Bringing PCC into The 21th century

Anders Magnusson

October 11, 2008

About PCC

- Written in the mid-late-70's by S.C. Johnson as a portable and retargetable C compiler.
- Based on theory from the Richie PDP-11 C compiler and Andy Snyder's 1975 master thesis on portable C compilers
- Was the reference implementation of C compilers and was ported to almost any existing architecture.
- Was the system C compiler on almost all Unix systems (on some still are!)

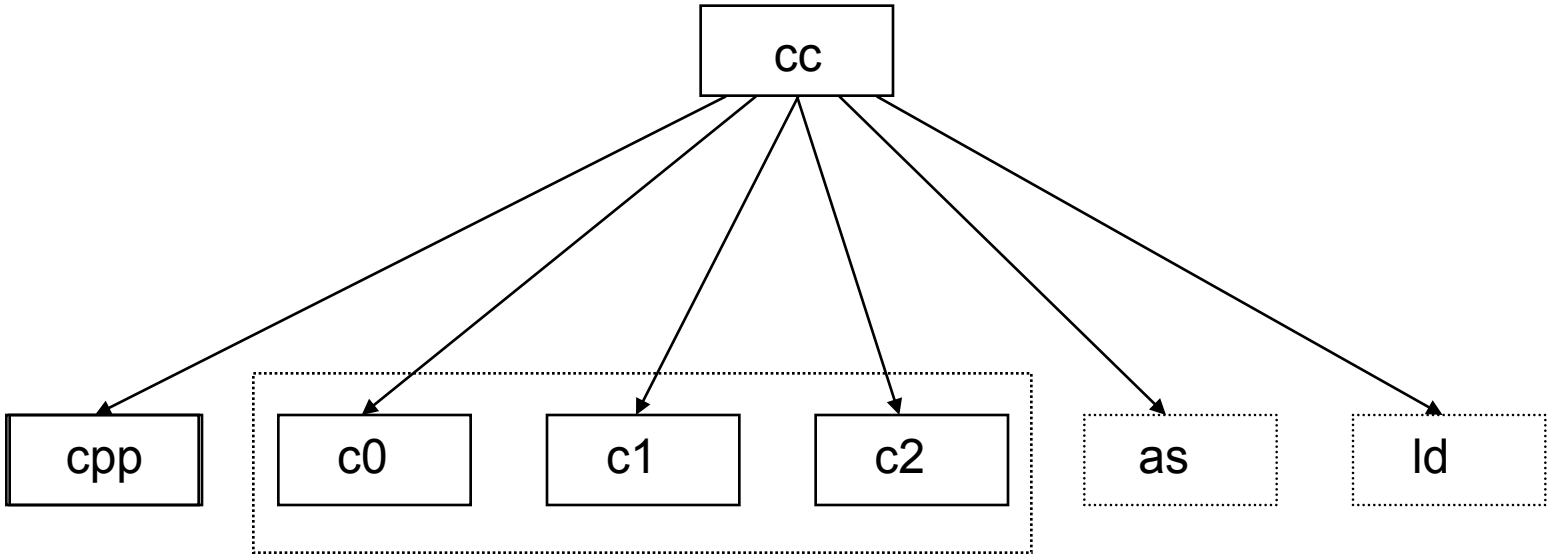
What have I done?

- Write a preprocessor that supports C99 features.
- Add the C99 features to the C compiler parser step (frontend).
- Rewrite the code generator (backend) almost entirely to be able to do optimizations.

Why?

- Needed a C compiler for PDP10 to be able to port NetBSD to it.
- Wanted a better C compiler than the Richie C compiler for PDP11.
- PCC was just released freely by Caldera.
- Have a slight interest in compilers.

Layout of a C compiler



cpp – The C PreProcessor

c0 – Parser and tree builder

c1 – Code generator

c2 – peephole optimizer

as – assembler

ld – linkage loader

PCC is small and simple

- The compiler consists of 30 files.
- The total size of the machine-independent part of the compiler is 15000 lines of code, 9000 in the C language parser and 6000 in the code generator.
- The machine-dependent part is 3000 lines, where 1000 is the C-specific code and 2000 is for the code generator.

C Parser step overview

- Handles all initializations and data segment allocations
- Does syntax checking of the compiled code, prototype checks and casts
- Builds parse trees, inserts casts, converts array references to register offset arithmetic
- Converts language-specific operators (comma operator, lazy evaluation) to non-C-specific code
- Keep track of the symbol table and the different name spaces
- Generates debugging information

C Parser machine-independent files

```
-rw-r--r--  1 ragge  wheel  31746 Sep  5 19:07 cgram.y
-rw-r--r--  1 ragge  wheel   3169 Oct  4  2004 gcc_compat.c
-rw-r--r--  1 ragge  wheel  17603 Apr  2  2005 init.c
-rw-r--r--  1 ragge  wheel   4133 May 19 22:52 inline.c
-rw-r--r--  1 ragge  wheel   7870 Sep  5 19:07 main.c
-rw-r--r--  1 ragge  wheel   7622 May 19 22:52 optim.c
-rw-r--r--  1 ragge  wheel   9701 Sep  5 19:07 pass1.h
-rw-r--r--  1 ragge  wheel  46282 Sep  5 19:07 pftn.c
-rw-r--r--  1 ragge  wheel  10216 Dec 11  2004 scan.l
-rw-r--r--  1 ragge  wheel   8956 May 21 10:31 stabs.c
-rw-r--r--  1 ragge  wheel   8371 Oct  3  2004 symtabs.c
-rw-r--r--  1 ragge  wheel  47022 Sep  5 19:07 trees.c
```

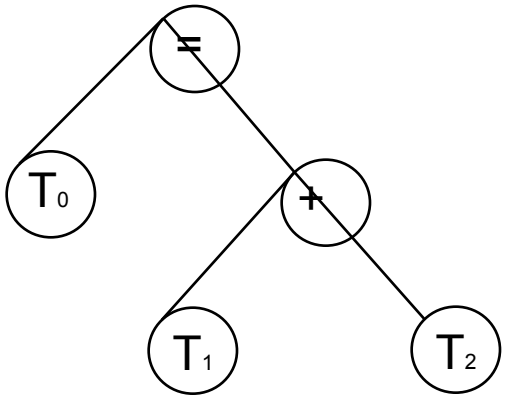
Parser step MD code

- 30 machine-dependent functions for the C parser, most of them can be copied.
- Function `clocal()` is called after each tree node is added to be able to do fast rewrite of trees.
- Only two files are cpu-specific

```
-rw-r--r--  1 ragge  wheel  11487 Oct  3 18:08 local.c
-rw-r--r--  1 ragge  wheel   5016 Sep  5 19:07 code.c
```

Internal tree structure

- The compiler builds binary trees in the parser step
- These trees follows through the compiler



Internal tree structures

- A node always have at least two properties
  - op – the operation the node is supposed to perform (PLUS, REG, ASSIGN, …)
  - type – the underlying (C) type of the operand (int, float, char *, …)
- Nodes are of three sorts
  - BITYPE – binary, node with two legs
  - UTYPE – unary, left is a leg
  - LTYPE – leaf, no legs
- A specific node op is always one of the above.

Nodes

- BITYPEs
  - PLUS, MINUS, DIV, MOD, MUL, AND, OR, ER, LS, RS, INCR, DECR, EQ, NE, LE, LT, GE, GT, ULE, ULT, UGE, UGT, CBRANCH, CALL, FORTCALL, STCALL, ASSIGN, STASG
- UTYPEs
  - COMPL, UMUL, UMINUS, FLD, SCONV, PCONV, PMCONV, PVCONV, UCALL, UFORTCALL, USTCALL, STARG, FORCE, GOTO, FUNARG, ADDROF
- LTYPEs
  - NAME, ICON, FCON, REG, OREG, TEMP

UTYPEs

- UMUL
  - Take value pointed to by expression
- FLD
  - Use only some bits in expression
- SCONV, PCONV
  - Convert expression value to scalar/pointer
- PMCONV, PVCONV
  - Multiply/divide expression for array reference
- STARG, FUNARG
  - (Structure) argument to function
- ADDROF
  - Take address of expression
- FORCE
  - Value should be put into return register

LTYPEs

- NAME
  - Reference to the data stored at an address in memory.
- ICON, FCON
  - A constant of some type. May be an address in memory.
- REG
  - A hardware register on the target machine.
- OREG
  - An offset from a register to a memory position, like the stack or in a structure.
- TEMP
  - A temporary variable generated by pass1 that is later converted to either a REG or an OREG.

The ‘NODE’

- The NODE typedef is the basic structure used through the compiler in both the parser and the code generator

```
typedef struct node {
  int    n_op;
  int    n_rall;
  TWORD  n_type;
  int    n_su;
  union {
         char *       _name;
         int   _stsize;
         union        dimfun *_df;
  } n_5;
  union {
         int   _label;
         int   _stalign;
         struct       suedef *_sue;
  } n_6;
  union {
         struct {
               union {
                     struct node *_left;
                     CONSZ _lval;
               } n_l;
               union {
                     struct node *_right;
                     int _rval;
                     struct symtab *_sp;
               } n_r;
         } n_u;
         long double       _dcon;
  } n_f;
} NODE;
```

Code generator steps

- There are four basic functions in the code generation pass, called in order (sort of)
  - `geninsn()`
    - Finds instructions that covers as much as possible of the expression tree; ``maximal munch''
  - `sucomp()`
    - Does Sethi-Ullman computation to find best sub-tree evaluation order
  - `genregs()`
    - Uses graph-coloring to do register assignment
  - `gencode()`
    - Emits the instructions and removes redundant code

Instruction selection

- The basic principle of the compiler is something like ”get a value into a register, work on it, and then write it back”. Matches RISC targets very well.
- Instruction selection is the first step in code generation.
- Assigning instructions is done by matching the trees top-down to find an instruction that covers the largest part of the tree.

Instruction selection #2

- If several instructions matches, the best instruction is selected based on some heuristics (other needs etc), or just the position in the table.
- To be kind to CISC targets with funny addressing modes, special target-dependent functions can be written to match indirect references:
  - shumul() finds out if a shape matches
  - offstar() sets the subtree into a usable state
  - myormake() will do the actual subtree conv.

Sethi-Ullman calculations

- Sethi-Ullman calculations is a way to find out how many registers needed to evaluate a parse tree on a simple architecture.
- It is usually used to see if a subtree must be stored to be able to evaluate a full tree.
- In PCC Sethi-Ullman is only used to find out in which order subtrees should be evaluated.
- Numbering of in-tree temporaries is done here.

Register assignment

- The current register allocator uses graph-coloring based on the George and Appel pseudocode from their ACM paper.
- Extensions to handle multiple register classes are added, with some ideas from a Smith, Holloway and Ramsey ACM paper but in a better and simpler way :-)
- If register allocation fails, `geninsn()` and `sucomp()` may have to be called again.

Instruction emitting

- Emitting of instruction is done bottom-up in the order found by `sucomp()`. Tree rewriting is used.
- Redundant code from the register allocation phase (reg-reg moves) are removed here (unless condition codes is needed)

Optimizations

- When optimizing is enabled, the C language parser will count all variables as temporaries and let the register allocator try to put them in registers.
- Redundant jumps (to next insn) are deleted.
- The trees are divided in basic blocks and a control-flow graph is built.
- The trees are converted in SSA form (not yet finished).

Code generator files

- Machine-independent

```
-rw-r--r--  1 ragge  wheel  12587 Sep  5 19:07 common.c
-rw-r--r--  1 ragge  wheel   8837 Sep 17 09:58 manifest.h
-rw-r--r--  1 ragge  wheel  19438 Oct  6 19:56 match.c
-rw-r--r--  1 ragge  wheel   4133 Sep 12 09:02 mkext.c
-rw-r--r--  1 ragge  wheel   4016 Feb  5  2005 node.h
-rw-r--r--  1 ragge  wheel  20153 Sep 17 09:58 optim2.c
-rw-r--r--  1 ragge  wheel  10270 Oct  6 19:57 pass2.h
-rw-r--r--  1 ragge  wheel  25770 Sep 17 09:58 reader.c
-rw-r--r--  1 ragge  wheel  36859 Oct  6 22:50 regs.c
```

- CPU-specific

```
-rw-r--r--  1 ragge  wheel  18825 Sep  8 21:19 local2.c
-rw-r--r--  1 ragge  wheel   7847 Sep 17 09:58 order.c
-rw-r--r--  1 ragge  wheel  24420 Oct  6 22:50 table.c
```

Code-generator CPU-specific code

- About 30 functions in total
- 18 functions are related to instruction emission.
- The table which is an array of optab entries which each describes an instruction.
- The offstar()/ormake() functions are among the most difficult to write. They searches for situations where indexing of instructions can be used.

Instruction table

- The table is an array of entries that is the basis for instruction selection.

```
{ PLUS,        INAREG|FOREFF,
       SAREG,            TINT|TUNSIGNED,
       SAREG|SNAME|SOREG,  TINT|TUNSIGNED,
          0,  RLEFT,
          "   addl AR,AL\n", },
{ OPSIMP,         INAREG,
       SAREG,             TCHAR|TUCHAR,
       SCON,           TANY,
          0,  RLEFT,
          "   Ob CR,AL\n", },
```

Instruction table

- Macro ops in table
  - Z – special machine dependent operations
  - F – this line deleted if FOREFF is active
  - S – field size
  - H – field shift
  - M – field mask
  - N – complement of field mask
  - L – output special label field
  - O – opcode string
  - B – byte offset in word
  - C – for constant value only
  - I – in instruction
  - A – address of
  - U – for upper half of address, only

Future directions

- f77 frontend;
  - The original f77 compiler that were targeted towards the Johnson and Richie compilers were quite simple to get running.
- C++ frontend;
  - Despite what people say I think it won't be so difficult to write one :-)
- as, ld, ...
  - Original code exists, just spend some time...

Nice books and papers

- A tour through the portable C compiler
  - S. C. Johnson 1978
- Iterated Register Coalescing
  - ACM paper, Appel & George 1996
- Compilers: Principles, Techniques, and Tools
  - ”Dragon book”, Ravhi, Sethi, Ullman, ...
- Modern compiler implementation in C/Java
  - Appel, ...

Related stuff

- The pcc web site; http://pcc.ludd.ltu.se
- Mailing lists;
  - pcc-list@ludd.ltu.se
  - pcc-commit-list@ludd.ltu.se

Funding? Yes please! :-)