近況報告

Tatsurou Sekiguchi

Recent Activity

- コンパイラを実装中 (since last October)
  - 9000 lines of code in Ocaml
- どんな?
  - Compiler backend として使える
    - SPARC, IA32, PowerPC などのアーキテクチャに対応
    - メモリを直接触れる
    - ランタイムが GC を提供できるようなコード生成
  - いま流行の安全性保証(PCC, TAL)はしないが拡張できる余地を残す
- 目標は?
  - 効率で GCC に勝つ

Why not?

- Using a JVM
  - Too slow even with JIT compilation especially for C-style programs
- Using a TAL
  - Too technically difficult to extend it
- Translating into the C language
  - Difficult to implement precise GC

Relevant Work and Systems

- Mobile computation
- GNU C compiler or MIC
- Compiler toolkits: ANDF, SUIF, Vortex
- .NET
- C--
- MLRISC

Mobile Computation

- Platforms
  - Omniware, vcode, etc.
- 共通点
  - Execution on multiple platforms
- 相違点
  - We don't need
    - Exactly same virtual machine
      - Integer size, endianness, data representation, etc.
    - Very fast compilation, but need efficient code

GNU C Compiler

- 共通点
  - Efficient execution rather than quick compilation
  - Native code generator for multiple platforms
- 問題点
  - Too complex (more than 0.3M lines of code)
  - Old fashioned (SSA を未使用)

MIC (1/2)

- JIT コンパイラ
  - 筑波大のプロジェクトのために作成
  - C プログラムのモーバイル計算で使用
- 構成
  - Modified GCC
    - 機種共通のバイトコード生成
  - Native code 生成器
    - Register allocation, peephole optimizations のみ

MIC (2/2)

- この構成の利点
  - GCC の最適化コードをそのまま流用
    - SPARCv8 ではほぼ同じ実行効率を達成
  - GCC のパーザーをそのまま流用
    - GCC がコンパイルできるものは実行可
- この構成の欠点
  - Register allocation 後の最適化が弱い
    - IA32 では spill が頻出 → 40% の速度低下
  - GC のサポートが困難
    - デバグ情報はユーザ定義変数のみ

Compiler Toolkits

- ANDF (developed by OSF)
  - Architecture and language Neutral Distribution Format
  - Superset of the C language
- SUIF
  - National Compiler Infrastructure Project (立ち消え?)に採用
  - Framework for parallel and optimizing compilers
  - Portability is not strongly supported
  - ツールキットは C 言語で実装
- Vortex
  - Compiler framework for object-oriented languages
  - Portability is not strongly supported (SPARC only?)

.NET

- Much similar to JVM
- Multiple languages will be supported
  - VB, C#, C, C++, Fortran, etc.
- Multiple platforms will be supported
  - Currently IA32 only
- Two modes
  - Managed: pointer arithmetic forbidden
  - Unmanaged: GC is not supported

C--

- Compiler backend for multiple languages, multiple platforms
- Still under development (prototypes are available)
- No support for garbage collection
- Weak (or rich?) support for concurrent execution
  - No support for native threads
  - The programmer can provide a scheduler in user level

MLRISC

- Compiler backend for SML/NJ
- Weak documentation, no Makefile
- Optimizations: (we don't implement some of them)

Dead code elimination	Global code motion
Register allocation	Conditional constant propagation
Global value numbering	Strength reduction
Constant folding	List scheduler
Algebraic simplification	VLIW support

Time-constrained instruction scheduler

Our Language System

- Compiler backend language
  - Explicit access to memory
  - Pointer arithmetic
  - No verification, but support for verification
  - Weak type checking
  - Architecture neutral
- Native code generator
  - Should do only architecture dependent optimizations
  - 標準的な最適化を実装する

Surface Language

- C-like syntax and semantics
  - No block of statements
- Weak type system
  - Only integer types and floating-point types
- Excluding ambiguities in the C language
  - Evaluation order (function arguments, ++, --, etc.)
  - Structure layout
  - Integer bitwidth, pointer bitwidth
- New features:
  - Exception handling
  - Built-in tail call instruction

Sample: fibonacci function

```
int32       fib( int32 n ) {
      int32       p, q;
      if ( n > 1 ) goto l;
      return 1;
  l:  p = fib (n - 1);
      q = fib (n - 2);
      return p + q;
}
```

Exception Handling

- Source code

```
...
x = foo() => k;

z = x + 1;
handler k(y):
...
```

- Implementation

```
...
call foo
ba k
add x,1,z
k: mov r,y
...
```

exceptional return

normal return

Built-in Tail Call Instruction

- Function call with a current stack frame
- Syntax:
  - Normal call: foo();
  - Tail call: jump foo();
- Adopted in C-- and .NET
- Purposes:
  - Tail call optimization
  - CPS-style program handling

Native Code Generator (1/2)

- 最適化の実装方針
  - アーキテクチャ依存のものだけ
  - Too advanced, too costly なものは除外
    - e.g. 線形計画問題を利用した命令スケジューリング
  - 最近の最適化技術を積極的に採用

Native Code Generator (2/2)

- DON'T
  - Interprocedural analysis and optimizations
  - Global code motion
  - Inlining
  - Loop unrolling
- DO
  - SSA transformation
  - Algebraic simplification
  - Constant propagation
  - Strength reduction
  - Liveness analysis
  - Register allocation
  - Induction variable elimination
  - Laze code motion
  - Machine SSA transformation
  - Critical path based list scheduling

SSA

- SSA とは?
  - 同じ変数への代入が一つしかないプログラム (static single assignment form)
  - CPS とほぼ同等 [Kelsey95][Appel98]
- 特徴
  - Flow insensitive にプログラムを扱える
  - それにも関わらず flow sensitive に扱ったのと同等の効果
- Machine SSA とは?
  - 機械語が特定の register を要求する場合を配慮した SSA (%y register in SPARC, edx:eax in mul of IA32)

Flow Insensitivity

- Non SSA

```
L0: if x < 0 goto L2;
L1: y = 10; goto L4;
L2: y = 20;
L3: printf( "%d¥n", y );
L4: z = x;
```

- SSA

```
L0: if x < 0 goto L2;
L1: y1 = 10; goto L4;
L2: y2 = 20;
L3: printf( "%d¥n", y2 );
L4: z = φ(y1,y2);
```

SSA and Functional Programming

- SSA

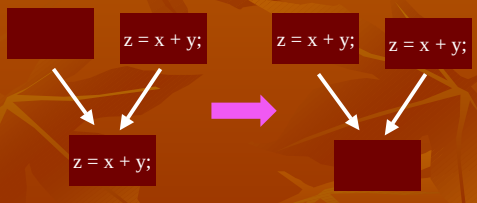
```
L0: if x < 0 goto L2;
L1: y1 = 10; goto L3;
L2: y2 = 20;
L3: y3 = φ(y1,y2);
L4: printf( "%d¥n", y3 );
```

- Functional equivalent

```
let l0(x) =
  let l3(y3) =
    printf( "%d¥n", y3 )
  in
  if x < 0 then
    let y1 = 10 in l3(y1)
  else
    let y2 = 20 in l3(y2)
```

Lazy Code Motion
(partial redundancy elimination)

- Generalization of loop invariant hoisting, common subexpression elimination ...



Progress

- SSA transformation
- Algebraic simplification
- Constant propagation
- Strength reduction
- Liveness analysis
- Machine SSA transformation

done

- Register allocation
- Induction variable elimination
- Laze code motion
- Critical path based list scheduling

on going

Summary

- A language system is under development
  - Requirements:
    - Highly efficient, and
    - Highly portable
  - Components
    - Surface language for compiler backend
    - Native code generator (JIT compiler)

Mac OS X is Linux PPC + MS Office

- Mac はもはや昔の(不便な) Mac ではない!
- UNIX らしい特徴
  - Open source
  - emacs, tcsh, ssh が標準でインストール済み
  - C compiler, X window system がただでダウンロード可能
  - Control キーが正しい位置にある
  - 標準で Samba をサポート
- UNIX を越える特徴
  - Powerpoint, Internet Explore が動作!