NAC: A lightweight IR for ASIP compilers

Nikolaos Kavvadias and K. Masselos
{nkavv,kmas}@uop.gr

Department of Computer Science and Technology,
University of Peloponnese,
Tripoli, Greece

18–21 July 2011
Introduction and motivation

- Heterogeneous platform design using ASIPs (Application-Specific Instruction-set Processors) involves compiler-based design space exploration
- Compiler IR (intermediate representation) serves dual purpose: program representation and abstract machine
- Free/open-source/academic compilers:
  - GCC [1]: GIMPLE IR, interesting, yet unstable
  - LLVM [2]: register-based IR, targeted by clang frontend
  - COINS: S-expression IR
  - Machine-SUIF: influenced Microsoft Phoenix
  - All these approaches involve excessive and complex infrastructures
- Commercial ASIP compilers:
  - Synopsys Processor Designer: uses ACE CoSy CCMIR
  - IP (ASIP) Designer from Target Compiler Technologies
  - Tensilica XCC compiler
  - Complex and inaccessible IRs
We propose NAC (N-Address Code), an open, extensible, typed-assembly language

- Extensible typed assembly language similar in concept to GCC’s GIMPLE and LLVM but with unique features
- Arbitrary $m$-to-$n$ mappings
- Very light/unbiased semantics: a single operation construct
- Bit-accurate data types (integer, fixed-point arithmetic)
- Scalar, single-dimensional array and streamed I/O procedure arguments
- Uses: RISC-like VM for static/dynamic analyses, CDFG extraction, reachability-based data flow analyses, input to HLS kernels, software compilation
- Minimal SSA (scan-based) algorithms [3, 4]
  - No sophisticated concepts/data structures, no computation of the iterated dominance frontier
  - Suitable for rapid prototyping compilers
NAC EBNF grammar

nac_top = {gvar_def} {proc_def}.
gvar_def = "globalvar" anum decl_item_list ";".
proc_def = "procedure" [anum] "(" [arg_list] ")"
        "{" [{lvar_decl}] [{stmt}] "}".
stmt = nac | pcall | id ":".
nac = [id_list "="] anum [id_list] ";".
pcall = ["(" id_list ")" "="] anum ["(" id_list ")"] ";".
id_list = id {""," id}.
decl_item_list = decl_item {"," decl_item}.
decl_item = (anum | uninitarr | initarr).
arg_list = arg_decl {""," arg_decl}.
arg_decl = ("in" | "out") anum (anum | uninitarr).
lvar_decl = "localvar" anum decl_item_list ";".
initarr = anum [" id "] "=" "{ numer "," numer } "}".
uninitarr = anum [" [id] "]".
anum = (letter | ")") {letter | digit}.
id = anum | (["] (integer | fxpnum)).
Example translation flow: ANSI C ⇒ NAC ⇒ Graphviz CDFG

**ANSI C**

```c
void popcount(uint inp, uint *outp)
{
    uint data, count, temp;
    data = inp;
    count = 0;
    while (data != 0)
    {
        count = count + (data & 0x1);
        data = data >> 0x1;
    }
    *outp = count;
}
```

**NAC IR**

```nac
procedure popcount (in u32 inp, out u32 outp)
{
    localvar u32 data, count, temp;
    BB1:
    data <= mov inp;
    count <= ldc 0;
    BB2 <= jmpun;
    BB2:
    temp <= and data, 1;
    count <= add count, temp;
    data <= shr data, 1;
    BB3, BB2 <= jmpeq data, 0;
    BB3:
    outp <= mov count;
}
```

NAC: A lightweight IR for ASIP compilers

Nikolaos Kavvadias and K. Masselos {nkavv,kmas}@uop.gr
IR transformations

- Motivation: single- (crcsp) and double-point (crcdp) genetic crossover operators
- Conventional mapping to IR produces slow code/implementation
- New IR nodes: bitins (bitfield insertion), bitext (extraction), concat (subword concatenation)
- IR extension through graph transformation, implementation on ByoRISC ASIP [5]

Post-transformation IR

<table>
<thead>
<tr>
<th>GA oper.</th>
<th>Bit-level oper.</th>
<th>Ni/No</th>
<th>Cycles (seq.)</th>
<th>CI cyc.</th>
<th>CI area (MAU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>crcsp</td>
<td>No/Yes</td>
<td>4/1</td>
<td>76-13</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>crcsp</td>
<td>No/Yes</td>
<td>8/1</td>
<td>41-6</td>
<td>3-1</td>
<td>0.977-0.142</td>
</tr>
<tr>
<td>crcsp</td>
<td>No/Yes</td>
<td>8/2</td>
<td>5-1</td>
<td>3-1</td>
<td>1.867-0.153</td>
</tr>
<tr>
<td>crcdp</td>
<td>No/Yes</td>
<td>4/1</td>
<td>111-18</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>crcdp</td>
<td>No/Yes</td>
<td>8/1</td>
<td>58-8</td>
<td>3-1</td>
<td>1.466-0.147</td>
</tr>
<tr>
<td>crcdp</td>
<td>No/Yes</td>
<td>8/2</td>
<td>5-1</td>
<td>3-1</td>
<td>2.800-0.164</td>
</tr>
</tbody>
</table>

Results on Virtex-4

Nikolaos Kavvadias and K. Masselos {nkavv, kmas}@uop.gr

NAC: A lightweight IR for ASIP compilers
NAC virtual machine profiling

- NACVM: the abstract machine inferred by a basic NAC instruction set
- Examples of static/dynamic profiling using C backend
  - *atsort* (all topological sorts), *coins* (compute change), multimode *cordic* (DSP algorithm), *easter* (Easter date calculations), *fixsqrt* (fixed-point sqrt), *perfect* (perfect number detection), *sieve* (prime sieve), *xorshift* (100 PRNG calls)
- Shown: lines (NAC, CDFG), num. of CDFGs (*P*), nodes (*N*), and edges (*E*), φ statements, dynamic non-SSA instructions

<table>
<thead>
<tr>
<th>Application</th>
<th>LOC (NAC)</th>
<th>LOC (dot)</th>
<th>P</th>
<th>V</th>
<th>E</th>
<th>#φs</th>
<th>#Instructions.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>atsort</em></td>
<td>155</td>
<td>484</td>
<td>2</td>
<td>136</td>
<td>336</td>
<td>10</td>
<td>6907</td>
</tr>
<tr>
<td><em>coins</em></td>
<td>105</td>
<td>509</td>
<td>2</td>
<td>121</td>
<td>376</td>
<td>10</td>
<td>405726</td>
</tr>
<tr>
<td><em>cordic</em></td>
<td>56</td>
<td>178</td>
<td>1</td>
<td>57</td>
<td>115</td>
<td>7</td>
<td>256335</td>
</tr>
<tr>
<td><em>easter</em></td>
<td>47</td>
<td>111</td>
<td>1</td>
<td>46</td>
<td>59</td>
<td>2</td>
<td>3082</td>
</tr>
<tr>
<td><em>fixsqrt</em></td>
<td>32</td>
<td>87</td>
<td>1</td>
<td>29</td>
<td>52</td>
<td>6</td>
<td>833900</td>
</tr>
<tr>
<td><em>perfect</em></td>
<td>31</td>
<td>65</td>
<td>1</td>
<td>23</td>
<td>36</td>
<td>4</td>
<td>6590739</td>
</tr>
<tr>
<td><em>sieve</em></td>
<td>82</td>
<td>199</td>
<td>2</td>
<td>64</td>
<td>123</td>
<td>12</td>
<td>515687</td>
</tr>
<tr>
<td><em>xorshift</em></td>
<td>26</td>
<td>80</td>
<td>1</td>
<td>29</td>
<td>45</td>
<td>0</td>
<td>2000</td>
</tr>
</tbody>
</table>

Nikolaos Kavvadias and K. Masselos  {nkavv,kmas}@uop.gr

NAC: A lightweight IR for ASIP compilers
Variable numbering with algorithm $P$

```plaintext
VariableNumbering(List NACs, List vars):
    ssa_vars = empty; var_reads = zeroes;
    var_writes = ones; set_writes = 0;
    curr_bb = 0; prev_bb = -1;
    bbnum = get number of basic blocks from NACs;
    for stmt in NACs do
        // Update curr_bb, prev_bb, var_reads, var_writes.
        if stmt.bb != curr_bb then
            prev_bb = curr_bb; curr_bb = stmt.bbix;
            if curr_bb > 1 and set_writes == 0 then
                var_writes = bbnum; set_writes = 1;
            var_reads = curr_bb;
        for input operand (opnd) in stmt do
            if opnd is a localvar and is scalar then
                // Create a numbered version of opnd.
                ssaopnd = opnd ## var_reads['opnd'];
                update input operands of stmt;
        for output operand (opnd) in stmt do
            get opnd_ix = index of opnd in vars;
            if opnd is a localvar and is scalar then
                // Create numbered ver. of opnd, then copy it.
                if stmt.bb > 1 then
                    var_writes['opnd'] += 1;
                    var_reads['opnd'] = var_writes['opnd'];
                    ssaopnd = opnd ## var_writes['opnd'];
                    insert ssaopnd to ssa_vars list;
                    update output operands of stmt;
            update stmt in NACs;
        delete localvar scalars from vars;
    merge ssa_vars with vars;
```

Nikolaos Kavvadias and K. Masselos {nkavv,kmas}@uop.gr
NAC: A lightweight IR for ASIP compilers
Example: Variable numbering with $P$ and $H$

- **Algorithm $P$** [3]
  - every variable is split at BB boundaries
  - $\phi$-functions are placed for each variable in each BB
  - preassigns variable versions

- **Algorithm $H$** [4]
  - doesn’t predetermine variable versions at control-flow joins
  - different $\phi$-insertion as well, same $\phi$-minimization/DCE to $P$

---

Nikolaos Kavvadias and K. Masselos  {nkavv,kmas}@uop.gr

NAC: A lightweight IR for ASIP compilers
$\phi$-insertion with algorithm $P$

\begin{verbatim}
PhiInsertion(List NACs, List vars, List labels, List nonssa_vars):
    phi_stmts = empty; bb_preds = zeroes; bb_preds_num = 0;
    (ST, G) = create CFG from (NACs, labels);
    for k in BBs(ST) do
        insert predecessor BBs of k in bb_preds;
        bb_preds_num = get number of predecessor BBs of k;
        for sopnd in nonssa_vars do
            if sopnd is localvar scalar, has def/use in k then
                phi_opnds_in = empty; phi_opnds_out = empty;
                // Create the phi stmt dest operand.
                if bb_preds_num > 1 then
                    ssaopnd_out = sopnd ## k+1;
                    insert ssaopnd_out to phi_opnds_out, vars;
                ix = 0;
                for n in bb_preds_num do
                    if bb_preds[n] != -1 then
                        ix = SSA ver of sopnd at last def in BB #n;
                    if ix == 0 then ix = bb_preds[n] + 1;
                // Create the phi stmt source operand.
                ssaopnd_in = sopnd ## ix;
                insert ssaopnd_in to phi_opnds_in;
                // Create the phi statement.
                if k == 0 and BB #k does not define sopnd then
                    phi_stmt = LOADCONST(phi_opnds_out);
                elsif BB #k has predecessors then
                    phi_stmt = PHI(phi_opnds_out, phi_opnds_in);
                insert phi_stmt to phi_stmts;
    merge NACs with phi_stmts;
    update absolute addresses (addr) in NACs, labels;
\end{verbatim}
Example: $\phi$-insertion with $P$ and $H$

$\phi$-insertion with $P$

$\phi$-insertion with $H$
Conclusions and references

- An extensible, typed-assembly intermediate language (NAC) was presented.
- Its applicability was illustrated through cases of IR extension, profiling and compilation pass development (SSA construction).
- Descriptions of minimal SSA construction algorithms with elementary data structures shown for the first time.


