COMP 515: Advanced Compilation for Vector and Parallel Processors

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

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

COMP 515 Lecture 23 22 November, 2011

Acknowledgments

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

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 Analysis Problems

- **Modification and Reference Side-effect**

```
            COMMON X(N),Y(N)  ! Static arrays
            ...
            DO I = 1, N
S0:             CALL P
S1:             X(I) = X(I) + Y(I)
            ENDDO
```

- Can parallelize I loop 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** $x \notin REF(S0) \wedge x \notin MOD(S0) \wedge y \notin REF(S0)$
  - TODO: replace REF by MOD for y term above

Alias Analysis

```
 COMMON Y ! static variable
SUBROUTINE S(A,X,N)
          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 since we don't know for sure if X and Y are aliased
- ALIAS(p,x): set of variables that may refer to the same location as formal parameter x on entry to procedure 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 may call procedure q
- Looks easy
- Construction difficult in presence of procedure parameters
- Also for virtual method calls in object-oriented languages

Call Graph Construction: Example with procedure parameter

```
      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
```

- This loop can be parallelized by making T a local variable in the loop

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

- 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
S0:      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
S0:         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

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

- Also we can express LIVE(s) as following

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

Constant Propagation

```
        SUBROUTINE S(A,B,N,IS,I1)
            REAL A(*), B(*)
            DO I = 0, N-1
S0:             A(IS*I+I1) = A(IS*I+I1) + B(I+1)
            ENDDO
        END
```

- If IS=0 the loop around S0 is a reduction
- If IS!=0 the loop can be vectorized
- **CONST(p):** set of variables with known constant values on every invocation of p
- Knowledge of CONST(p) useful for interprocedural constant propagation

Interprocedural Problem Classification

- May and Must problems
  - MOD, REF and USE are 'May' problems
  - KILL is a 'Must' problem
- Flow sensitive and flow insensitive problems
  - Flow sensitive: control flow info included in analysis
  - Flow insensitive: control flow info is (conservatively) ignored
- May and Must classification can apply to call graph edges as well

Flow Insensitive Side-effect Analysis

- **Assumptions**
  - No procedure nesting i.e., no inner functions
  - All parameters passed by reference
  - Size of the parameter list bounded by a constant,
- **We will formulate and solve MOD(s) problem**

Solving MOD

$$MOD(s) = DMOD(s) \cup \bigcup_{x \in DMOD(s)} ALIAS(p,x)$$

- **DMOD(s):** set of variables which are directly modified as side-effect of call at s (ignoring aliases)

$$DMOD(s) = \{v \mid s \Rightarrow p, v \xrightarrow{s} w, w \in GMOD(p)\}$$

- **GMOD(p):** set of global variables and formal parameters w of p that are modified, either directly or indirectly as a result of invocation of p
  - Global variables are modeled as special "parameters" in this formulation

Example: DMOD and GMOD

```
S0:     CALL P(A,B,C)

            …

            SUBROUTINE P(X,Y,Z)

                INTEGER X,Y,Z

                X = X*Z

                Y = Y*Z

            END
```

- GMOD(P)={X,Y}
- DMOD(S0)={A,B}

Solving GMOD

- GMOD(p) contains two types of variables
  - Variables explicitly modified in body of P: This constitutes the set IMOD(p)
  - Variables modified as a side-effect of some procedure invoked in p
    - Global variables are viewed as parameters to a called procedure

$$GMOD(p) = IMOD(p) \cup \bigcup_{s=(p,q)} \{z \mid z \xrightarrow{s} w, w \in GMOD(q)\}$$

  - The above formulation is impractical for recursive programs

Solving GMOD

- The previous iterative method may take a long time to converge
  - Problem with recursive calls

```
        SUBROUTINE P(F0,F1,F2,…,Fn)
            INTEGER X,F0,F1,F2,..,Fn
            …
    S0:     F0 = <some expr>
            …
    S1:     CALL P(F1,F2,…,Fn,X)
            …
        END
```

Solving GMOD

- Decompose GMOD(p) differently to get an efficient solution in the presence of recursion
- Key: Treat side-effects to global variables and reference formal parameters separately
- LOCAL refers to local variables in q

$$GMOD(p) = \boxed{IMOD^{+}(p)} \cup \boxed{\bigcup_{s=(p,q)} GMOD(q) \cap \neg LOCAL}$$

Solving for IMOD+

- $x \in IMOD^{+}(p)$ **if**
  - $x \in IMOD(p)$ **or**
  - $x \xrightarrow{s} z, z \in GMOD(q), s = (p,q)$ **and x is a formal parameter of p**
- **Formally defined**

$$IMOD^{+}(p) = IMOD(p) \cup \bigcup_{s=(p,q)} \{z \mid z \xrightarrow{s} w, w \in RMOD(q)\}$$

- **RMOD(p): set of formal parameters in p that may be modified in p, either directly or by assignment to a reference formal parameter of q as a side effect of a call of q in p**

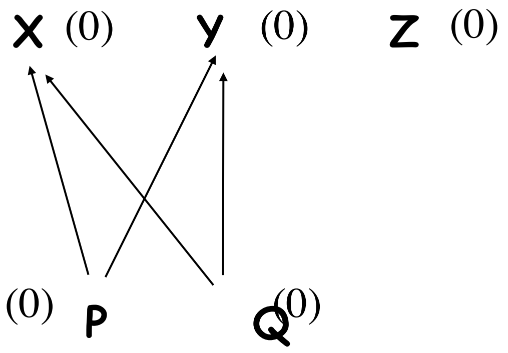
Solving for RMOD

- **RMOD(p):** set of formal parameters in p that may be modified in p, either directly or by assignment to a reference formal parameter of q as a side effect of a call of q in p
- Binding Graph $G_B=(N_B,E_B)$
  - One vertex for each formal parameter of each procedure
  - Directed edge from formal parameter, f1 of p to formal parameter, f2 of q if there exists a call site s=(p,q) in p such that f1 is bound to f2
- Use a marking algorithm to compute RMOD(p) (Figure 11.2)
  - Mark each vertex as false initially
  - Mark formals of P in IMOD(p) as true
  - Perform a closure operation (propagate bits)
    - Mark f1 as true if G_B has an edge from f1 to f2 and f2 is marked true
    - Use worklist algorithm (or reverse DFS, if you prefer)

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
   INTEGER X,Y,Z
   X = Y + Z
   Y = Z + 1
END

SUBROUTINE B(P,Q)
   INTEGER P,Q,I
   I = 2
   CALL A(P,Q,I)
   CALL A(Q,P,I)
END
```

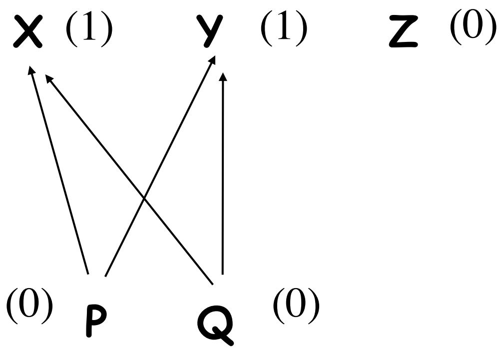


- **IMOD(A)={X,Y}**
- **IMOD(B)={I}**

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
   INTEGER X,Y,Z
   X = Y + Z
   Y = Z + 1
END

SUBROUTINE B(P,Q)
   INTEGER P,Q,I
   I = 2
   CALL A(P,Q,I)
   CALL A(Q,P,I)
END
```

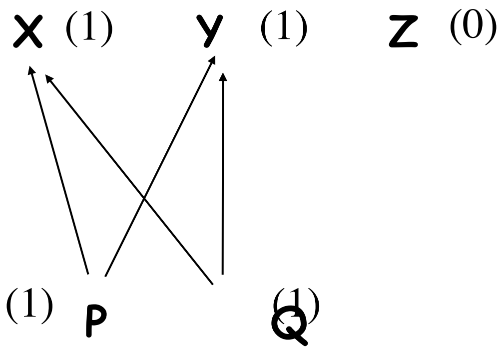


- **IMOD(A)={X,Y}**
- **IMOD(B)={I}**
- **Worklist={X,Y}**

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
   INTEGER X,Y,Z
   X = Y + Z
   Y = Z + 1
END

SUBROUTINE B(P,Q)
   INTEGER P,Q,I
   I = 2
   CALL A(P,Q,I)
   CALL A(Q,P,I)
END
```



- RMOD(A)={X,Y}
- RMOD(B)={P,Q}
- Complexity: $O(N_B + E_B)$

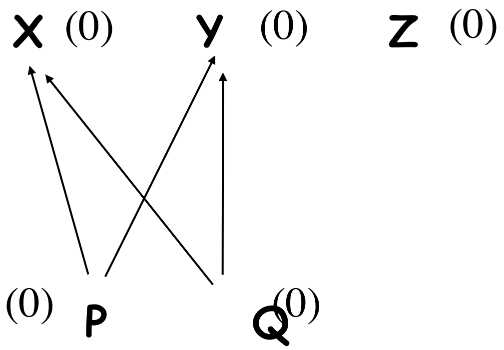
$N_B \le \mu N$ $E_B \le \mu E$

$O(N + E)$

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
    INTEGER X,Y,Z
    X = Y + Z
    Y = Z + 1
END

SUBROUTINE B(P,Q)
    INTEGER P,Q,I
    I = 2
    CALL A(P,Q,I)
    CALL A(Q,P,I)
END
```

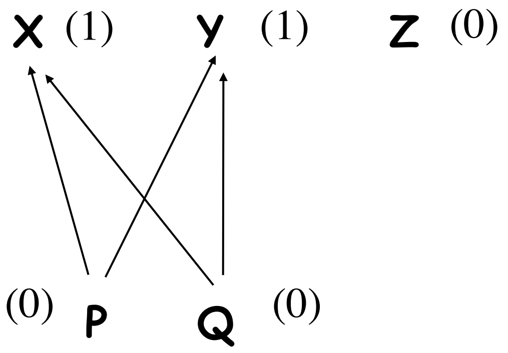


- **IMOD(A)={X,Y}**
- **IMOD(B)={I}**

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
    INTEGER X,Y,Z
    X = Y + Z
    Y = Z + 1
END

SUBROUTINE B(P,Q)
    INTEGER P,Q,I
    I = 2
    CALL A(P,Q,I)
    CALL A(Q,P,I)
END
```

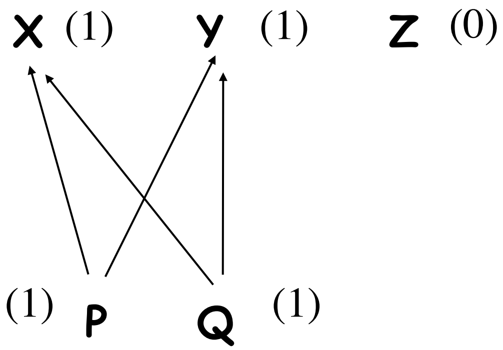


- **IMOD(A)={X,Y}**
- **IMOD(B)={I}**
- **Worklist={X,Y}**

Solving for RMOD

```
SUBROUTINE A(X,Y,Z)
    INTEGER X,Y,Z
    X = Y + Z
    Y = Z + 1
END

SUBROUTINE B(P,Q)
    INTEGER P,Q,I
    I = 2
    CALL A(P,Q,I)
    CALL A(Q,P,I)
END
```



- **RMOD(A)={X,Y}**
- **RMOD(B)={P,Q}**
- **Complexity:** $O(N_B + E_B)$

$N_B \le \mu N$ $\quad E_B \le \mu E$

$O(N + E)$

Solving for IMOD$^+$

- **After gathering RMOD(p) for all procedures, update RMOD(p) to IMOD$^+$(p) using this equation**

$$IMOD^+(p) = IMOD(p) \cup \bigcup_{s=(p,q)} \{z \mid z \xrightarrow{s} w, w \in RMOD(q)\}$$

- **This can be done in O(NV+E) time**

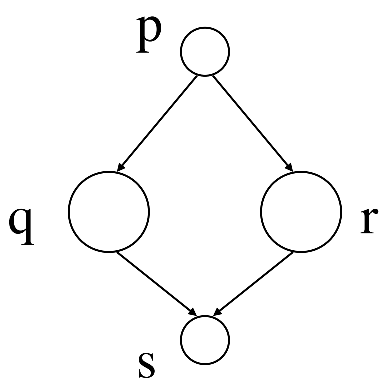
Solving for GMOD

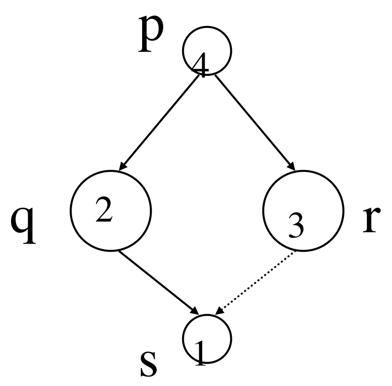
- After gathering IMOD+(p) for all procedures, calculate GMOD (p) according to the following equation

$$GMOD(p) = IMOD^{+}(p) \cup \bigcup_{s=(p,q)} GMOD(q) \cap \neg LOCAL$$

- This can be solved using a DFS algorithm based on Tarjan's SCR algorithm on the Call Graph

Solving for GMOD

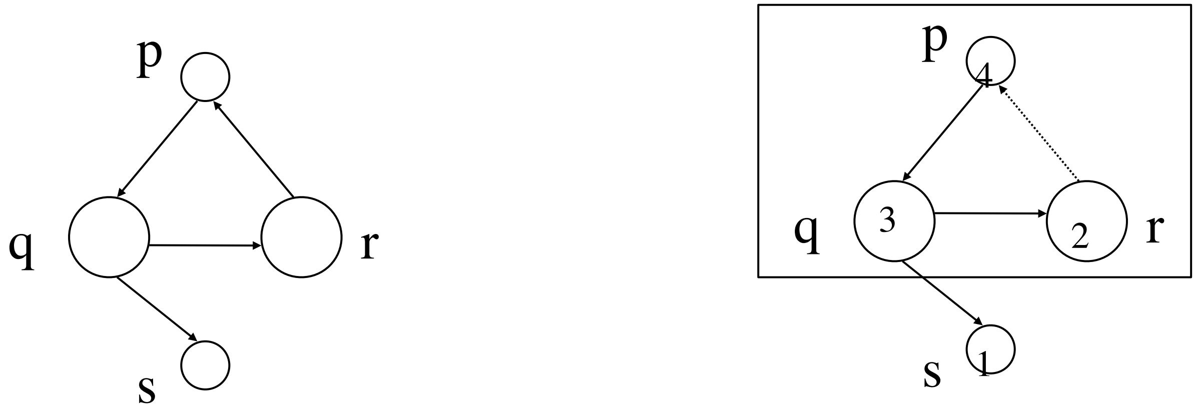




Initialize GMOD(p) to IMOD$^+$(p) on discovery

Update GMOD(p) computation while backing up

Solving for GMOD



Initialize GMOD(p) to IMOD$^+$(p) on discovery

Update GMOD(p) computation while backing up

For each node u in a SCR update GMOD(u) in a cycle

O((N+E)V) Algorithm

Overview: Interprocedural Analysis

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