Automatic Porting of BFD Library

International Workshop on
System-on-Chip for Real-Time Applications

*Maghsoud Abbaspour, Jianwen Zhu*

Electrical and Computer Engineering
University of Toronto
July, 2002

jzhu@eecg.toronto.edu
http://www.eecg.toronto.edu/~jzhu
Outline

- **Background**
- Architecture Modeling and Description
- Implementation and Results
- Conclusions
Problem Definition

- System-On-Chip dominated by software
- Is software really cheap?
- Yes if development tools readily available
- Software development suite
  - Compiler
  - Backend tools: assembler/linker etc.
  - Simulator
  - Debugger
Wasn’t it a Solved Problem in 1960s?

- Considered trivial except perhaps a few black magic tricks
  - Virtually no paper published on linker/assembler construction
  - No treatment in CS/CE curriculum except lab instructions
- Not true anymore for modern computer system
- Linker example
  - Driven by OS: shared library/dynamic linking
  - Driven by Language (C++): constructor/destructor, template, inline
  - Driven by compiler technology: whole program analysis/optimization
Isn’t It Just Engineering?

Why can’t I manually port the “portable” GNU tools?

Why can’t I hack legacy tool from Company XYZ?
  - GNU’s binutils has 250K lines of code!
  - John Levine: Backend gurus can be “packed in one single room”.
  - Target-dependent interface not cleanly defined
  - Target-dependent implementations scattered in many C files
  - Target-dependent information overlaps with what needed for other tools like compiler/simulator/debuggers.

Need “automatic hacking”
Wasn’t it a Solved Problem in 1990s?

- Yes! Related work:
  - Code generator generators: IBURG etc
  - Compiler community: Zephyr, MDES
  - CAD community: MIMOLA, CHESS, EXPRESSION, FLEXWARE, LISA, ISDL, MESCAL
  - Commercial ARM, ARC, Tensilica etc

- An No! Our contribution:
  - Formal model of ISA: independent of ADL
  - Application Binary Interface (ABI)
  - Open-source, production quality package
Background: Anatomy of Object Files

- Three types:
  - Relocatable object files
  - Executable files
  - Shared object files

- Many legacy formats
  - a.out
  - Common Object File Format (COFF)
  - Portable Executable (PT)
  - (ELF)

<table>
<thead>
<tr>
<th>Linking view</th>
<th>Execution view</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF header</td>
<td>ELF header</td>
</tr>
<tr>
<td>Program header table (optional)</td>
<td>Program header table</td>
</tr>
<tr>
<td>Section 1</td>
<td>Segment 1</td>
</tr>
<tr>
<td>...</td>
<td>Segment 2</td>
</tr>
<tr>
<td>Section n</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Section header table</td>
<td>Section header table (optional)</td>
</tr>
</tbody>
</table>
Background: Anatomy of GNU’s binutils Package

- Binary File Descriptor (BFD)
  Library: libbfd
  - Relocatable object files
  - Executable files
  - Shared object files
- opcodes Library
- gas
- Target-independent components
Outline

- Background
- *Architecture Modeling and Description*
- Implementation and Results
- Conclusions
Definitions

- Architecture modeling:
  Defining an abstraction (model) of the target-dependent information as well as their relationship
  - Instruction Set Architecture (ISA)
  - Application Binary Interface (ABI)
  - Micro-architecture

- Architecture Description Language (ADL)
  A language that can help capture the architecture model in a human-friendly form.

- Architecture description
Babel Specification Language

- A general-purpose IP specification language
  - Frontend to the IPSUITE infrastructure
  - Captures arbitrary graph of data

- Help capture multiple “facets” of an IP
  - Behavioral facet
  - Architectural: ISA, ABI, ILP facets
  - Physical: Plan, Mask facets
Babel Specification Language Architecture

- Extensible: type system to define domain-specific data model
- Specification are checked using type inference engine
- Domain-specific plug-ins to compile spec into IP
Specifying Behavior

■ The need
  ■ Instruction selection
  ■ Simulator generation
  ■ Dynamic linking
  ■ Calling convention specification

■ The form
  ■ Field: logical registers
  ■ Method: logical instructions

■ SPARC behavioral specification

```cpp
facet Sparc::beh {
  unsigned[32] g0, g1, g2, g3, g4, g5, g6, g7;
  unsigned[32] l0, l1, l2, l3, l4, l5, l6, l7;
  unsigned[32] i0, i1, i2, i3, i4, i5, i6, i7;
  unsigned[32] fp, sp;

  ...byte lodb( byte* src1, int offset ) {
    return *(src1+offset);
  }

  short lods( short* src1, int offset ) {
    return *(src1+offset);
  }

  ...
}
```
Specifying ISA and ABI

Data Model:

```java
class domain.ISA {
    {}Store stores;
    {}CellGroup groups;
    {}Field fields;
    {}Format formats;
    {}Instrn instrns;
    {}string properties;
}
```

```java
class domain.ABI {
    // ... register convention
    // ... argument passing scheme
    // ... frame layout
    // ... view change
    {}Reloc relocs;
    method plt;
}
```

SPARC ISA and ABI facets

```java
facet Sparc::isa = new domain.ISA {
    stores = { ... }
    fields = { ... }
    formats = { ... }
    instrns = { ... }
    properties = { ... }
}
```

```java
facet Sparc::abi = new domain.ABI {
    ...
    relocs = { ... }
    ...
}
```
Specifying Storages

- Size and granularity
- Register window
- Mapping to logical registers
- Data modeling

```c
typedef int *int Cell;
typedef []field CellGroup;

class Store {
    Store( int gran, int size );

    int depth;
    int overlap;
    field pointer;
    {}CellGroup cells;
}
```

```
SPARC storage specification

stores = {
    sGPR = new Store( 32, 32 ) {
        depth = 128;
        overlap = 24;
        pointer = cwp;
        maps = {
            gpr = [
                g0, g1, g2, g3, g4, g5, g6, g7,
                o0, o1, o2, o3, o4, o5, o6, o7,
                10, 11, 12, 13, 14, 15, 16, 17,
                10, 11, 12, 13, 14, 15, 16, 17
            ]
            alias = [
                x, x, x, x, x, x, x, x,
                x, x, x, x, x, x, x, x,
                x, x, x, x, x, x, x, x,
                x, x, x, x, x, x, fp, x
            ]
        }
    }
}
...
Specifying Instruction Format

- Instruction format:
  - Opcode field
  - Register field
  - Immediate/Relocatable field

- Data model

```java
class Field {
    Field( int size ); // opcode field
    Field( // immediate field
        int size, boolean isSigned, int argn0
    );
    Field( // register field
        int size, boolean isDest,
        int argn0, CellGroup group
    );
    Field( // relocatable field
        int size, boolean isSigned,
        int argn0, int relotype
    );
}
typedef [ ]Field Format;
```

- SPARC instruction format

```java
fields = {
    op = new Field( 2 );
    op2 = new Field( 3 );
    rdg = new Field( 5, true, 0, gpr );
    rdf = new Field( 5, true, 0, fpr );
    rdd = new Field( 5, true, 0, dpr );
    rdfsr = new Field( 5, true, 0, fsr );
    a = new Field( 1 );
    cond = new Field( 4 );
    imm22 = new Field( 22, false, 1, 9 );
    disp22 = new Field( 22, true, 1, 8 );
    ...
}

formats = {
    F3A = [op,rdg,op3,rs1g,i,asi,rs2g],
    F3B = [op,rdg,op3,rs1g,i,simm13],
    F3AF = [op,rdf,op3,rs1g,i,unused8,rs2g],
    F3BF = [op,rdf,op3,rs1g,i,simm13],
    F3AD = [op,rdd,op3,rs1g,i,unused8,rs2g],
    F3BD = [op,rdd,op3,rs1g,i,simm13],
    F3AFSR = [op,rdfsr,op3,rs1g,i,unused8,rs2g],
    F3BFSR = [op,rdfsr,op3,rs1g,i,simm13],
    ...
}
```
Specifying Instructions

- Assembly format
  - Instruction emitting
  - Instruction assembling
  - Instruction de-assembling

- Binary format

- Opcodes

- Patterns

```java
class Instrn {
  string   asmFormat;
  Format   binFormat;
  []int     opcodes;
  {}method  patterns;
}
```

- Sparc instructions

```java
instrns = {
  typedef int[0:(1<<13-1)] int13;

  ldsb = new Instrn {
    asmFormat = "ldsb [%1],%0; [%1+%2],%0":
    asmFormat = F3A;
    opcode = [0x3, 0x9, 0x0, 0x0];
    patterns = {
      byte.OP_LOD(pointer),
      Sparc.lodb( point, int )
    }
  }

  andi = new Instrn {
    asmFormat = "and #1,%2,%0; %2,#1,%0";
    binFormat = F3B;
    opcode = [0x2, 0x1, 0x1];
    patterns = {
      uint.OP_BAND(uint, uint13)
    }
  }

  ...
}
```
Specifying Relocations

- Relocation type: expression applied on information kept in
  - Relocation field
  - Relocation entry
- Linkers: base, GOT, GOT offset, PLT offset, place of instruction

Expression encoding

<table>
<thead>
<tr>
<th>-GOT</th>
<th>+A</th>
<th>+B</th>
<th>+G</th>
<th>+GOT</th>
<th>+L</th>
<th>-P</th>
<th>+S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

= 0x43

=> S+A−P

Sparc instructions

```java
enum ABIComplainKind {
    KIND_ABI_IGNORE = 0,
    KIND_ABI_BIT = 1,
    KIND_ABI_SIGN = 2,
    KIND_ABI_UNSIGNED = 3
}

class Reloc {
    Reloc(
        int id, int exprCode,
        int rightShift, int bitSize,
        int bitPos, int complain
    );
}

relocs = {
    r_sparc_none = new Reloc(
        0, 0x00, 0, 0, 0, KIND_ABI_IGNORE );
    r_sparc_8 = new Reloc(
        1, 0x41, 0, 8, 0, KIND_ABI_BIT );
    r_sparc_wdisp30 = new Reloc(
        7, 0x43, 2, 30, 0, KIND_ABI_SIGN );
    ...
}
```
Outline

- Background
- Architecture Modeling and Description
- *Implementation and Results*
- Conclusions
Automatic Porting of Binary Utilities

- Bridge the gap between
  - Our architecture model
  - GNU’s porting interface
- Template files
  - C, header, configuration, Makefile
  - Placeholder for substitution
- No change to GNU binutils distribution

- Retargeting binutils
More on Implementation

- Retargeting BFD library
  - Type definitions
  - Data generation: internal representation of relocation method and PLT
  - Function generation: relocation checking, relocator, dynamic section generators

- Retargeting opcodes library
  - Internal representation of instructions
  - Instructions in hash table keyed by mnemonics
  - Instruction disassembling

- Retargeting gas
  - Generates line parser
  - Mnemonic matching, pattern matching
  - Symbol, immediate, register field collection
  - Instruction encoding
  - Relocation generation

- Retargeting other tools
  - Depends only on BFD and opcodes library to generate line parser
  - Only configuration changes
Implementation Statistics

■ Our Tools

<table>
<thead>
<tr>
<th>Component</th>
<th>#lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babel compiler</td>
<td>3538</td>
</tr>
<tr>
<td>ISA plug-in</td>
<td>2124</td>
</tr>
<tr>
<td>ABI plug-in</td>
<td>1474</td>
</tr>
<tr>
<td>rbinutils</td>
<td>2785</td>
</tr>
<tr>
<td>ISA/ABI data model</td>
<td>207</td>
</tr>
</tbody>
</table>

■ Processor specification

<table>
<thead>
<tr>
<th>Processor</th>
<th>Beh (#lines)</th>
<th>ISA (#lines)</th>
<th>ABI (#lines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC</td>
<td>196</td>
<td>2300</td>
<td>56</td>
</tr>
<tr>
<td>SimpleScalar</td>
<td>90</td>
<td>1048</td>
<td>45</td>
</tr>
<tr>
<td>Alpha</td>
<td>1144</td>
<td>4511</td>
<td>52</td>
</tr>
<tr>
<td>I386</td>
<td>1234</td>
<td>20406</td>
<td>44</td>
</tr>
</tbody>
</table>
Result: Retargeting SPARC

- **Generated BFD code**
  
<table>
<thead>
<tr>
<th>File</th>
<th>#line</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfd/elf32-sparc.c</td>
<td>1899</td>
</tr>
<tr>
<td>include/elf/sparc.h</td>
<td>114</td>
</tr>
<tr>
<td>include/elf/common.h</td>
<td>641</td>
</tr>
<tr>
<td>bfd/archures.c</td>
<td>1088</td>
</tr>
<tr>
<td>bfd/config.bfd</td>
<td>1093</td>
</tr>
<tr>
<td>configure.in</td>
<td>838</td>
</tr>
<tr>
<td>bfd/cpu-sparc.c</td>
<td>78</td>
</tr>
<tr>
<td>bfd/targets.c</td>
<td>1243</td>
</tr>
<tr>
<td>bfd/bfd-in2.h</td>
<td>3918</td>
</tr>
<tr>
<td>config.sub</td>
<td>1449</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>12361</strong></td>
</tr>
</tbody>
</table>

- **Generated opcodes code**

<table>
<thead>
<tr>
<th>File</th>
<th>#line</th>
</tr>
</thead>
<tbody>
<tr>
<td>include/opcode/sparc.h</td>
<td>116</td>
</tr>
<tr>
<td>opcodes/sparc-opc.c</td>
<td>2128</td>
</tr>
<tr>
<td>opcodes/Makefile.am</td>
<td>658</td>
</tr>
<tr>
<td>opcodes/configure-in</td>
<td>283</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3185</strong></td>
</tr>
</tbody>
</table>

- **Generated gas code**

<table>
<thead>
<tr>
<th>File</th>
<th>#line</th>
</tr>
</thead>
<tbody>
<tr>
<td>gas/config/sparc-tmp.h</td>
<td>521</td>
</tr>
<tr>
<td>gas/config/tc-sparc.c</td>
<td>2436</td>
</tr>
<tr>
<td>gas/config/tc-sparc.h</td>
<td>205</td>
</tr>
<tr>
<td>gas/configure.in</td>
<td>984</td>
</tr>
<tr>
<td>gas/Makefile.am</td>
<td>2280</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>6424</strong></td>
</tr>
</tbody>
</table>

- **Generated ld code**

<table>
<thead>
<tr>
<th>File</th>
<th>#line</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld/Makefile.am</td>
<td>985</td>
</tr>
<tr>
<td>ld/emulparams/elf32-sparc.sh</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>997</strong></td>
</tr>
</tbody>
</table>

- **Verification: SPEC2000**
Conclusion

- ISA model should be made formal
- ABI model should not be ignored
- Use typed, generic language as ADL
- Leverage open-source, production quality software

Future work
- Exploit polymorphism in specification to reduce specification size
- Use behavioral specification for relocation calculation
- Automatic simulator generation
- Automatic porting of GDB