Precomputing Memory Locations for Parametric Allocations

Jörg Herter Sebastian Altmeyer

Department of Computer Science
Saarland University

WCET Workshop, July 2010

Current Situation

What we have ...

1. Sound and precise WCET analysis
2. Dynamic Memory Allocation
   - often clearer program structure
   - easy memory reuse (e.g. in-situ transformations)

... but can we have both together?

Why is DSA Problematic? An Example ...

Consider a program that ...

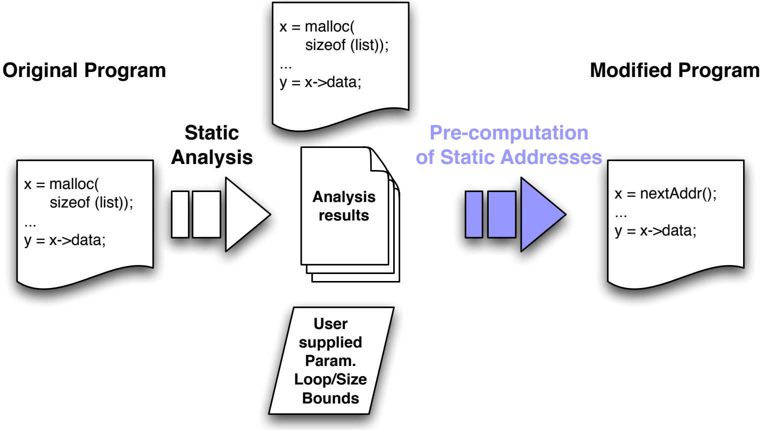
1. builds-up an internal data structure A
2. works on A
3. transforms this structure to another data structure B
4. works on B

Why is DSA Problematic? An Example ...

What are the challenges for a WCET analysis?

1. builds-up an internal data structure A
   $\rightarrow$ unknown cache state after allocation
2. works on A
   $\rightarrow$ unknown cache set mappings of list elements
3. transforms this structure to another data structure B
   $\rightarrow$ unknown cache state after (de-)allocation
4. works on B
   $\rightarrow$ unknown cache set mappings of list elements

Big Picture


Original Program
x = malloc(
sizeof (list));
...
y = x->data;
Static Analysis
x = malloc(
sizeof (list));
...
y = x->data;
Analysis results
User supplied Param. Loop/Size Bounds
Pre-computation of Static Addresses
Modified Program
x = nextAddr();
...
y = x->data;

What are *Good* Memory Addresses?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

What do we consider good memory addresses for heap allocated objects?

- Good addresses enable a subsequent WCET analysis to calculate tight WCET bounds,
- exhibit optimal cache performance by
- minimal memory consumption.

What are *Good* Memory Addresses?

Unfortunately, optimizing ...

- for tight WCET bounds requires knowledge about subsequently applied analyses
- for cache performance too costly to compute[1]
- for memory consumption is (still) NP hard

[1] Petrank and Rawitz: The hardness of cache conscious data placement, 2005

What we can do ...

We can use

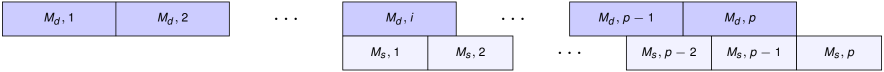
- simple heuristics[2] for achieving *good* cache performance
- and only approximate minimum memory consumption.

[2] Chilimbi et al: putting temporaneously accessed objects adjacent in memory

An Example: Intuitive Solution

Assume A and B to be a singly- and a doubly-linked list, respectively.

An intuitive, memory and cache optimal memory placement is:



An Example: Normalization

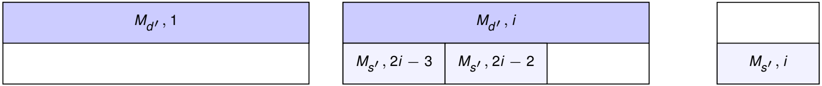
Assume a target hardware with cache line size 32 bytes and let the algorithm

- consider 4 consecutive elements of A as a single object
- consider 8 consecutive elements of B as a single object

According to our heuristics, all cache benefits are now already exploited!

An Example: Computed Patterns

Computed patterns (chunks) for our list-copy example:



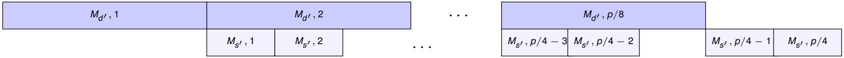
1 repetition

$(p/8-1)$ repetitions
$i \in [2; p/8]$

2 repetitions
$i \in [p/4-1; p/4]$

Examples—In-Situ List Copy

Putting these chunks consecutively in memory yields ...



This allocation scheme is

- almost memory optimal and
- cache optimal.

Quick Summary

How Does Our Algorithm Work?

- start with formal description of a program's allocation behavior
- normalize according to target hardware
- compute conflict free 'patterns'
- suitable allocation scheme is any concatenation of these patterns

Conclusions & Future Work

- novel algorithm to statically precompute memory addresses
- less limitations than previous approach
- promising results for benchmarks

- Future Work:
  - full evaluation of the approach
  - fully automatize whole work chain

An Example: What was gained?

Program	Memory Allocation	WCET Bound (cycles)
In-Situ List-Copy	TLSF	111,698,519
	TLSF*	1,710,656
	Pre-computed	830,202