
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

Recap

- More transformations to expose more fine-grained parallelism
 - Node Splitting
 - Recognition of Reductions
 - Index-Set Splitting
 - Run-time Symbolic Resolution
 - Loop Skewing
- Unified framework to generate vector code
- Note: these transformations are useful for generating other forms of parallel code as well (beyond vector)

} Previous lecture

} This lecture

Run-time Symbolic Resolution

- “Breaking Conditions”

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

Transformed to..

```
IF(L.LE.0 .OR. L.GT.N) THEN
  A(L+1:N+L) = A(1:N) + B(1:N)
ELSE
  DO I = 1, N
    A(I+L) = A(I) + B(I)
  ENDDO
ENDIF
```

Run-time Symbolic Resolution

- Identifying minimum number of breaking conditions to break a recurrence is NP-hard
 - NOTE: in practice, this can be more important for conditions related to pointer aliasing than for array subscripts
- Heuristic:
 - Identify when a critical dependence can be conditionally eliminated via a breaking condition

Loop Skewing

- Reshape Iteration Space to uncover parallelism

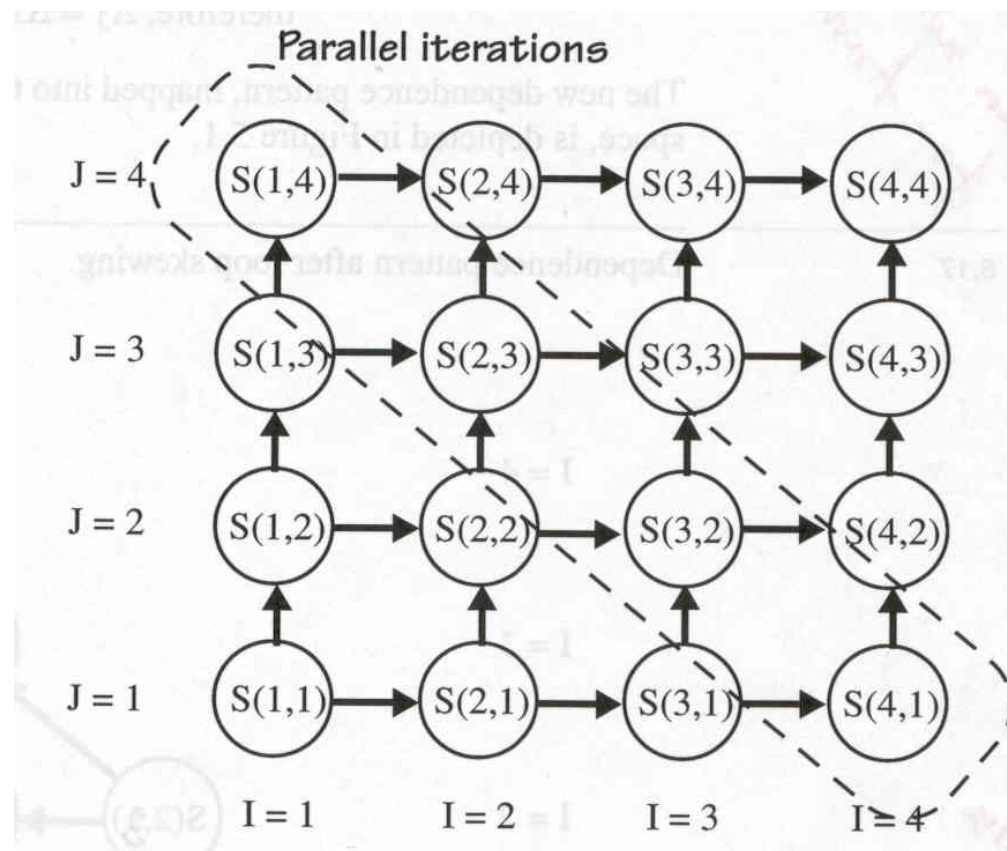
```
DO I = 1, N
  DO J = 1, N
    S: A(I,J) = A(I-1,J) + A(I,J-1)
  ENDDO
ENDDO
```

The diagram illustrates data dependencies between iterations. An arrow labeled "(=, <)" points from iteration (I-1, J) to (I, J). Another arrow labeled "<, =" points from iteration (I, J-1) to (I, J).

Parallelism not apparent

Loop Skewing

- Dependence Pattern before loop skewing

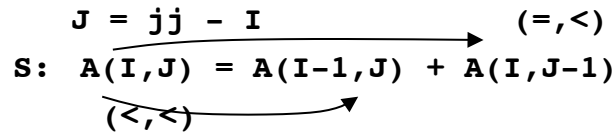


Loop Skewing

- Do the following transformation called loop skewing

$jj=J+I$ or $J=jj-I$

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



Note: Direction Vector Changes, but statement body remains the same
(Examples in textbook usually copy propagate $J=jj-I$ in all uses of J)

Loop Skewing

- Dependence pattern after loop skewing
 - Replace j by jj in figure below



Loop Skewing

```
DO I = 1, N ! DV = { (<,<), (=, <) }  
  DO jj = I+1, I+N  
S:  A(I,jj-I) = A(I-1,jj-I) + A(I,jj-I-1)  
  ENDDO
```

ENDDO

Loop interchange to..

```
DO jj = 2, N+N ! DV = { (<,<), (<, =) }  
  DO I = max(1,jj-N), min(N,jj-1)  
S:  A(I,jj-I) = A(I-1,jj-I) + A(I,jj-I-1)  
  ENDDO
```

ENDDO

Vectorize to..

```
DO jj = 2, N+N  
  FORALL I = max(1,jj-N), min(N,jj-1)  
S:  A(I,jj-I) = A(I-1,jj-I) + A(I,jj-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
- We will later study Unimodular and Polyhedral transformations, which include generalizations of loop skewing

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 in each loop nest [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 Benchmarks

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

17 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

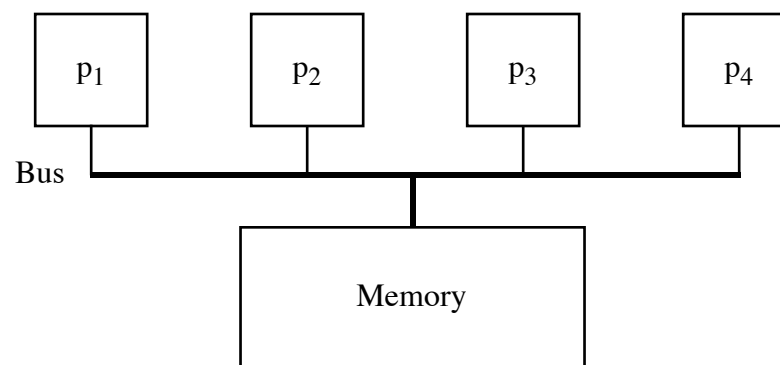
Chapter 6 of Allen and Kennedy

Introduction

- Previously, our transformations targeted vector and superscalar architectures.
- In Chapter 6, we worry about transformations for symmetric multiprocessor machines.
- The difference between these transformations tends to be one of granularity.

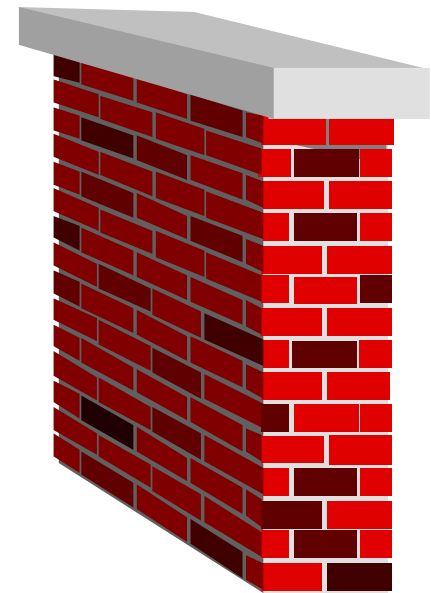
Review

- **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 at the end of a parallel loop.
- A barrier in a program forces all processes to reach a certain point before execution continues.
- Bus contention can cause slowdowns.



Techniques for parallelizing a single loop

- Single loop methods
 - Privatization
 - Loop distribution
 - Loop fusion
 - Alignment
 - Code replication

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.

Example of Privatizable Scalar Variable

```
{ int i, v, a[n], b[n], c[n], d[n];
  double e[n], f[n];
  .../ * initialization for a,b,c,d,e,f */

  for (i = 0; i < n; i++) {
    if (i & 1)
      v = a[i];
    else
      v = b[i];
    610 { while (c[v] != d[v])
        620 { v = c[v];
            e[i] = f[i] + v;
          }
    }
    printf ("%d \n", v);
  }
}
```

- “Method of, system for, and computer program product for efficient identification of private variables in program loops by an optimizing compiler”, US Patent 5,790,859, issued Aug 1998.

REMINDER:

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.