
COMP 515: Advanced Compilation for Vector and Parallel Processors

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



Announcements

- I've blocked off 2:30pm - 4:30pm on April 16th (Thurs) for one-on-one discussions of Homework 2
 - Please contact Bob Garcia to schedule a time

Interprocedural Analysis and Optimization

Chapter 11

Introduction

- **Interprocedural Analysis**
 - Gathering information about the whole program instead of a single procedure
- **Interprocedural Optimization**
 - Program transformation modifying more than one procedure using interprocedural analysis



Overview: Interprocedural Analysis

- Examples of Interprocedural problems
- Classification of Interprocedural problems
- Solve two Interprocedural problems
 - Side-effect Analysis
 - Alias Analysis



Some Interprocedural Problems

- **Modification and Reference Side-effect**

```
COMMON X,Y
```

```
...
```

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

```
S0: CALL P
```

```
S1: X(I) = X(I) + Y(I)
```

```
ENDDO
```

- **Can vectorize if P**
 1. neither modifies nor uses X
 2. does not modify Y

Modification and Reference Side Effect

- $MOD(s)$: set of variables that may be modified as a side effect of call at s
- $REF(s)$: set of variables that may be referenced as a side effect of call at s

DO I = 1, N

S0: CALL P

S1: X(I) = X(I) + Y(I)

ENDDO

- Can vectorize S1 if

$$x \notin REF(S0) \wedge x \notin MOD(S0) \wedge y \notin REF(S0)$$



Alias Analysis

```
SUBROUTINE S(A,X,N)
  COMMON Y
  DO I = 1, N
S0:      X = X + Y*A(I)
  ENDDO
END
```

- Could have kept X and Y in different registers and stored in X outside the loop
 - What happens when there is a call, CALL S(A,Y,N)?
 - Then Y is aliased to X on entry to S
 - Can't delay update to X in the loop any more
 - ALIAS(p,x): set of variables that may refer to the same location as formal parameter x on entry to p
-

Call Graph Construction

- Call Graph $G=(N,E)$
 - N: one vertex for each procedure
 - E: one edge for each possible call
 - Edge (p,q) is in E if procedure p calls procedure q
 - Looks easy
 - Construction difficult in presence of procedure parameters
 - Also for virtual method calls in object-oriented languages
-

Call Graph Construction

```
SUBROUTINE S(X,P)
S0:    CALL P(X)
      RETURN
END
```

- P is a procedure parameter to S
 - What values can P have on entry to S ?
 - $CALL(s)$: set of all procedures that may be invoked at s
 - Resembles the alias analysis problem
-

Live and Use Analysis

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

```
PARALLEL DO I = 1, N
  LOCAL †
  † = X(I)*C
  A(I) = † + B(I)
  C(I) = † + D(I)
  IF(I.EQ.N) T = †
ENDDO
```

- This loop can be parallelized by making T a local variable in the loop
 - Copy of local version of T to the global version of T is required to ensure correctness
 - What if T was not live outside the loop?
-

Live and Use Analysis

- Solve Live analysis using Use Analysis
 - $USE(s)$: set of variables having an upward exposed use in procedure p called at s
 - If a call site, s is in a single basic block(b), x is live if either
 - x in $USE(s)$ or
 - P doesn't assign a new value to x and x is live in some control flow successor of b
-

Kill Analysis

```
DO I = 1, N
SO:      CALL INIT(T,I)
         T = T + B(I)
         A(I) = A(I) + T
ENDDO
```

- To parallelize the loop:
 - INIT must not create a recurrence with respect to the loop
 - T must not be upward exposed (otherwise it cannot be privatized)
-

Kill Analysis

```
DO I = 1, N
SO:    CALL INIT(T,I)
      T = T + B(I)
      A(I) = A(I) + T
ENDDO
```

```
SUBROUTINE INIT(T,I)
  REAL T
  INTEGER I
  COMMON X(100)
  T = X(I)
END
```

- T has to be assigned before being used on every path through INIT
 - If INIT is of this form we can see that T can be privatized
-

Kill Analysis

- **KILL(s)**: set of variables assigned on every path through procedure p called at s and through procedures invoked in p
- T in the previous example can be privatized under the following condition
- Also we can express LIVE(s) as following

$$T \in (KILL(S0) \cap \neg USE(S0))$$

$$LIVE(s) = USE(s) \cup (\neg KILL(s) \cap \bigcup_{b \in succ(s)} LIVE(b))$$