Slides for some talk

Marc Friedman
friedman@cs

Runtime Code Generation: Goals

To defer compilation of key program fragments until runtime, when runtime constants become known.

Benefits:

Runtime constants can become immediates, instead of memory fetches.

New constants can be folded and propagated, rather than computed again and again.

If loop bounds are runtime constants, loops can be fully unrolled.

If predicates are runtime constants, conditional branches become unconditional, unreachable code is thrown away, and layout may be improved.

Limitations of the First Cut

Runtime constants must be marked by the programmer.

Runtime constants must be scalars.

Initially, runtime compilation of a *dynamic block* occurs the first time the block is run.

Runtime constants must not be modified after that time.

Overall Design Outline

Changes to C source code

The Static Compiler

Templates: The static/dynamic compiler interface

Dynamic compiler

Changes to C source code

```
main(int argc)
{
  int a = 5, b = 2;
#pragma runtime constant a b
#pragma dynamic block
  if (argc == 2)
    goto plus;
  else
    goto minus;
  exit(0);
 plus:
  printf("%d", a + b);
 minus:
  printf("%d", a - b);
}
```

The Static Compiler

The static compiler does most of the work of dynamic compilation ahead of time.

It's gcc, with a few twists.

For *dynamic blocks*, it produces *templates*, a low-level IR.

It does constant folding of runtime constants before they are known, leaving labeled holes in the templates to be filled in with values by the dynamic compiler.

It does loop unrolling by a fixed constant number ahead of time, producing multiple versions of a loop to be pasted together by the dynamic compiler.

It produces code sequences to compute values dependent on the runtime constants (*computed constants*), and a map from those values to the places in the template they will go.

Templates

The output of the static compiler for a dynamic block consists of:

A graph of machine code templates with holes for runtime constants.

A code sequence (without side effects) which computes *computed constants*

A patch table: where and how to put these constants in the templates, and how to patch loops

The address of the jump to the dynamic compiler

What does this have to do with SPIN?

If SPIN has heavily-used, rarely modified globals, it should achieve speedups.

If we extend the compiler to allow data structures as runtime constants, that should get even more speedups. (Flat or deep?)

Templates may be help optimize the dispatcher.

Some unsolved problems

Compiling a region at runtime must not have any side effects that may not have arisen normally. Hence, division by zero is a problem.

You can only load up to 8-bit immediates in one instruction on Alpha. Larger values require 5 (?) instructions or a memory reference.

GCC currently produces unrelocatable code for switch statements.

And many more...

What's to do?

Recognize pragmas; represent in RTL

Find workload; identify good dynamic blocks and runtime variables

Do some constant folding in the static compiler

Generate code with templates

Write dynamic compiler

Run some experiments