GNU Octave - JIT Compiler

Free your numbers *faster*

Max Brister

July 18, 2012

Just In Time

- Compile during execution
- Use runtime information to help compilation
- Goal: Produce code “As fast as C” or faster?

Speedup?

- Where does the speedup come from?

```
a = b + c;
```

- What if we don't know the type of a and b?
- We might as well interpret

Speedup?

```
a = a + b;
```

- What if a and b are scalars?
- We can save a few function calls, a table look up, and a malloc
- Not very much overall time saved
- Large percentage time saved
- Large compile overhead

Intermediate Representations (IR)

- Octave's parse tree
- New Octave linear IR for type inference
- LLVM's SSA based IR

Operations/Functions

```
y = sin (x);
```

- sin is a matrix or function?
- Convert to $y = paren(sin, x)$;
- Infer to $scalar : y = paren(sin : sin, scalar : x)$;

Type Inference - Simple Version

- Assume each variable has one type
- Fail if broken
- Problems?

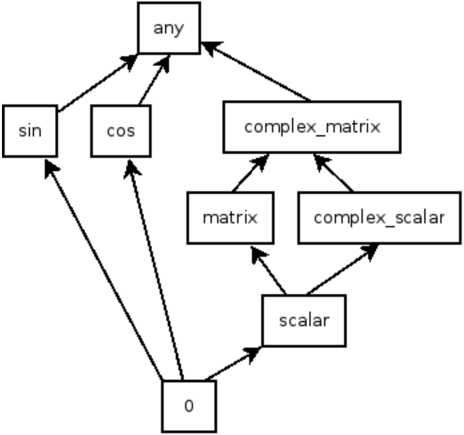
```
for i=1:n
  ...
  for j=1:n
    temp = ...
    ...
  endfor
endfor
```

Static Single Assignment (SSA)

- Only allow once assignment to a variable
- Now each variable can only have one type
- Use a ϕ function to merge branches

```
if:
  a.0 = 5
else:
  a.1 = 6
if_tail:
  a.2 = phi (a.0, a.1)
```

Type Lattice



Type Assignment

- Operations - entry in a table
- Phi - join operator
- Backwards branches change already processed types
- Iterative algorithm - worst case depends on lattice depth

Future

- Sparse Conditional Constant Propagation
- Represents constants in a type lattice

```
for i = 1:n
  if nargout > 1
    b(i) = 100;
  endif
  a(i) = 5;
endfor
```

- Over specialization problem

LLVM

- Provides traditional compiler optimizations
- Easy interface for JIT
- Move to GCC?

```
llvm::Function *llvm_func = ...;
optimizations->run (llvm_func);
fptr = eng->getPointerToFunction (llvm_func);
```

Octave linear IR to LLVM IR

- Implement each operation in LLVM

```
scalar: a = +(scalar: b, scalar: c)
to
a = call add_scalar_scalar (b, c)
to
a = fadd b, c
```

- Sometimes just a function call to C
- Represent types as simple structs

Variable or Function?

```
function x = foo
  eval ('bar = 5');
  x = bar;
end
```

- Different in MATLAB and Octave

```
function x = foo
  eval ('AA = 5');
  x = AA;
end
```

- Same in MATLAB and Octave

- Complex matrix assignment
- Debugging?
- Types for .oct functions?