(I)+C(I)
  D(I+1) = A(I)*2.0
ENDDO
A(N) = B(N)+C(N)
```

Alignment

- If an array is involved in a recurrence, then alignment isn't possible.
- If two dependencies between the same statements have different dependency distances, then alignment doesn't work.
- We can fix the second case by replicating code:

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



```
DO I = 1,N
  A(I+1) = B(I)+C
  ! Replicated Statement
  IF (I .EQ 1) THEN
    t = A(I)
  ELSE
    t = B(I-1)+C
  END IF
  X(I) = A(I+1)+t
ENDDO
```

Alignment

Theorem: Alignment, replication, and statement reordering are sufficient to eliminate all carried dependencies in a single loop containing no recurrence, and in which the distance of each dependence is a constant independent of the loop index

- We can establish this constructively.
 - Let $G = (V, E, \mu)$ be a weighted graph. $v \in V$ is a statement, and $\mu(v_1, v_2)$ is the dependence distance between v_1 and v_2 . Let $o: V \rightarrow \mathbb{Z}$ give the offset of vertices.
 - G is said to be carry free if $o(v_1) + \mu(v_1, v_2) = o(v_2)$.
 - See pp. 250 - 254 in book for details
-

Homework

- Reading list for next class
 - Chapter 6, *Creating Coarse-Grained Parallelism*
- Homework assignment for discussion in next class
 - Exercise 6.1 a)