Macroscopic Data Structure Analysis & Optimization

Chris Lattner, Prof. Vikram Adve, UIUC

Macroscopic Data Structure Optimization

Q. Can compilers optimize entire data structures?

Primary Goals:

- **Identify distinct data structure instances**
- **Find important properties of those instances**
- **Optimize each data structure instance based on its usage**
- **Give some control over dynamic layout to the compiler**
- **Develop algorithms suitable for a commercial compiler**

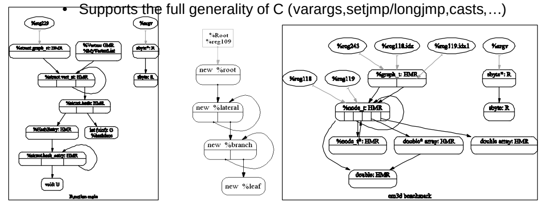
Applications:

- **Application performance (the focus of this poster)**
- **Safety (see SAFECode poster)**
- **Program understanding**
- **Static garbage collection**

Data Structure Analysis (DSA)

Identify Recursive Data Structures & their Properties

- Aggressive Context-Sensitive Analysis
- Captures points-to, mod/ref, type information
- Extremely fast: analyzes 200K LOC programs in < 2s
- Can support standard alias analysis clients & macroscopic clients
- Supports the full generality of C (varargs,setjmp/longjmp,casts,…)



DSA Algorithm Highlights

Basic algorithm design:

- Context-sensitive, unification-based, flow-insensitive algorithm
- Provides speculative type information and field-sensitivity
- Computes which memory is passed into/out of the analysis region

Bottom-Up phase computes Fn behavior with all callees

- Computes "total effect" of calling the function
- Incrementally constructs program call graph
- BU results are used by Pool Allocation & Pointer Compression

Top-Down phase adds information from callees

- BU computes no information about callers of a function
- TD pass is useful for alias analysis clients

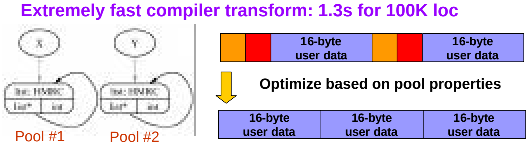
See llvm-tv demo for more examples of graphs

Automatic Pool Allocation [PLDI'05]

Allocate memory from pool instead of the heap:

- Partition distinct data structures in memory
  - Better for cache, locality, allocation speed, etc
- Give compiler information about dynamic location of memory
  - Needed to perform memory layout optimizations at runtime
- Give compiler control over layout of data structure
  - Can segregate or collocate nodes in the RDS
  - Can optimize away inter-object padding in many cases (below)

Extremely fast compiler transform: 1.3s for 100K loc



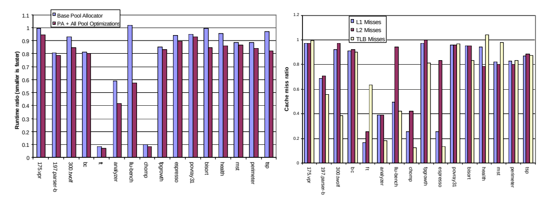
Pool Allocation Performance Effect

Pool Allocation & optzns improve RDS performance:

- 10-20% in many cases, ~2x in 2 cases, > 10x in two cases

Biggest source of speedup is cache and TLB effects:

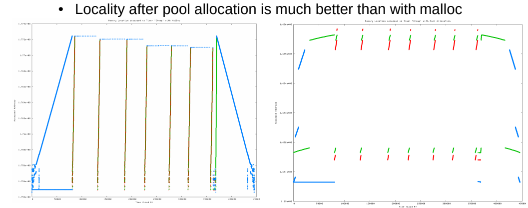
- Deinterlacing disjoint data structures, reducing inter-object padding



Pool Allocation Locality Effect

Graph Load Addresses vs Program Time: (for "chomp")

- 3 linked lists: Pool allocation segregates them into distinct pools
- With malloc, green and red nodes are interlaced with each other
  - Traversal of one brings the other into cache (green/red overlap)
- Locality after pool allocation is much better than with malloc



access pattern with malloc **access pattern with poolalloc**

Transparent Pointer Compression [MSP'05]

Problem: 64-bit pointers cost 2x as much as 32-bit ptrs

- Reduces effective cache capacity and memory bandwidth

Idea: Reduce 64-bit pointers to 32-bit pool indices

- Use pool allocation to segregate data structures
- Pointer dereferences become *(PoolBase+Idx) instead of *Ptr

Implementation: Interprocedural Restructuring xform

```
std::set<float> T
```

implemented as

```
struct tree {
  tree* Left;
  tree* Right;
  tree* Parent;
  float data;
};
```

sizeof(set<float>) = 32

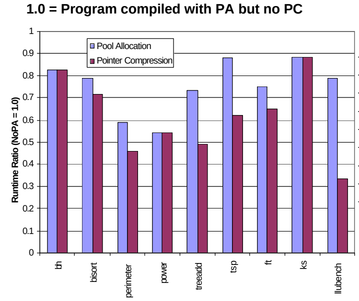
pointer compressed

```
struct tree {
  uint32 Left;
  uint32 Right;
  uint32 Parent;
  float data;
};
```

sizeof(set<float>) = 16

Pointer Compression Perf. Impact

1.0 = Program compiled with PA but no PC



Peak Memory Usage

	PA	PC
bh	7.7MB	7.7MB
bisort	48MB	24MB
perimeter	256MB	149MB
power	924KB	854KB
treeadd	96MB	48MB
tsp	131MB	114MB
ft	8.9MB	4.5MB
ks	47KB	47KB
llubench	3MB	1.5MB

UltraSPARC IIIi w/1MB Cache

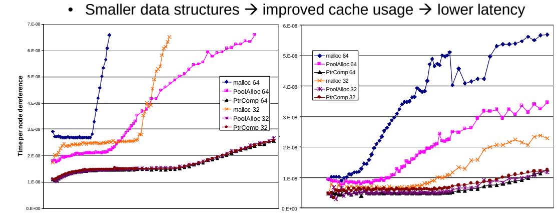
Load Latency vs Heap Size

How does ptr comp vary with heap size & architecture?

- Methodology: take a small pointer intensive program, vary input size

Pointer comp. can double performance over pool alloc

- Smaller data structures → improved cache usage → lower latency



SparcV9 Performance Scaling **AMD64 Performance Scaling**

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

http://llvm.cs.uiuc.edu