Multi-Platform Auto-Vectorization

Dorit Nuzman, IBM
Richard Henderson, RedHat
Multi-Platform Auto-Vectorization - Talk Layout

- Vectorization for SIMD
  - Alignment Example

- Vectorization in GCC

- Vector Abstractions
  - Abstractions for Alignment

- Multi-platform Evaluation

- Related Work & Conclusion
Vectorization

✧ SIMD (Single Instruction Multiple Data) model
  ✧ Communications, Video, Gaming
  ✧ MMX/SSE, Altivec

✧ Programming for Vector Platforms
  ✧ Fortran90
    \[ a[0:N] = b[0:N] + c[0:N]; \]
  ✧ Intrinsics
    \[
    \begin{align*}
    &\text{vector float } vb = \text{vec}\_load (0, \text{ptr}\_b); \\
    &\text{vector float } vc = \text{vec}\_load (0, \text{ptr}\_c); \\
    &\text{vector float } va = \text{vec}\_add (vb, vc); \\
    &\text{vec}\_store (va, 0, \text{ptr}\_a);
    \end{align*}
    \]

✧ Autovectorization: Automatically transform serial code to vector code by the compiler.
What is vectorization

- Data elements packed into vectors
- Vector length → Vectorization Factor (VF)
- No Data Dependences
- SIMD Architectural Capabilities

VF = 4

<table>
<thead>
<tr>
<th>VR1</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VR5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vector Registers

OP(a)
OP(b) → VOP(VR1)
OP(c)
OP(d)

Data in Memory:

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
Limitations of SIMD Architectures:
Unaligned memory access

Vector Registers

Data in Memory:

OP(c) → VOP(c, d, e, f)
OP(d)
OP(e)
OP(f)

V3 ← vec-shift-left v1, v2, 2
V3 ← vec-permute v1, v2, {2,3,4,5}

V1 ← extql (v1, addr), v2 ← extqh (v2, addr) (alpha)
V3 ← vec-or (v1, v2)

V3 ← alvn.ps v1,v2,addr (MIPS64)
V3 ← load-left, load-right (MIPS MDMX)
Multi-Platform Auto-Vectorization - Talk Layout

- Vectorization for SIMD
  - Alignment Example

- Vectorization in GCC

- Vector Abstractions
  - Abstractions for Alignment

- Multi-platform Evaluation

- Related Work & Conclusion
GCC

- Free Software Foundation
- Multi-platform

BladeCenter JS20
GCC

- Free Software Foundation
- Multi-platform
- Who’s involved
  - Volunteers
  - Linux distributors (RedHat, Suse…)
  - Code Sourcery, AdaCore…
  - IBM, HP, Intel, Apple…

Linux on Power

Open, powerful and affordable, a key to innovation
IBM Labs in Haifa

GCC Passes

- Ada front-end
- Fortran front-end
- C++ front-end
- C front-end
- Parse trees

GIMPLE Abstractions

- Middle-end GIMPLE trees

SSA optimizations

- CCP
- PRE
- DCE
- CSE
- DSE
- Loop opts
- Forward prop
- Copy prop
- VRP

Loop optimizations

- Invariant motion
- Unswitching
- Linear transform
- If-conversion
- Vectorization
- Unrolling

Sibling call optimizations

- Common subexpression elimination
- Loop optimizations
- Data flow analysis
- Instruction combination
- Instruction scheduling
- Register allocation and reloading
- Instruction scheduling (repeated)
- Branch shortening

Vector Size

- Mips port
- i386 port
- Rs6000 port
- Machine description

Assembly

RTL
Multi-Platform Auto-Vectorization - Talk Layout

- Vectorization for SIMD
  - Alignment Example

- Vectorization in GCC

- Vector Abstractions
  - Abstractions for Alignment

- Multi-platform Evaluation

- Related Work & Conclusion
Vector Abstractions: Why Needed

- **Represent high-level idioms** that otherwise can’t be vectorized
  - reduction
  - special idioms (sad, subtract-and-saturate, dot-product)

- **Express vector operations in GIMPLE**
  - “reduc-plus”
  - extract, shuffle,…

- **API for targets** to convey availability and cost of a functionality

<table>
<thead>
<tr>
<th>optab/type</th>
<th>char</th>
<th>short</th>
<th>int</th>
<th>v16char</th>
<th>v8short</th>
<th>v4int</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>f1</td>
<td>f2</td>
<td>f3</td>
<td>f4</td>
<td>f5</td>
<td>f6</td>
</tr>
</tbody>
</table>

```c
s = 0;
for (i=0; i<N; i++) {
    s = s + a[i] * b[i];
}
```
Vector Abstractions: Considerations

- **Generality vs. applicability**
  - General enough to cover all uses
  - Minimize increase of operation-codes
  - Not generally supported

- **Compound vs. building blocks**
  - Increase of operation-codes
  - Complicated “black-box” operations
  - Increase ways to represent same functionality
  - Improved direct support of a high-level idiom over basic functionalities

- **GCC conventions**
  - naming, existing-operation-codes, default values…

- **Performance**
  - Translates to most efficient code
Vector Abstractions: Abstractions for alignment

- **Implicit Realignment**
  - `misaligned_ref (ptr, mis)`

- **Explicit Realignment**
  - `aligned_ref (ptr)`
  - `realign_load (v1, v2, RT)`
  - `Realignment Token (RT)`

Data in Memory:

```
0  1  2  3  4  5  6  7
a b c d  e f g h
V1  V2
```

```
V3 ← movdqu
V3 ← load-left, load-right
```

(MMX/SSE)

```
V3 ← vec-shift-left v1, v2, 2
```

(Altivec)

```
V3 ← vec-permute v1, v2, \{2,3,4,5\}
```

(Alpha)

```
V1 ← extql (v1, addr), v2 ← extqh (v2, addr)
V3 ← vec-or (v1, v2)
```

(MIPS MDMX)

```
V3 ← alvn.ps v1,v2,addr
```

(MIPS64)
Handling Alignment

```c
for (i=0; i<N; i++){
    x = a[i];
    b[i] = x;
}
```

```c
addra_0 = &a[0];
addrb = &b[0];
vector vx; vx1, vx2;

addra_i = addra_0;
LOOP:
    vx1 = misaligned_ref(addra_i, 0);
    vx2 = align_ref(addra_i+15);
    vx = realign_load(vx1, vx2, addra_i);
    indirect_ref(addrb) = vx;
    addra_i += 16; addrb += 16;
```

```c
addra_0 = &a[0];
addrb = &b[0];
vector vx, vx1, vx2;

vx1 = align_ref(addra_0);
addra_i = addra_0 + 15;
LOOP:
    vx2 = align_ref(addra_i);
    vx = realign_load(vx1, vx2, addra_i);
    indirect_ref(addrb) = vx;
    addra_i += 16; addrb += 16;
```
Handling Alignment

```c
for (i=0; i<N; i++){
    x = a[i];
    b[i] = x;
}
```

```c
addra_0 = &a[0];
addrb = &b[0];
vector vx;
addra_i = addra_0;
LOOP:
vx = misaligned_ref (addra_i,0);
indirect_ref (addrb) = vx;
addra_i += 16; addrb += 16;

vx1 = align_ref (addra_0);
RT = target_get_RT (addra_0);
addra_i = addra_0 + 15;
LOOP:
vx2 = align_ref (addra_i);
vx = realign_load (vx1, vx2, RT);
indirect_ref (addrb) = vx;
addra_i += 16; addrb += 16; vx1 = vx2;
```
GIMPLE Vector Abstractions

- **Alignment:**
  - misaligned_ref, align_ref
  - realign_load, target_get_RT

- **Reduction:**
  - reduc_plus

- **Special patterns:**
  - dot_prod, sad
  - sub_sat
  - widen_mult, widen_sum

- **Conditional operations:**
  - (cond) ? x : y

- **Type Conversions**
  - unpack_high, unpack_low
  - pack_mod, pack_sat

- **Strided-Accesses:**
  - extract_odd, extract_even
  - interleave_high, interleave_low
Multi-Platform Auto-Vectorization - Talk Layout

- Vectorization for SIMD
  - Alignment Example

- Vectorization in GCC

- Vector Abstractions
  - Abstractions for Alignment

- Multi-platform Evaluation

- Related Work & Conclusion
Multi-Platform Evaluation

- IBM PowerPC970, Altivec (VS = 16)
- Intel Pentium4, SSE2 (VS = 16)
- AMD Athlon64, SSE2 (VS = 16)
- Intel Itanium2 (VS = 8)
- MIPS64, paired-single-fp (VS = 8)
- Alpha (VS = 8)
Multi-Platform Evaluation

Vectorization Speedup Factors - Aligned

- powerpc
- pentium4
- athlon64
- itanium2
- alpha
- mips64

- blas.sdot_fp
- blas.saxpy_fp
- blas.dscal_fp
- vecmax_fp
- checksum_s16
- chromakey_u16
- vecmax_s16
- vecsum_u8
- chromakey_u8
- vecmax_u8
Multi-Platform Evaluation

Vectorization Speedup Factors - Unaligned

- powerpc970
- pentium4
- athlon64
- itanium2
- alpha
- mips64
Related Work

- Vectorizing compilers available for a specific architecture
  - XL (Eichenberger, Wu). Altivec
  - icc (Bik). MMX/SSE
  - CoSy (Krall). VIS
  - SUIF (Larsen, Amarasinghe; Shin, Chame, Hall) – Altivec

- Vectorizing compilers available for multiple SIMD targets
  - source-to-source compilers
    - Vienna MAP, 2-way, domain-specific patterns. BG +
    - SWARP. source-to-source, multimedia patterns. Trimedia +

- This Work:
  - In a robust industrial-strength compiler
  - Experimental results on several different SIMD platforms
Concluding Remarks

✧ SIMD
  ✧ Hardware limitations
  ✧ Unique Hardware mechanisms
  ✧ Diverse nature

✧ Multi-platform vectorizer
  ✧ Bridge gap across different SIMD targets
  ✧ Efficiently support each individual platform
  ✧ Identify proper abstractions

✧ Developing the vectorizer in the GCC platform
  ✧ Collaborative investment of different vendors/developers
  ✧ Open, available
The End