Multi-Platform Auto-Vectorization

Dorit Nuzman, IBM
Richard Henderson, RedHat

IBM Labs in Haifa

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

    ```
    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

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

OP(a)

OP(b)

OP(c)

OP(d)

→ VOP(VR1)

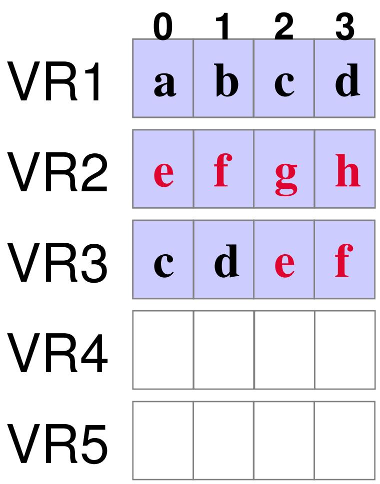
vectorization

Vector operation

Data in Memory:

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

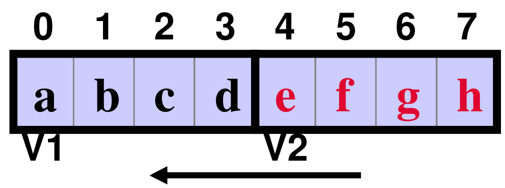
Limitations of SIMD Architectures:
Unaligned memory access



Vector Registers

OP(**c**)
OP(**d**)
OP(**e**)
OP(**f**)

→ VOP(**c, d, e, f**)



V3 ← vec-shift-left v1, v2, 2

V3 ← vec-permute v1, v2, {2,3,4,5}

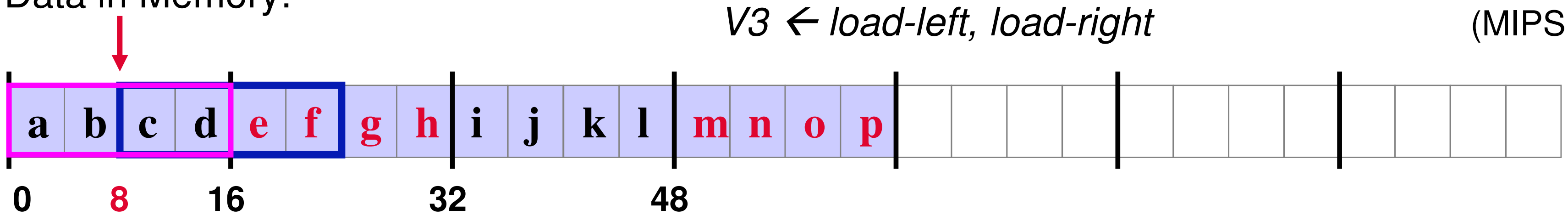
V1 ← extql (v1, addr), v2 ← extqh (v2, addr) (alpha)

V3 ← vec-or (v1, v2)

V3 ← alvn.ps v1,v2,addr (MIPS64)

V3 ← load-left, load-right (MIPS MDMX)

Data in Memory:

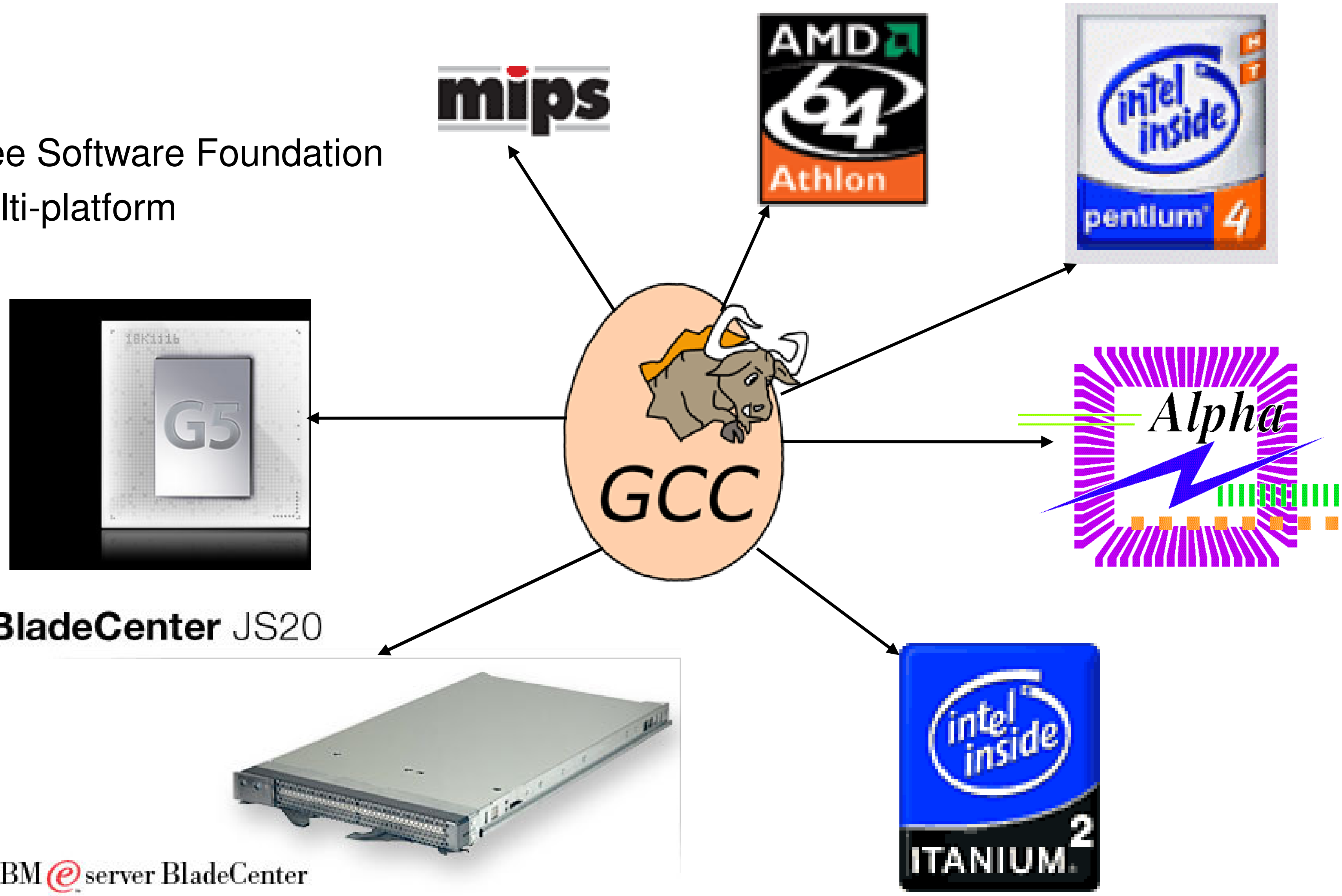


Multi-Platform Auto-Vectorization - Talk Layout

- Vectorization for SIMD
  - Alignment Example
- Vectorization in GCC
- Vector Abstractions
  - Abstractions for Alignment
- Multi-platform Evaluation
- Related Work & Conclusion

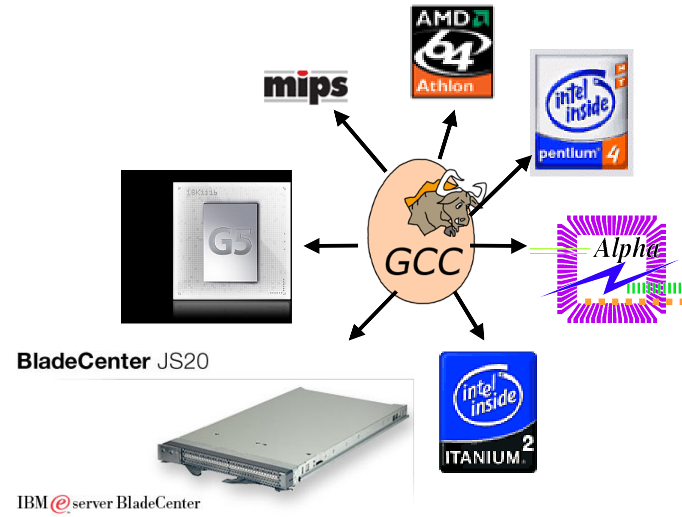
GCC

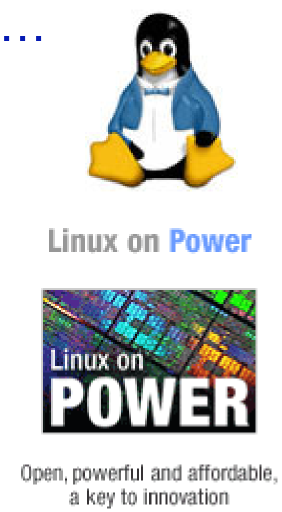
- Free Software Foundation
- Multi-platform



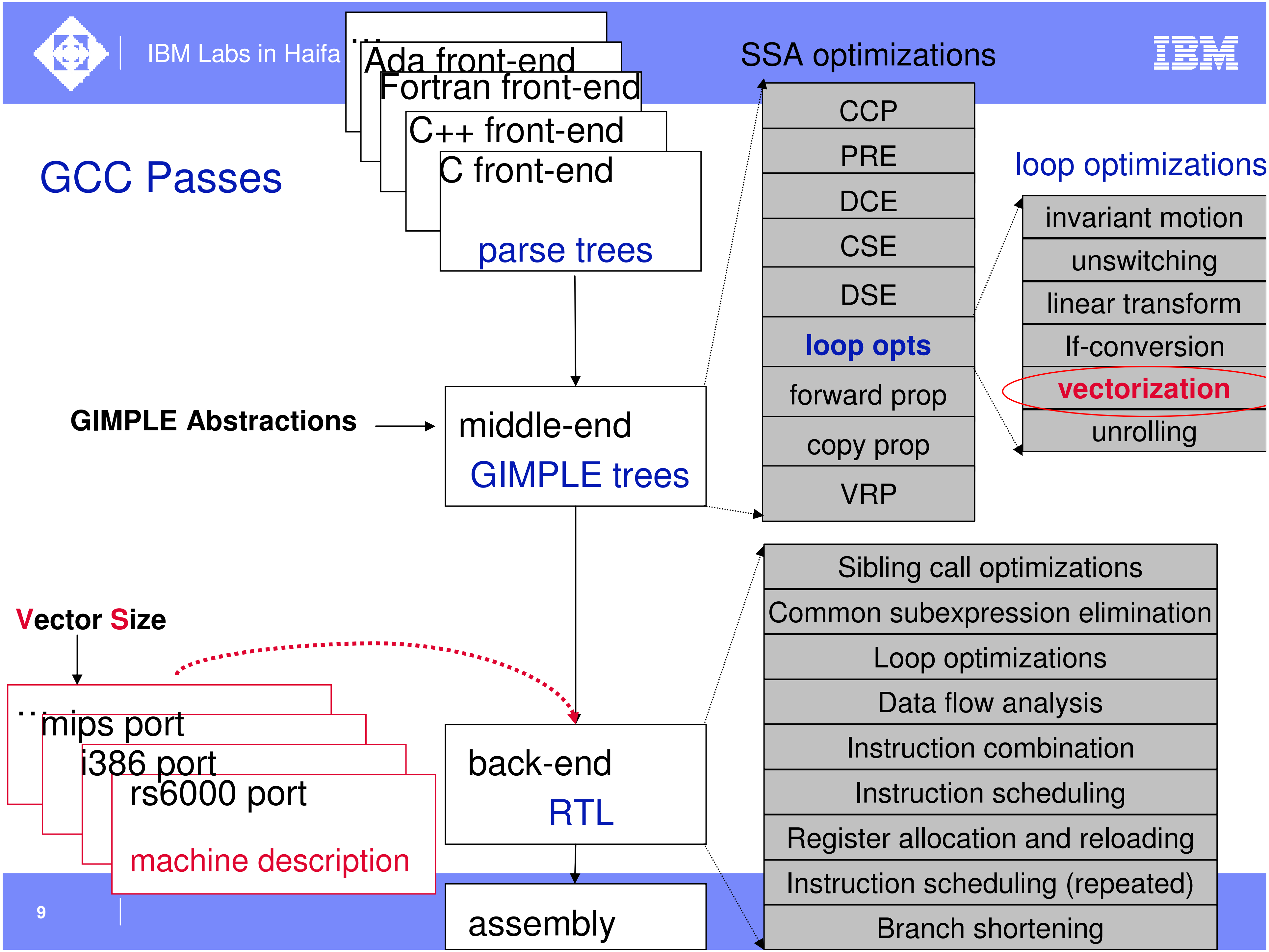
GCC

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





GCC Passes


...
Ada front-end
Fortran front-end
C++ front-end
C front-end
parse 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
GIMPLE Abstractions
middle-end
GIMPLE trees
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
back-end
RTL
assembly

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

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

- **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,…

s1,s2,s3,s4

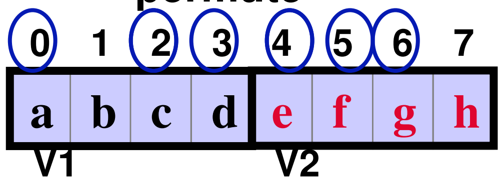
4	6	8	10

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

optab/type	char	short	int	v16char	v8short	v4int
add	f1	f2	f3	f4	f5	f6

Vector Abstractions: Considerations

permute



- 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 convensions
  - naming, existing-operation-codes, default values…
- Performance
  - Translates to most efficient code

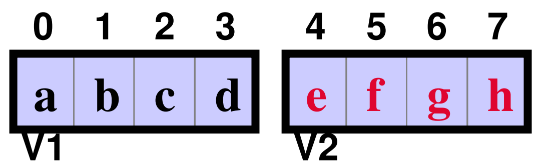
subtract-and-saturate, dot-product

Vector Abstractions: Abstractions for alignment

- Implicit Realignment
  - misaligned_ref (ptr, mis)
- Explicit Realignment
  - aligned_ref (ptr)
  - realign_load (v1, v2, RT)
  - Realignment Token (RT)

V3 ← movdqu (MMX/SSE)

V3 ← load-left, load-right (MIPS MDMX)



V3 ← vec-shift-left v1, v2, ***2***

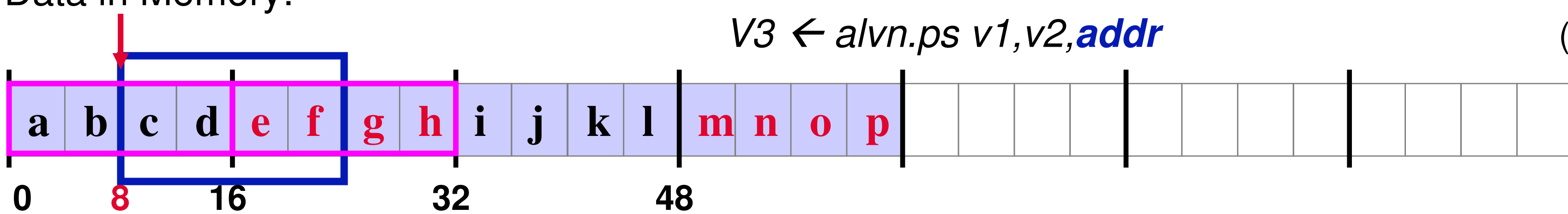
V3 ← vec-permute v1, v2, ***{2,3,4,5}*** (Altivec)

V1 ← extql (v1, ***addr****), v2 ← extqh (v2,* ***addr****)*

V3 ← vec-or (v1, v2) (Alpha)

*V3 ← alvn.ps v1,v2,****addr*** (MIPS64)

Data in Memory:



Handling Alignment

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

```
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;
```

```
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; vx1 = vx2;
```

Handling Alignment

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

```
addra_0 = &a[0];
adrb = &b[0];
vector vx;


addra_i = addra_0;
LOOP:
  vx = misaligned_ref (addra_i,0);


 indirect_ref (addrb) = vx;
  addra_i += 16; addrb += 16;
```

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

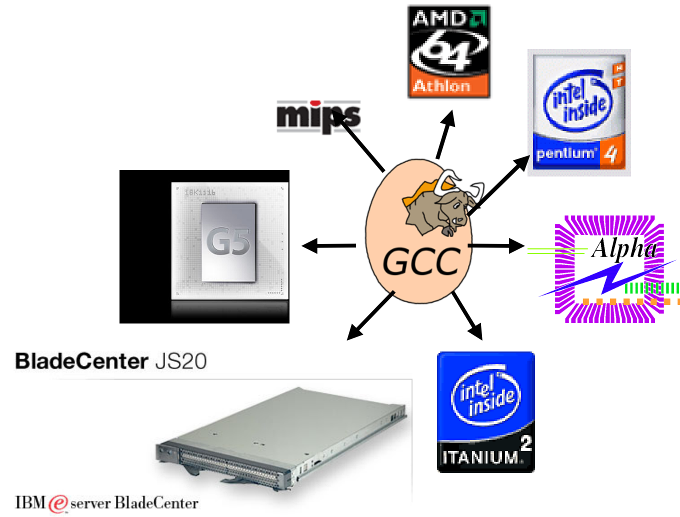
- Alignment:
  - misaligned_ref, align_ref
  - realign_load, target_get_RT
- Reduction:
  - reduc_plus
- Special patterns:
  - dot_prod, sad
  - sub_sat
  - widen_mult, widen_sum
- Conditional operations:
  - (cond) ? x : y
- Type Conversions
  - unpack_high, unpack_low
  - pack_mod, pack_sat
- Strided-Accesses:
  - extract_odd, extract_even
  - interleave_high, interleave_low

Multi-Platform Auto-Vectorization - Talk Layout

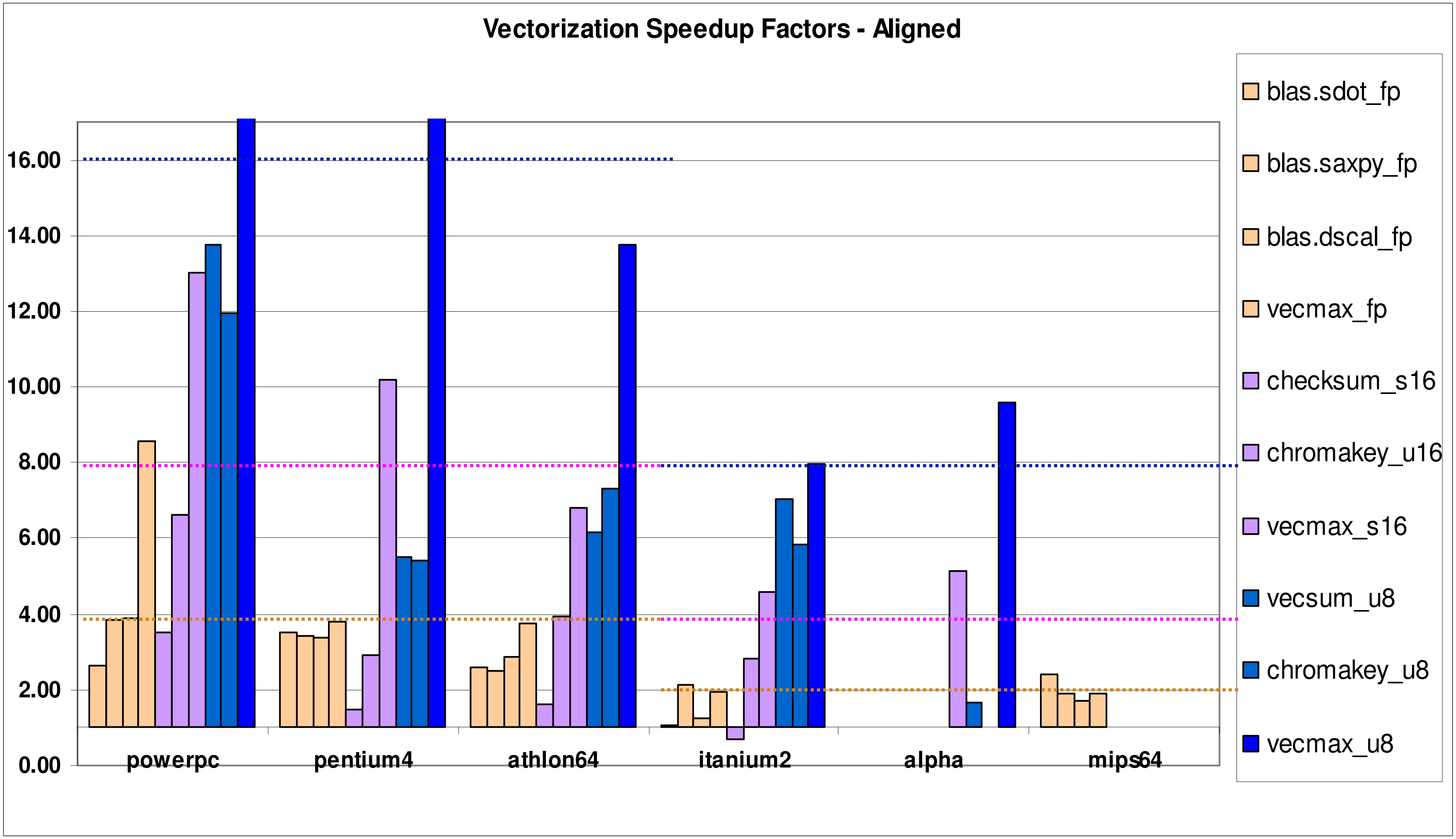
- Vectorization for SIMD
  - Alignment Example
- Vectorization in GCC
- Vector Abstractions
  - Abstractions for Alignment
- Multi-platform Evaluation
- Related Work & Conclusion

Multi-Platform Evaluation

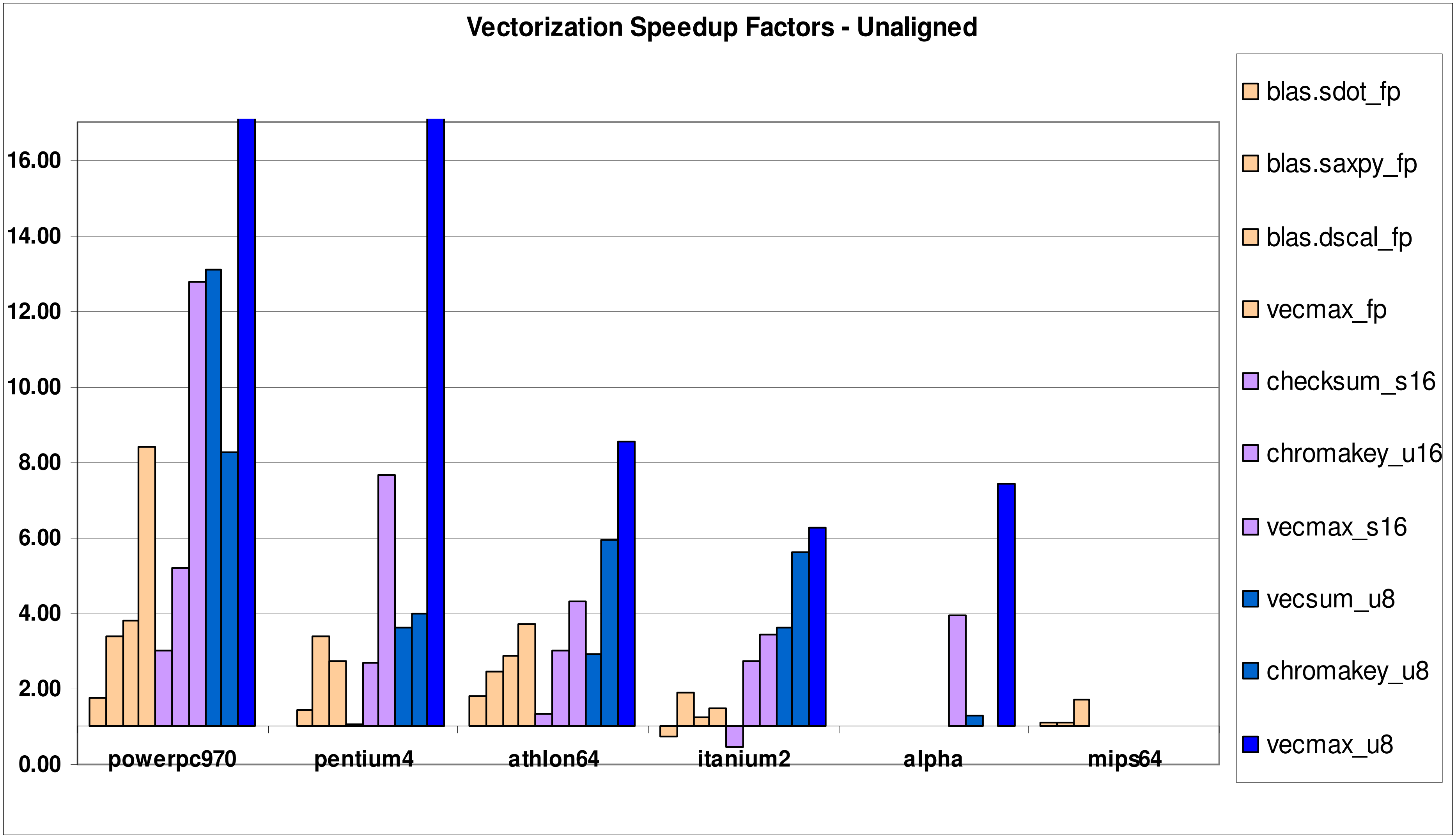
- IBM PowerPC970, Altivec (VS = 16)
- Intel Pentium4, SSE2 (VS = 16)
- AMD Athlon64, SSE2 (VS = 16)
- Intel Itanium2 (VS = 8)
- MIPS64, paired-single-fp (VS = 8)
- Alpha (VS = 8)



Multi-Platform Evaluation



Multi-Platform Evaluation



Related Work

- Vectorizing compilers available for a specific architecture
  - XL (Eichenberger, Wu). Altivec (new: bluegene, cell)
  - 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
  - http://gcc.gnu.org/projects/tree-ssa/vectorization.html

The End