Auto-vectorization in GCC

Dorit Naishlos

dorit@il.ibm.com
Vectorization in GCC - Talk Layout

Background: GCC
HRL and GCC

Vectorization
Background
The GCC Vectorizer
Developing a vectorizer in GCC
Status & Results
Future Work

Working with an Open Source Community

Concluding Remarks
GCC – GNU Compiler Collection

Open Source
Download from gcc.gnu.org

Multi-platform

2.1 million lines of code, 15 years of development

How does it work

cvs
mailing list: gcc-patches@gcc.gnu.org
steering committee, maintainers

Who’s involved
Volunteers
Linux distributors
Apple, IBM – HRL (Haifa Research Lab)
IBM Labs in Haifa

GCC Passes

C front-end
C++ front-end
Java front-end

parse trees

middle-end
generic trees

back-end
RTL

SSA GIMPLE:

int i, a[16], b[16]
for (i=0; i < 16; i++)
a[i] = a[i] + b[i];

int i_0, i_1, i_2;
int T.1_3, T.2_4, T.3_5;

i_0 = 0;
L1:  i_1 = PHI<i_0, i_2>
    if (i_1 < 16) break;
    T.1_3 = a[i_1 ];
    T.2_4 = b[i_1 ];
    T.3_5 = T.1_3 + T.2_4;
    a[i_1] = T.3_5;
    i_2 = i_1 + 1;
goto L1;

L2:
GCC Passes

front-end
parse trees

middle-end
generic trees

back-end
RTL

genéric trees

into SSA

SSA optimizations

out of SSA

gimple trees

generic trees

GCC 4.0

misc opts

loop optimizations

loop opts

generic trees

vectorization

loop opts

misc opts
GCC Passes

front-end
 parse trees

middle-end
 generic trees

back-end
 RTL

Fortran 95 front-end
IPO
CP
Aliasing
Data layout
Vectorization
Loop unrolling
Scheduler
Modulo Scheduling

The Haifa GCC team:
Leehod Baruch
Revital Eres
Olga Golovanevsky
Mustafa Hagog
Razya Ladelsky
Victor Leikehman
Dorit Naishlos
Mircea Namolaru
Ira Rosen
Ayal Zaks

Power4

machine description
Vectorization in GCC - Talk Layout

Background: GCC
HRL and GCC

Vectorization

Background
The GCC Vectorizer
  Developing a vectorizer in GCC
Status & Results
Future Work

Working with an Open Source Community

Concluding Remarks
Programming for Vector Machines

Proliferation of SIMD (Single Instruction Multiple Data) model
MMX/SSE, Altivec

Communications, Video, Gaming

Fortran90
\[ a[0:N] = b[0:N] + c[0:N] ; \]

Intrinsics
\[ \text{vector float } \text{vb} = \text{vec_load} (0, \text{ptr}_b) ; \]
\[ \text{vector float } \text{vc} = \text{vec_load} (0, \text{ptr}_c) ; \]
\[ \text{vector float } \text{va} = \text{vec_add} (\text{vb}, \text{vc}) ; \]
\[ \text{vec_store} (\text{va}, 0, \text{ptr}_a) ; \]

Auto-vectorization: Automatically transform serial code to vector code by the compiler.
What is vectorization

VF = 4

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

OP(a)
OP(b)
OP(c)
OP(d)

Vector Registers
Data elements packed into vectors
Vector length \( \leq \) Vectorization Factor (VF)

Data in Memory:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>n</th>
<th>o</th>
<th>p</th>
</tr>
</thead>
</table>

OP(a) \[\rightarrow\] VOP( \[\cdot\] VR1 )
Vector operation

vectorization
Vectorization

original serial loop:

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

loop in vector notation:

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

loop in vector notation:

```c
for (i=0; i<(N-N%VF); i+=VF){
    a[i:i+VF] = a[i:i+VF] + b[i:i+VF];
}
```

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

Loop based vectorization

No dependences between iterations
Loop Dependence Tests

```plaintext
for (i=0; i<N; i++){
    D[i] = A[i] + Y
}

for (i=0; i<N; i++){
}

for (i=0; i<N; i++)
    for (j=0; j<N; j++)
```
Loop Dependence Tests

for (i=0; i<N; i++){
    D[i] = A[i] + Y
}

for (i=0; i<N; i++){
    D[i] = A[i] + Y
}

for (i=0; i<N; i++){
    for (j=0; j<N; j++)
}

for (i=0; i<N; i++)

for (i=0; i<N; i++)
    D[i] = A[i] + Y

for (i=0; i<N; i++)

for (i=0; i<N; i++)
**Classic loop vectorizer**

- **data dependence tests**
  - array dependences

**dependence graph**

- find SCCs
- reduce graph
- topological sort

**Cyclic:**
- keep sequential loop for this nest.

**non Cyclic:**
- replace node with vector code

**int exist_dep(ref1, ref2, Loop)**

- **Separable Subscript tests**
  - **ZeroIndexVar**
  - **SingleIndexVar**
  - **MultipleIndexVar** (GCD, Banerjee...)

- **Coupled Subscript tests**
  - (Gamma, Delta, Omega...)

**for i**

**for j**

**for k**

\[ A[5][i+1][j] = A[N][i][k] \]
Vectorizer Skeleton

Basic vectorizer 01.01.2004

- Known loop bound
- Arrays and pointers
- 1D aligned arrays
- Unaligned accesses
- Force alignment
- Invariants
- Conditional code
- Idiom recognition
- Saturation
- Mainline

get candidate loops
- Nesting, entry/exit, countable

Memory references
- Access pattern analysis
- Data dependences
- Alignment analysis

Scalar dependences

Vectorizable operations
- Data types, VF, target support

Vectorize loop

Code:

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

```
li r9,4
li r2,0
mtctr r9
L2:
    lvx v0,r2,r30
    lvx v1,r2,r29
    vaddfp v0,v0,v1
    stvx v0,r2,r0
    addi r2,r2,16
    bdnz L2
```
Vectorization on SSA-ed GIMPLE trees

```c
int i;
int a[N], b[N];
for (i=0; i < 16; i++)
    a[i] = a[i] + b[i];

int T.1, T.2, T.3;

loop:
    if ( i < 16 ) break;
    T.11 = a[i];
    T.12 = a[i+1];
    T.13 = a[i+2];
    T.14 = a[i+3];
    T.21 = b[i];
    T.22 = b[i+1];
    T.23 = b[i+2];
    T.24 = b[i+3];
    T.31 = T.11 + T.21;
    T.32 = T.12 + T.22;
    T.33 = T.13 + T.23;
    T.34 = T.14 + T.24;
    a[i] = T.31;
    a[i+1] = T.32;
    a[i+2] = T.33;
    a[i+3] = T.34;
    i = i + 4;
goto loop;
```

 VF = 4
“unroll by VF and replace”

```c
int i;
int a[N], b[N];
for (i=0; i < 16; i++)
a[i] = a[i] + b[i];

v4si VT.1, VT.2, VT.3;
v4si *VPa = (v4si *)a, *VPb = (v4si *)b;
int indx;

loop:
    if ( indx < 4 ) break;
    VT.1 = VPa[indx];
    VT.2 = VPb[indx];
    VT.3 = VT.1 + VT.2;
    VPa[indx] = VT.3;
    indx = indx + 1;
goto loop;
```
Alignment

Data in Memory:

Alignment support in a multi-platform compiler

- General (new trees: realign_load)
- Efficient (new target hooks: mask_for_load)
- Hide low-level details

Vector Registers

VR1

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

VR2

| e | f | g | h |

VR3

| c | d | e | f |

VR4

VR5

Alignment

OP(c)

OP(d)

OP(e)

OP(f)

(VR1,VR2) ↦ vload (mem)

mask ↦ (0,0,1,1,1,1,0,0)

VR3 ↦ pack (VR1,VR2),mask

VOP(VR3)

misalign = -2
Handling Alignment

Alignment analysis
Transformations to force alignment
loop versioning
loop peeling
Efficient misalignment support

for (i=0; i<N; i++)
{  
  x = q[i]; //misalign(q) = unknown
  p[i] = x; //misalign(p) = -2
}

Peeling for p[i] and versioning:
Loop peeling (for access p[i]):
for (i = 0; i < 2; i++)
{  
  x = q[i]; //misalign(q) = unknown
  p[i] = x; //misalign(p) = -2
}

else 
{
  for (i = 3; i<N; i++)
  {  
    x = q[i]; //misalign(q) = unknown
    p[i] = x; //misalign(p) = -2
  }
}

Vectorization in GCC - Talk Layout

Background: GCC
HRL and GCC

Vectorization

Background
The GCC Vectorizer
Developing a vectorizer in GCC
Status & Results
Future Work

Working with an Open Source Community

Concluding Remarks
Vectorizer Status

In the main GCC development trunk
Will be part of the GCC 4.0 release
New development branch (autovect-branch)
Vectorizer Developers:
  - Dorit Naishlos
  - Olga Golovanevsky
  - Ira Rosen
  - Leehod Baruch
  - Keith Besaw (IBM US)
  - Devang Patel (Apple)
Preliminary Results

Pixel Blending Application
- small dataset: 16x improvement
- tiled large dataset: 7x improvement
- large dataset with display: 3x improvement

```c
for (i = 0; i < sampleCount; i++) {
    output[i] = ( (input1[i] * α)>>8 + (input2[i] * (α-1))>>8 );
}
```

SPEC gzip – 9% improvement

```c
for (n = 0; n < SIZE; n++) {
    m = head[n];
    head[n] = (unsigned short)(m >= WSIZE ? m-WSIZE : 0);
}
```

Kernels:

- lvx v0,r3,r2
- vsubuhs v0,v0,v1
- stvx v0,r3,r2
- addi r2,r2,16
- bdnz L2
Performance improvement (aligned accesses)

- $a[i] = b[i]$
- $a[i+3] = b[i-1]$
- $a[i] = b[i] + 100$
- $a[i] = b[i] \& c[i]$
- $a[i] = b[i] + c[i]$

Bar chart showing performance improvement for different operations with different data types:

- **Float**
- **Int**
- **Short**
- **Char**
Performance improvement (unaligned accesses)

\[ a[i] = b[i] \]
\[ a[i+3] = b[i-1] \]
\[ a[i] = b[i] + 100 \]
\[ a[i] = b[i] & c[i] \]
\[ a[i] = b[i] + c[i] \]
Future Work

1. Reduction
2. Multiple data types
3. Non-consecutive data-accesses
1. Reduction

Cross iteration dependence
Prolog and epilog
Partial sums

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

```
loop:
s_1 = phi (0, s_2)
i_1 = phi (0, i_1)
xa_1 = a[i_1]
xb_1 = b[i_1]
tmp_1 = xa * xb
s_2 = s_1 + tmp_1
i_2 = i_1 + 1
if (i_2 < N) goto loop
```
2. Mixed data types

```c
short b[N];
int a[N];
for (i=0; i<N; i++)
    a[i] = (int) b[i];
Unpack
```
3. Non-consecutive access patterns

A[i, i={0,5,10,15,…}; access_fn(i) = (0,+,5)

\[ VOP(\{a, f, k, p\}) \]

Data in Memory:

```
a b c d e f g h i j k l n n o p
```
Developing a generic vectorizer in a multi-platform compiler

Internal Representation
  machine independent
  high level

Low-level, architecture-dependent details
  vectorize only if supported (efficiently)
    may affect benefit of vectorization
    may affect vectorization scheme
  can’t be expressed using existing tree-codes
Vectorization in GCC - Talk Layout

Background: GCC
HRL and GCC

Vectorization
Background
The GCC Vectorizer
Developing a vectorizer in GCC
Status & Results
Future Work

Working with an Open Source Community

Concluding Remarks
Working with an Open Source Community - Difficulties

It’s a shock

“project management” ??
  No control
  What’s going on, who’s doing what
  Noise

Culture shock
  Language
  Working conventions

How to get the best for your purposes
  Multiplatform
  Politics
Working with an Open Source Community - Advantages

World Wide Collaboration
    Help, Development
    Testing

World Wide Exposure

The Community
Concluding Remarks

GCC

HRL and GCC
Evolving - new SSA framework

GCC vectorizer

Developing a generic vectorizer
in a multi-platform compiler

Open

GCC 4.0

The End