Compiler-Enabled Cache Management for Pointer-Intensive Codes

Yao Guo, Saurabh Chheda, Csaba Andras Moritz

Department of Electrical & Computer Engineering
University of Massachusetts, Amherst

Motivation

- Compiler-managed caches have been shown to be successful in energy optimizations
  - Real programs often contain pointer intensive parts
  - Precise pointer related analysis is difficult or often not possible for large/complex programs
- Question: can we do better with pointers than just ignoring them?
- Our proposal
  - Speculative Pointer and Distance Analysis
  - Architectural support

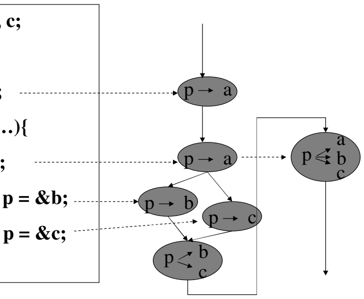
Background on Pointer Analysis

- **Pointer Analysis** determines the set of possible values (called location sets) for each pointer variable in a program
- Different types of analysis possible
  - Type based (e.g. Steensgaard '96)
    - global results (less precise)
  - Dataflow based (e.g. Emami '94, Wilson '95)
    - a result for each program point
    - using points-to graphs
    - most precise
- Our Approach is based on Points-to Graphs

Points-to Graphs

- Dataflow based pointer analysis

```
int a, b, c;
int *p;
p = &a;
for (……){
  *p = 5;
  if (…) p = &b;
    else p = &c;
}
```



Efficient Management of Pointer Accesses

- What information do we need to know about pointers in compiler managed caches?
- How do we perform pointer analysis to gather this information?
- Challenges
  - Complex/large programs, precision, static libraries, undefined pointers, recursion

Our Approach

- Speculative Pointer and Distance Analysis
  - Speculate when you don't have provable information
  - Speculate to reduce complexity or to complete analysis
  - Speculate on pointer access patterns to evaluate strides (locality)
  - Speculate to improve precision of analysis
    - precision is the width of location set
- Context is compiler managed caches
  - Architectural support to validate
  - Compile-time tradeoffs: degree of speculation

Pointer Information Extracted

- *Speculative points-to information*
  - Context and flow sensitive location sets for pointers
- *Frequencies* of locations within location sets
  - We can remove less frequent locations
- Determine *if critical* (e.g., loop-based) or not
  - Focus on more critical pointer variables
- Determine *locality*
  - Pointers change during each loop iteration
  - Stride can be determined with distance analysis
  - Various degrees of confidence can be established
- Determine when a pointer access *is irregular*
  - Negative result is actually useful

7

Speculate to Complete Analysis

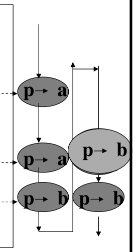
- Applications have static libraries and sources may not be always available
  - Ignore or speculate on their impact on location sets
- Undefined pointers
  - ignore or treat as no information available (e.g., map to dynamic cache management)
- Recursive procedure calls
  - Speculatively stops after a fixed # of iterations

8

Speculative Optimization for Precision

- Optimize pointer location set widths in loop bodies speculatively
- Remove unlikely or infrequent location sets
  - Improves ability to manage access statically in our context
  - Improves precision for the typical case

```
int a,b;
int *p;
p = &a;
for (i=0…1000){
  *p = 5;
  p = &b;
}
```



9

Speculative Distance Analysis

- Objective
  - Capture strides in pointer accesses within loops
  - Determine if there is locality (e.g., if strides are small)
    - Fixed strides can be easily determined
    - Program variable strides can be determined speculatively

10

Speculative Distance Analysis Examples

- Fixed Stride

```
……
int *p;
int arr[100];

p = &arr;
for(… i …){
  *p = i;
  p += 4;
}
```

Stride: 4

- Variable Stride

```
……
int *p;
int arr[10];
p = …;
for(… i …){
  *p = i;
  p += foo(i);
}
```

p->arr

Stride: Small

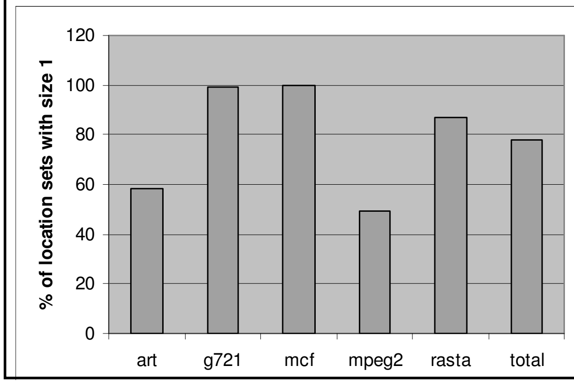
11

Applicability in Cache Management

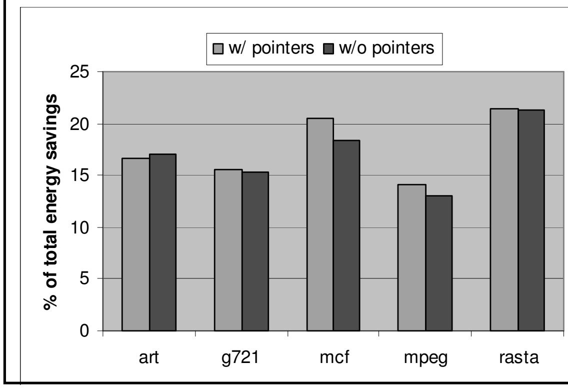
- Cool-Mem (our previous work, ASPLOS 2002)
- Combines static and dynamic memory management
  - fully static (most energy efficient)
  - statically speculative (almost as energy efficient)
  - conventional (least energy efficient)
- Improves energy efficiency by reducing redundancy
- More statically managed accesses reduce energy consumption

12

Results – Location Sets Distribution



Results – Energy Savings Comparison



Conclusions

- **Traditional pointer analysis**
  - Based on compile time provable information
  - Stops analysis if that cannot be guaranteed
  - Used in many performance optimization techniques
- **Speculative pointer and distance analysis**
  - Always completes analysis without restrictions
  - Extracts precise information for our purposes
  - Targets compiler-enabled memory systems
- **Preliminary results**
  - Saves energy for pointer-intensive programs

15