SSAにより強力な最適化が行われるようになった

シリーズの新機能の実証

（前編）

岸 哲夫

2006年5月24日にGCC4.1.1がリリースされました．このGCC4シリーズは従来のものより格段に進歩しているとのことです．この記事ではGCC4シリーズの新機能について2回に分けて概要，ベンチマーク，インストール方法を証明します．（筆者）

GCC4 シリーズの概要

● SSA：静的単一代入形式について

従来のGCCでは，まずCソースを中間言語（RTL：Register Transfer Language）に変換し，その後アセンブリ・コードに変換しています．次に内部でアセンブルを行います．中間言語の段階で構文解析を行って最適化が行われていました．また，ハードウェアに依存する最適化もこの時点で行われました．

GCC4では「tree-ssa」という「木」の概念を元に最適化します．今までよりも少し効率が良くなると思います．ここで「tree-ssa」のSSAは，静的単一代入形式を意味します（詳しくは後述）．

オプションの`-dr`を付加してコンパイルすると，`*.expand`というLisp風言語のソース・ファイルが作成されます（**リスト1**，**リスト2**）．これが中間言語のソースです．

従来のSSA最適化のオプション`-fssa`は使えません．新バージョンでは最適化を行うとあたりまえのようにSSAを適用するからです．

この場合，GCC内部ではフロントエンドが中間言語（RTL）を生成し，バックエンドがRTLからアセンブリ・コードを生成します（**リスト3**）．

リスト1　元のCソース

```
//ピープ・ホール最適化をする例
#include <stdio.h>
main()
{
    int x   =   0;
    int y   =   0;
    x   =   x   +   1;
    y   =   x^2;
    printf("%d¥n",x);
    printf("%d¥n",y);
    y   =   y   +   0;
    y   =   y   +   0;
    y   =   y   +   0;
    y   =   y   +   0;
    y   =   y   +   0;
    goto L1;
    return;
L1:
    return;
    printf("%d¥n",x);
}
```

RTLの段階ではマシン非依存になっています．RTLを理解できる人は，ここで人手による最適化を行ってもかまいません．

GCCではC言語以外にもJavaやFortranなどが扱えますが，RTLを作るまでがフロントエンドのしごとなので，ここまではコンパイル対象の各言語に依存しますが，それ以降は共通の処理となります．

つまり，今回の変更はC言語のみならず，ほかのGCCで扱える言語に関する最適化効率も上がるはずです．

さて，実際にSSAを使った最適化について説明します．

```
a = x + 2;
a = a + 1;
```

リスト3　アセンブリ・コード

```
    .file   "test230.c"
    .def    ___main;    .scl    2;  .type   32; .endef
    .section .rdata,"dr"
LC0:
    .ascii "%d¥12¥0"
    .text
.globl _main
    .def    _main;  .scl    2;  .type   32; .endef
_main:
    leal    4(%esp), %ecx
    andl    $-16, %esp
    pushl   -4(%ecx)
    pushl   %ebp
    movl    %esp, %ebp
    pushl   %ecx
    subl    $36, %esp
    call    ___main
    movl    $0, -12(%ebp)
    movl    $0, -8(%ebp)
    incl    -12(%ebp)
    movl    -12(%ebp), %eax
    xorl    $2, %eax
    movl    %eax, -8(%ebp)
    movl    -12(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    $LC0, (%esp)
    call    _printf
    movl    -8(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    $LC0, (%esp)
    call    _printf
L2:
    addl    $36, %esp
    popl    %ecx
    popl    %ebp
    leal    -4(%ecx), %esp
    ret
    .def    _printf;    .scl    2;  .type   32; .endef
```

リスト2　中間言語(RTL)ソース

```
;; Function main (main)


;; Generating RTL for tree basic block 0

;; Generating RTL for tree basic block 1


;;
;; Full RTL generated for this function:
;;
(note 2 0 7 NOTE_INSN_DELETED)

;; Start of basic block 0, registers live: (nil)
(note 7 2 3 0 [bb 0] NOTE_INSN_BASIC_BLOCK)

(insn 3 7 4 0 (set (reg:SI 59)
        (reg:SI 2 cx)) -1 (nil)
    (nil))

(note 4 3 5 0 NOTE_INSN_FUNCTION_BEG)

(note 5 4 6 0 NOTE_INSN_DELETED)

(call_insn 6 5 8 0 (call (mem:QI (symbol_ref:SI ("__main") [flags 0x41]) [0 S1 A8])
        (const_int 0 [0x0])) -1 (nil)
    (expr_list:REG_EH_REGION (const_int 0 [0x0])
        (nil))
    (nil))
;; End of basic block 0, registers live:
 (nil)

;; Start of basic block 1, registers live: (nil)
(note 8 6 10 1 [bb 1] NOTE_INSN_BASIC_BLOCK)

(insn 10 8 12 1 (set (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])
        (const_int 0 [0x0])) -1 (nil)
    (nil))

(insn 12 10 14 1 (set (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 y+0 S4 A32])
        (const_int 0 [0x0])) -1 (nil)
    (nil))

(insn 14 12 16 1 (parallel [
            (set (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])
                (plus:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) -1 (nil)
    (nil))

(insn 16 14 17 1 (parallel [
            (set (reg:SI 60)
                (xor:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])
                    (const_int 2 [0x2])))
            (clobber (reg:CC 17 flags))
        ]) -1 (nil)
    (nil))

(insn 17 16 19 1 (set (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffc])) [0 y+0 S4 A32])
        (reg:SI 60)) -1 (nil)
    (expr_list:REG_EQUAL (xor:SI (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                    (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])
            (const_int 2 [0x2]))
        (nil)))

(insn 19 17 20 1 (set (reg:SI 61)
        (mem/c/i:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -8 [0xfffffff8])) [0 x+0 S4 A32])) -1 (nil)
    (nil))

(insn 20 19 21 1 (set (mem:SI (plus:SI (reg/f:SI 56 virtual-outgoing-args)
                (const_int 4 [0x4])) [0 S4 A32])
        (reg:SI 61)) -1 (nil)
    (nil))
```