State-of-the-Art

Automatic Parallelization Of Sequential Programs (using Compile Time Analysis):

- Static data structures (static arrays or stack directed pointers).
  
  - for(i=0; i<n; i++) {
    
  - in general...
  - Challenges!

  - Dynamic memory (Heap) structures

  - Heap structure
  - Unknown at compile time
  - Potentially Unbounded
  - Not limited by the number of tree nodes

  - Max Depth of Dependence = 2

Dynamic Dependence Analysis:

- Dependencies between program statements.
- Dependencies between shared memory locations.
- Dependencies between pointer fields.
- Dependencies between heap structures.

Observation:

- Computation of d depends on that of d and r.
- Computation of d and r can be done in parallel.
- With n processing units (n number of tree nodes), time required = O(log n)

Application

void treeAdd(tree *t) {
  
  if(NULL == t) {
    
  else {
    
  }
}

Motivation

Given a heap traversal code and a heap data structure

p-list:
while(p→next != NULL){
  
S1:  temp = p→num;
S2:  p→next = p→next;
S3:  p→num = temp;
S4:  p→prev = p→prev;
S5:  p→num = temp;
S6:  p = p→next;
}

Conclusion And Future Work

- Dependence detection technique depends on feasibility of depth of dependence computation.
- Plug and Play shape analysis framework.
- Fast dependence analysis; No shape graph manipulation.
- 

Current Activity:
Implementing a prototype of our analysis for a subset of JAVA using SOOT framework.

Future Plans:
To support complete JAVA and show effectiveness on large benchmarks.

References

Max Depth Of Dependence

- Cannot Determine
- Max Depth Of Dependence
- Conservative Decision (Not Parallelizable)

How To??

- Two paths are aliases if they access same node.
- Dependence analysis detects conflicting node accesses.

Flow - Anti - Output - Dependence

Not Parallelizable

Parallelizable

Traversal Code

Dependence Analysis

Shape Analysis

Path Alias Information

Dependence??

Associated Pointer Tag next prev num Index Tag next prev num Index
p-list; p 0 0 0 0 R & W R & W
while() { R & W R & W
p→num = num;

while(p→next != NULL){
  
S1:  q = p;
S2:  p→num = p→num;
S3:  p→next = p→next;
S4:  p→num = p→num;
S5:  p→next = p→next;
S6:  p = p→next;
}

Paths (p)<0,0,1,0,0> conflicts with (p_prv)<1,1,1,1,1>
& (p_nxt)<1, 0, 1, 1> conflicts with (p)<1, 0, 1, 1>

as inferred by the path alias information

Application

Dependence Detection

Path Abstraction:

n + 2 Field Array, n = number of data store fields

Max Depth Of Dependence

Conservative Decision (Not Parallelizable)

References