Metaccompilation: from Brook to SVM

Benjamin Chelf
March 12, 2002
Big Picture

Brook

IR

SRF Language

C Code

SSS
Who does what?

Brook

IR

SVM IR

C Code

SSS

Ian

Ben C.

Mattan

Francois

???
Goals for Metaccompilation

- Successfully parse Brook code
- Warn on incorrect code (syntax & semantics)
- Maintain representation for SVM and provide access to all necessary information
- Output C++ code for runtime library


```plaintext

def annotations {
    decl:
        "out", "in", "[%d..%d]" =>
            mc_is_parm (mc_stmt), msg1 &&
            is_stream (mc_type (mc_stmt)), msg2 &&
            is_kernel (cur_func ()), msg3;

    function:
        "kernel";

    typedef:
        "stream";
}
```
Checking Brook

sm typecheck {
    all:
    ${mc_is_modify_expr (mc_stmt) &&
    mc_tag_comp (mc_lhs (mc_stmt), "in")}
    ==> 

    {}
    mc_warn (mc_stmt,
    "Not allowed to assign to an in")
    
}

Also check that an 'out' is only written to once
Some necessary information

- kernel functions

  Arguments
  - Names
  - Stream type or other?
  - in, out, reduce, stencil?

Code

- main

  Flow of streams through kernels
A simple example: Addition

typedef stream float * floats;

kernel void Add (floats A, floats B, out floats C) {
    *C = *A + *B;
}

kernel void FloatsPrint (floats D) {
    printf ("%f\n", *D);
}

kernel void FloatsLoad (FileStream fp, out floats E) {
    float x;
    brfscanf (fp, "%f\n", &x);
    *E = x;
}
int main () {
    floats a;
    floats b;
    floats c;
    FileStream fp1 ("floatsA.txt", "rt");
    FileStream fp2 ("floatsB.txt", "rt");
    FloatsLoad (fp1, a);
    FloatsLoad (fp2, b);
    Add (a, b, c);
    FloatsPrint (c);
    return 0;
}
Flow of function calls

1. FloatsLoad (fp1, a);
2. FloatsLoad (fp2, b);
3. Add (a, b, c);
4. FloatsPrint (c);
Flow of streams

Flowchart:
- **FloatsLoad (fp1, a);**
- **FloatsLoad (fp2, b);**
- **Add (a, b, c);**
- **FloatsPrint (c);**
Output for runtime library

- Developers need to test code

- We have the IR, why not output C++ code?

- Current implementation is 200 lines of code

- Works in conjunction with runtime library
Goals for Metaccompilation (revisited)

- Successfully parse Brook code
  10 lines of code
- Warn on incorrect code
  Approximately 10 to 20 lines of code per check
- Maintain representation for SVM
  35 lines of code to calculate stream dependences
- Output C++ code for runtime library
  Currently 200 lines of code
What's next?

- Continue updating syntax as Brook evolves
- Add more Brook-specific semantic checks
- Work with Mattan and Francois providing them with the information from the IR
- Continue the maintenance of code generation for developers of Brook applications