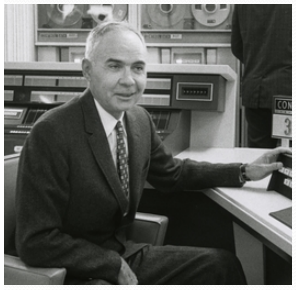
Allen-Kennedy Vectorization Algorithm

Control Data Corporation

Formed by several defecting Minneapolis-based Sperry Univac managers and engineers in 1957

- -Co-founder of ERA William Norris, co-founded CDC and was unanimous choice as CEO and ran the firm for 3 decades
- -Sperry Univac's Seymour Cray became CDC's chief design engineer



William Norris

Seymour Cray

Late 1960s and 1970s CDC #2 (to IBM) in the Computer Industry (CDC is 1st in Supercomputing)

CDC Revenue

1962 $41M
1967 $245M
1969 $1.02B
1972 $1.2B
1977 $2.3B
1982 $3.2B

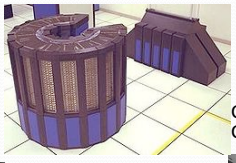
March 6, 1963

Control Data Corporation listed on the NYSE

Cray Research

Seymour Cray leaves CDC in 1972 to form Cray Research, Inc. (HQ Minneapolis)
Cray Research displaces CDC as supercomputer leader

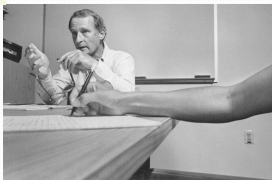
Cray Research takes over the lead in supercomputing with Cray I (1976), and furthers its lead with Cray X-MP (1983) and Cray II (1985) supercomputers.



Cray II

Cray Research founder and CEO Seymour Cray

Cray I



Arguably the Highest Compiler Success

- David Cook's group pioneered dependence-based program parallelization
- Ken Kennedy's group developed the algorithm we know today
  - Allen/Kennedy paper, one of the most cited in literature
  - The Rice compiler
    - renowned for developing techniques that work
      - vectorization, interprocedural analysis, parallelization, locality improvements
  - The algorithm ushered in the supercomputing era, according to Steve Wallach
- Recent uses
  - deadlock-free locking in concurrent code
  - a Rochester paper by Mehdi Manshadi

Vectorization

- Theorem 2.8. It is valid to convert a sequential loop to a parallel loop if the loop carries no dependence.
- Want to convert loops like:

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

- to `X(1:N) = X(1:N) + C` (Fortran 77 to Fortran 90)
- What about converting:

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

to `X(2:N+1) = X(1:N) + C` ?

Loop Distribution

- Can statements in loops which carry dependences be vectorized?

```
      D0 I = 1, N
S1      A(I+1) = B(I) + C
S2      D(I) = A(I) + E
      ENDDO
```

- Can it be converted to?

```
S1    A(2:N+1) = B(1:N) + C
S2    D(1:N) = A(1:N) + E
```


Loop Distribution

```
      DO I = 1, N
S1      A(I+1) = B(I) + C
S2      D(I) = A(I) + E
      ENDDO
```

- **transformed to:**

```
      DO I = 1, N
S1    A(I+1) = B(I) + C
      ENDDO
      DO I = 1, N
S2      D(I) = A(I) + E
      ENDDO
```

- **leads to:**

```
S1    A(2:N+1) = B(1:N) + C
S2    D(1:N) = A(1:N) + E
```


Loop Distribution

- Does loop distribution always work?

```
      DO I = 1, N
S1        A(I-1) = B(I) + C
S2        B(I-1) = A(I) + E
      ENDDO
```

$S_1\ \delta_1\ S_2$ and $S_2\ \delta_1\ S_1$

- What about:

```
      DO I = 1, N
S1          B(I) = A(I) + E
S2          A(I+1) = B(I) + C
      ENDDO
```


Advanced Vectorization Algorithm

procedure codegen (L, k, D)

// L is the maximal loop nest containing the statement.
// k is the current loop level
// D is the dependence graph for statements in L.
find the set $\{S_1, S_2, \ldots, S_m\}$ of SCCs in D

construct L_p from L by reducing each S_i to a single node

use topological sort to order nodes in L_p to $\{p_1, p_2, \ldots, p_m\}$

for i = 1 to m do begin
  if p_i is a dependence cycle then
    generate a level-k DO
    construct D_i be p_i dependence edges in D at level k+1 or greater
    codegen (p_i, k+1, D_i)
    generate the level-k ENDDO
  else
    vectorize p_i with respect to every loop containing it
  end
end vectorize

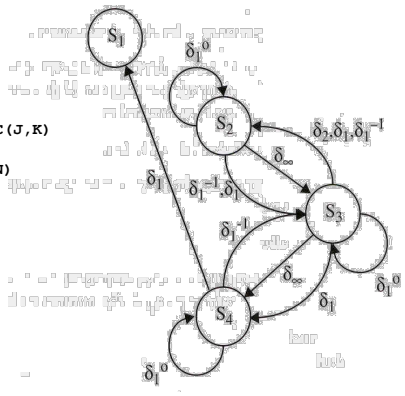
A Simple Example

```
      DO I = 1, N
              DO J = 1, M
S1                A(I+1,J) = A(I,J) + B
              ENDDO
      ENDDO
```

- Dependence graph?
- Dependence level?
- Vectorization process?

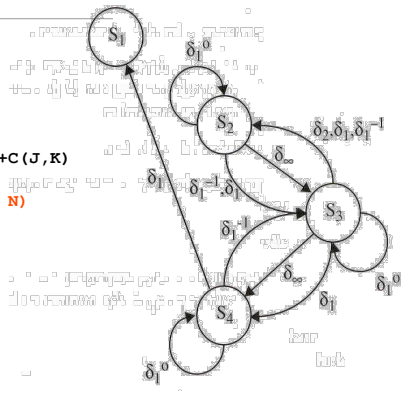
Advanced Vectorization Algorithm

```
DO I = 1, 100
S1   X(I) = Y(I) + 10
      DO J = 1, 100
S2      B(J) = A(J,N)
        DO K = 1, 100
S3        A(J+1,K)=B(J)+C(J,K)
        ENDDO
S4      Y(I+J) = A(J+1, N)
      ENDDO
ENDDO
```

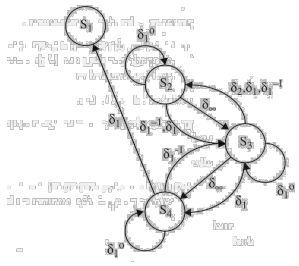


Advanced Vectorization Algorithm

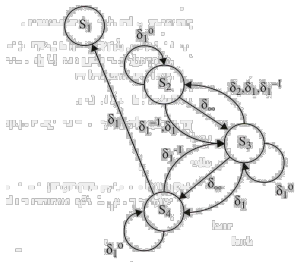
```
DO I = 1, 100
S1   X(I) = Y(I) + 10
       DO J = 1, 100
S2       B(J) = A(J,N)
           DO K = 1, 100
S3           A(J+1,K)=B(J)+C(J,K)
           ENDDO
S4       Y(I+J) = A(J+1, N)
       ENDDO
ENDDO
```




```
DO I = 1, 100
S1   X(I) = Y(I) + 10
       DO J = 1, 100
S2       B(J) = A(J,N)
           DO K = 1, 100
S3           A(J+1,K)=B(J)+C(J,K)
           ENDDO
S4       Y(I+J) = A(J+1, N)
       ENDDO
ENDDO
```



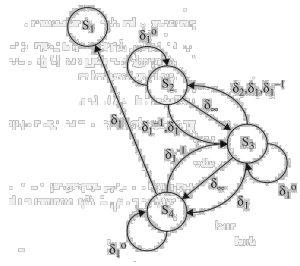

```
DO I = 1, 100
S1   X(I) = Y(I) + 10
       DO J = 1, 100
S2       B(J) = A(J,N)
           DO K = 1, 100
S3           A(J+1,K)=B(J)+C(J,K)
           ENDDO
S4       Y(I+J) = A(J+1, N)
       ENDDO
ENDDO
```




```
DO I = 1, 100
S1   X(I) = Y(I) + 10
       DO J = 1, 100
S2       B(J) = A(J,N)
           DO K = 1, 100
S3           A(J+1,K)=B(J)+C(J,K)
           ENDDO
S4       Y(I+J) = A(J+1, N)
       ENDDO
ENDDO
```

procedure *vectorize* (*L*, *k*, *D*)
// *L* is the maximal loop nest containing the statement.
// *k* is the current loop level
// *D* is the dependence graph for statements in *L*.
find the set $\{S_1, S_2, \ldots, S_m\}$ of SCCs in D
construct L_k from *L* by reducing each S_i to a single node
use topological sort to order nodes in L_k to $\{p_1, p_2, \ldots, p_m\}$

for *i* = 1 **to** *m* **do begin**
  if p_i is a dependence cycle **then**
    generate a level-*k* DO
    construct D_i be p_i dependence edges in *D* at level *k*+1 or greater
    codegen (p_i, *k*+1, D_i)
    generate the level-*k* ENDDO
  else
    vectorize p_i with respect to every loop containing it
  end
end *vectorize*




```
DO I = 1, 100
  DO J = 1, 100
    B(J) = A(J,N)
    A(J+1,1:100)=B(J)+C(J,1:100)
  ENDDO
  Y(I+1:I+100) = A(2:101,N)
ENDDO

X(1:100) = Y(1:100) + 10
```

Lock Allocation

- Composition of concurrent code
  - fine-grained locking
    - deadlock
  - eg. thread 1 acquires A and then B and thread 2 acquires B and then A
  - solved by requiring same order
    - what if they cannot
- Lock dependence
  - if B depends on A, then A must be acquired first
  - take the dependence graph, compute SCC
  - locks involved in SCC are coarsened into a single lock
  - the order is the topological sort

Review Questions

- What is vectorization?
  - Does vectorization implies loop distribution?
- What are the steps of the Allen-Kennedy algorithm?
  - Why is topological sort needed? What happens if the dependence graph is cyclic?
  - After a level-k loop is serialized, how the algorithm update the dependence graph and why?
  - Does the algorithm always vectorize a loop nest one level at a time? (hint: not always)
- How to allocate locks in concurrent code to prevent deadlock?