Autovectorization 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

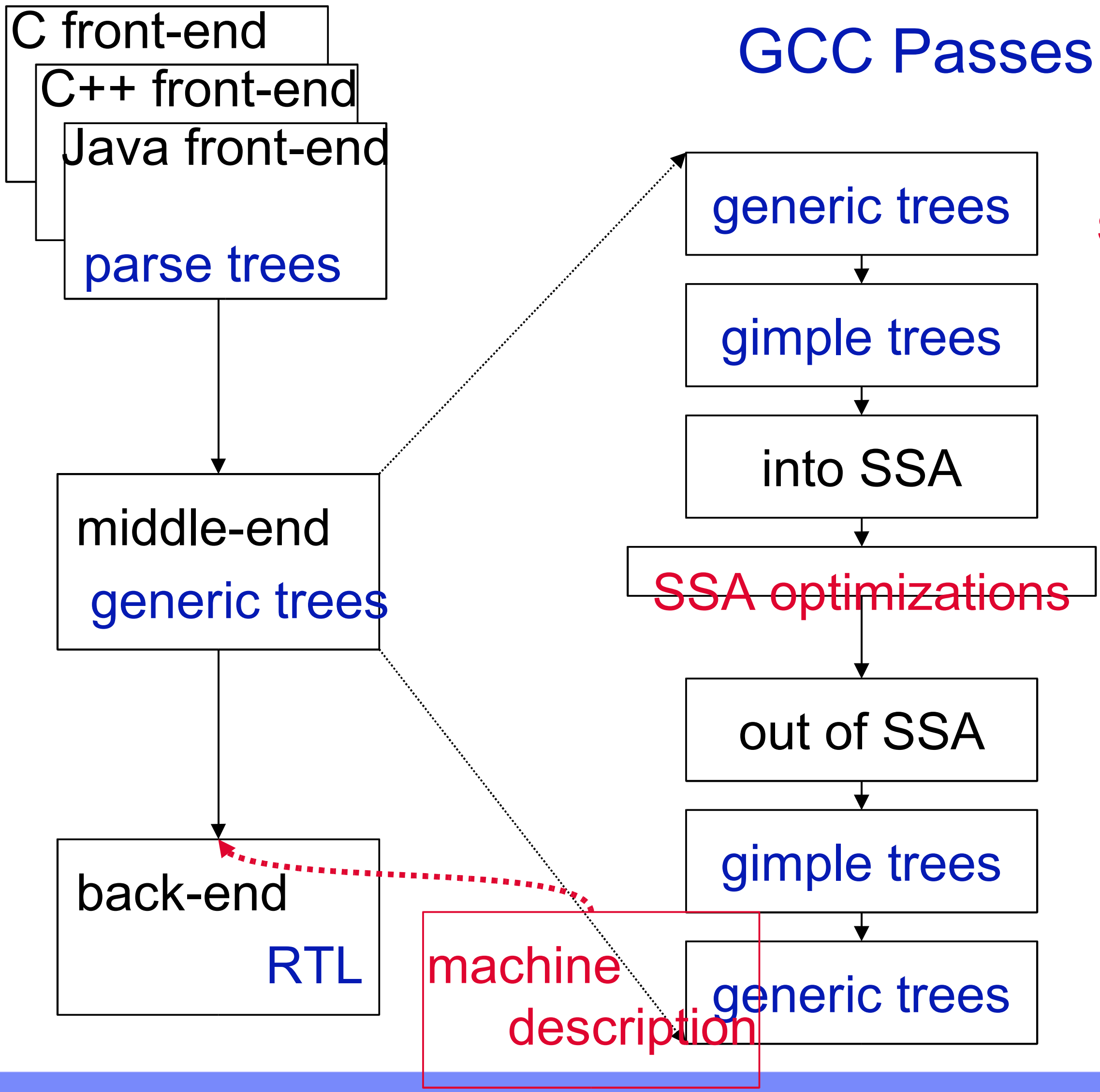
- Multi-platform

- 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)

GCC Passes



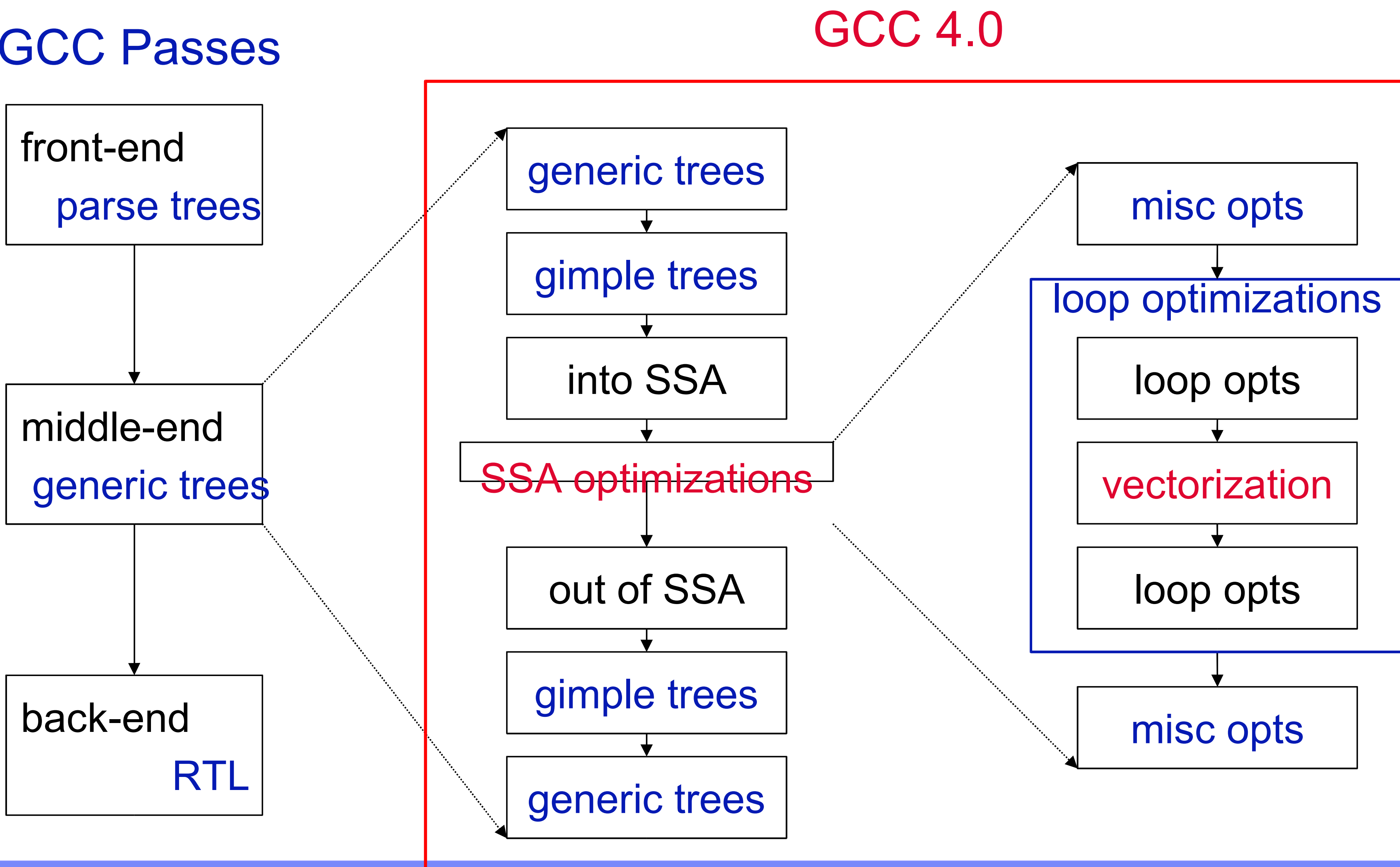

```
int i, a[16], b[16]
for (i=0;  i < 16;  i++)
    a[i] = a[i] + b[i];
```

SSA GIMPLE:

```
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


GCC 4.0
front-end
parse trees
middle-end
generic trees
back-end
RTL
generic trees
gimple trees
into SSA
SSA optimizations
out of SSA
gimple trees
generic trees
misc opts
loop optimizations
loop opts
vectorization
loop opts
misc opts

GCC Passes

front-end
parse trees

middle-end
generic trees

back-end
RTL

machine description

Fortran 95 front-end

IPO
- CP
- Aliasing
- Data layout

Vectorization

Loop unrolling

Scheduler

Modulo Scheduling

Power4

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

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

```
vector float vb = vec_load (0, ptr_b);
vector float vc = vec_load (0, ptr_c);
vector float va = vec_add (vb, vc);
vec_store (va, 0, ptr_a);
```

Autovectorization: Automatically transform serial code to vector code by the compiler.

What is vectorization

VF = 4

	0	1	2	3
VR1	a	b	c	d
VR2				
VR3				
VR4				
VR5				

Vector Registers

OP(a)

OP(b)

OP(c)

OP(d)

vectorization

VOP(VR1)

Vector operation

Data elements packed into vectors

Vector length ↙ Vectorization Factor (VF)

Data in Memory:

a	b	c	d	e	f	g	h	i	j	k	l	n	n	o	p											

Vectorization

original serial loop:

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

vectorization

loop in vector notation:

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

vectorized loop

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

epilog loop

loop in vector notation:

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

Loop based vectorization

No dependences between iterations

Loop Dependence Tests

```
for (i=0; i<N; i++){
    D[i] = A[i] + Y
    A[i+1] = B[i] + X

}
```

```
for (i=0; i<N; i++){
    B[i] = A[i] + Y
    A[i+1] = B[i] + X

}
```

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        A[i+1][j] = A[i][j] + X
```

Loop Dependence Tests

```
for (i=0; i<N; i++){
    A[i+1] = B[i] + X
    D[i] = A[i] + Y
}
```

```
for (i=0; i<N; i++)
    A[i+1] = B[i] + X

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

```
for (i=0; i<N; i++){
    D[i] = A[i] + Y
    A[i+1] = B[i] + X
}
```

```
for (i=0; i<N; i++){
    B[i] = A[i] + Y
    A[i+1] = B[i] + X
}
```

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        A[i+1][j] = A[i][j] + X
```

Classic loop vectorizer

data dependence tests

array dependences

dependence graph

find SCCs

reduce graph

topological sort

for all nodes:

Cyclic:

keep sequential loop for this nest.

non Cyclic:

replace node with vector code

loop transform to break cycles

int exist_dep(ref1, ref2, Loop)

Separable Subscript tests

Zero**I**ndex**V**ar

Single**I**ndex**V**ar

Multiple**I**ndex**V**ar (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

get candidate loops
nesting, entry/exit, countable

- known loop bound

memory references
- **access pattern analysis**
- **data dependences**
- **alignment analysis**

- arrays and pointers
- 1D aligned arrays
- unaligned accesses
- force alignment

scalar dependences

- invariants

vectorizable operations
data-types, VF, target support

- conditional code
- idiom recognition
- saturation

vectorize loop

mainline

```
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

```
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;
S1:    T.1 = a[i ];
S2:    T.2 = b[i ];
S3:    T.3 = T.1 + T.2;
S4:    a[i] = T.3;
S5:    i = i + 1;
goto loop;
```

```
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"

```
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

Alignment support in a multi-platform compiler

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

	0	1	2	3
VR1	a	b	c	d
VR2	e	f	g	h
VR3	c	d	e	f
VR4				
VR5				

Vector Registers

OP(c)

OP(d)

OP(e)

OP(f)

→ VOP(VR3)

(VR1,VR2) ↗ vload (mem)

mask ↗(0,0,1,1,1,1,0,0)

VR3 ↗ pack (VR1,VR2),mask

VOP(VR3)

Data in Memory:

a	b	c	d	e	f	g	h	i	j	k	l	n	n	o	p												

misalign = -2

Handling Alignment

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

Aart J.C. Bik, Milind Girkar, Paul M. Grey, Ximmin Tian. Automatic intra-register vectorization for the Intel architecture. International Journal of Parallel Programming, April 2002.

```
vector *vp_q = q;  vp_q1 = &q[VF-1];
vector *vp_p = p;
int indx = 0, vector vx, vx1, vx2;
LOOP:
 vx = vp_q[indx];

 vp_p[indx] = vx;
 indx++;
```

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

Loop peeling (for access p[i]):

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

Loop versioning:

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

Peeling for p[i] and versioning:

```
for (i = 0; i < 2; i++){
  x = q[i];
  p[i] = x;
}
if (q is aligned){
  for (i = 3; i<N; i++){
    x = q[i]; //misalign(q) = 0
    p[i] = x; //misalign(p) = 0
  }
}else {
  for (i = 3; i<N; i++){
    x = q[i]; //misalign(q) = unknown
    p[i] = x; //misalign(p) = 0
  }
}
```

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

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

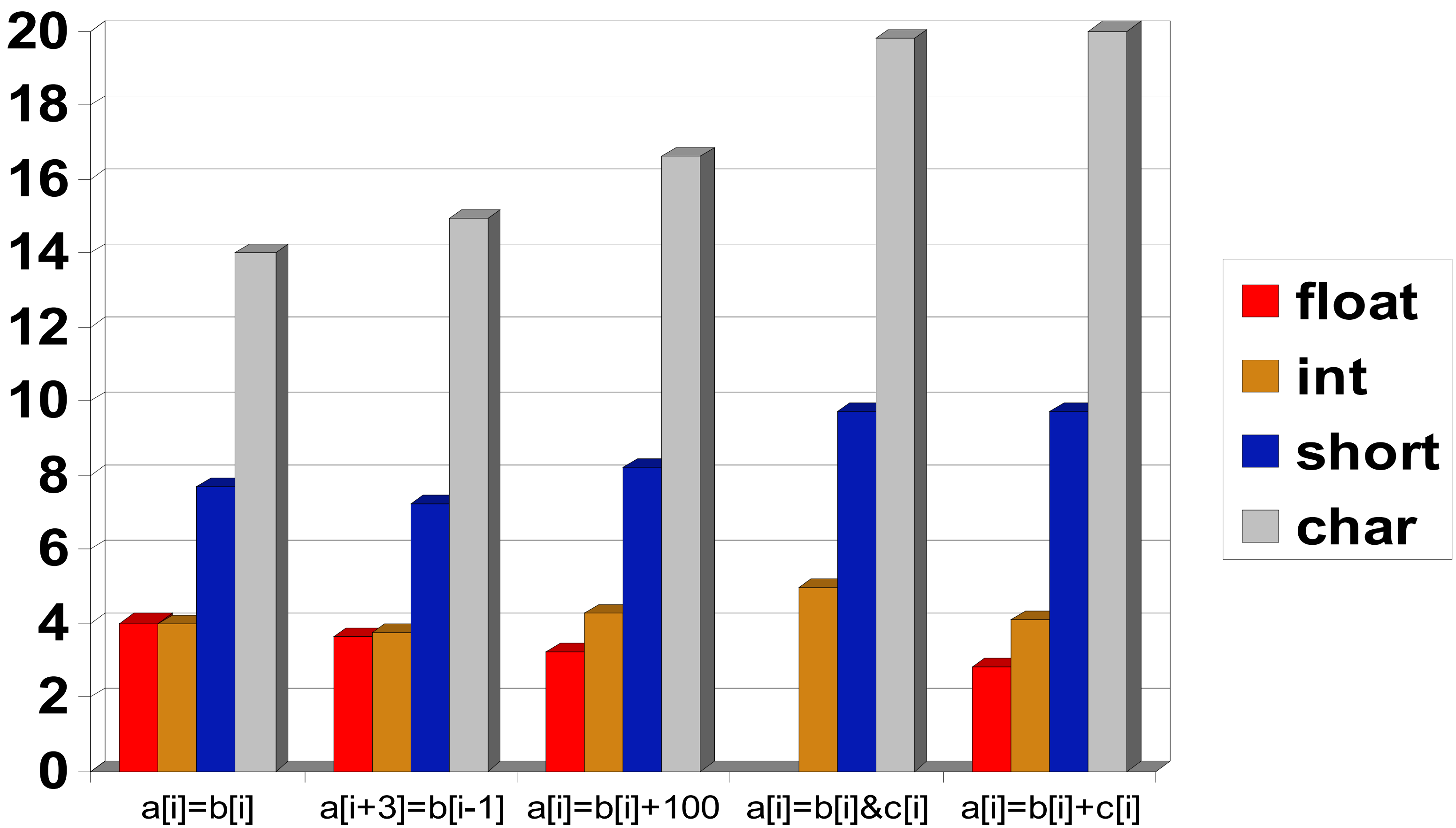
SPEC gzip – 9% improvement

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

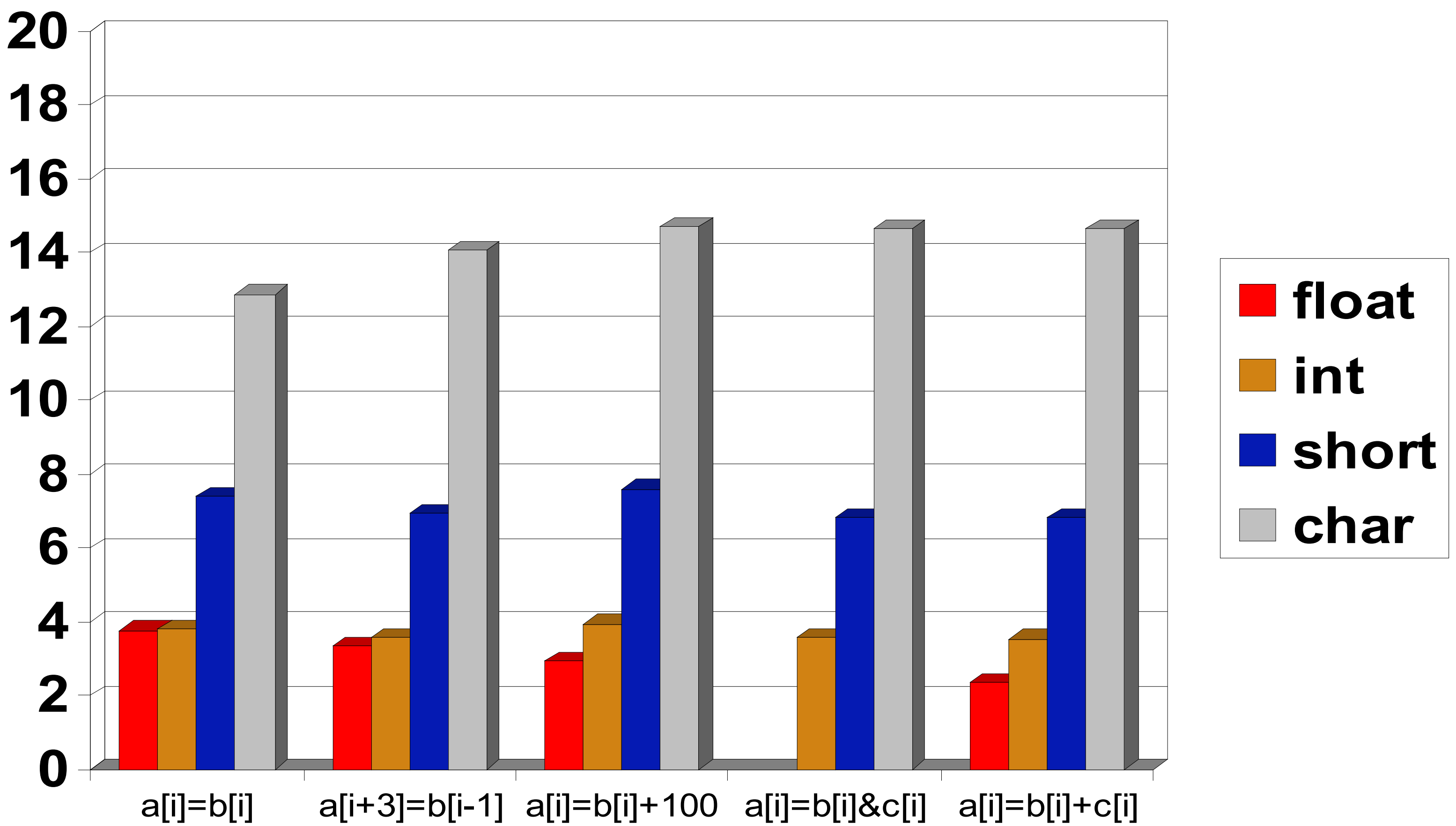
```
lvx v0,r3,r2
vsubuhs v0,v0,v1
stvx v0,r3,r2
addi r2,r2,16
bdnz L2
```

Kernels:

Performance improvement (aligned accesses)



Performance improvement (unaligned accesses)



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];
}
```

s1,s2,s3,s4

0	0	0	0

0	1	2	3	+	0	1	2	3
4	6	8	10	+	4	5	6	7

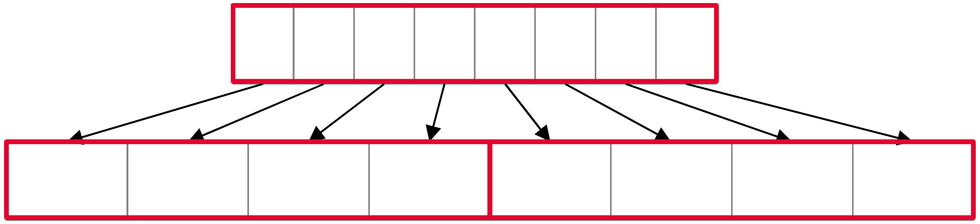
28

```
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

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)

	0	1	2	3
VR1	a	b	c	d
VR2	e	f	g	h
VR3	i	j	k	l
VR4	n	n	o	p
VR5	a	f	k	p

OP(**a**)

OP(**f**)

OP(**k**)

OP(**p**)

→ VOP(VR5)

(VR1,…,VR4) ↗ vload (mem)

mask ↗(1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1)

VR5 ↗ pack (VR1,…,VR4),mask

VOP(VR5)

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

http://gcc.gnu.org/projects/tree-ssa/vectorization.html

The End