

IBM



Putting Haifa on the Free-Software Map: GCC Development at IBM Research

Dorit Naishlos
IBM Haifa Labs
dorit@il.ibm.com





Putting Haifa on the Free-Software Map: GCC Development at IBM Research



Talk Goals:

To share our technical and social experience in developing and contributing optimizations to GCC

To show key technical contributions we made to GCC, and the challenges involved

To invite you to have a closer look and participate



Putting Haifa on the Free-Software Map: GCC Development at IBM Research - Talk Layout



1. Introduction
2. GCC structure
3. Haifa's recent contributions to GCC
 - Vectorization (GCC 4.0)
 - Modulo scheduling (GCC 4.0)
 - Inter-Procedural constant propagation (GCC 4.1)
 - Data-layout optimizations (GCC 4.x ?)
4. Concluding Remarks
 - Working with the GCC community



GCC – GNU Compiler Collection

Free Software Foundation (FSF)

2.1 million lines of code, 18 years of development (since 1987)

Multiple programming languages

C, C++, Objective-C, Ada, Java, Fortran

Multiplatform: Multiple target processors and operating systems

IBM eServers, RISC, embedded, ... (over 30 architectures!)

Linux, Unix, Windows, OS/X, embedded, ...

GCC is de-facto standard in the Linux eco-system



GCC – Development Process

Organizational Structure

Global source code repository (cvs)

gcc-patches@gcc.gnu.org, IRC

Global (12) and subsystem (ca. 80) maintainers

GCC Steering Committee (13 members)

Development Cycle

Staged approach

(1: major changes, 2: minor changes, 3: bugfixes only)

Development branches for preparing major changes

Release branches for QA cycle and maintenance

Development Policies

Mandatory design and code review by maintainers

Successful test cycle pre-requisite for each code check-in

Coding style guidelines strictly enforced



GCC – Development Process (Cont')

Legal Issues

Copyright held by the FSF (assignments required!)

Licensed under the GPL

Who's involved

Volunteers

Academia

Linux distros

Red Hat, SUSE

IBM

IBM LTC

Haifa Lab

IBM Research

Vendors: CodeSourcery, ARM, AdaCore, more





IBM and Open Source

Invests and participates in major OSS projects

Apache (1998), <http://www.eclipse.org>,
GCC haifa-scheduler (1997), more

why?

January 2001 - \$1 billion investment in **linux**

Linux Technology Center (LTC): <http://ibm.com/linux/ltc>

250+ IBM developers worldwide: Adelaide, Austin, Bangalore, Beaverton, Boeblingen, Boulder, Canberra, Chicago, Denver, **Haifa**, Hawthorne, Hursley, Tokyo, Mount Laurel, Portland, Poughkeepsie, Raleigh, Rochester, San Francisco, Seattle, Somers, and Yorktown

January 2005 - Opened 500 patents to the Open Source community



IBM and Open Source – why?

<http://www.research.ibm.com/journal>

Use OSS as a business tool to promote IBM products/platforms

“Power everywhere”

<http://www.power.org>

Linux on Power



Open, powerful and affordable,
a key to innovation

Extend IBM mindshare

Build relationships with a broad spectrum of developers

Drive rapid adoption of open standards

Apache, XML, Globus grid architecture



UNLEASH THE POWER
of Linux with IBM eServer OpenPower™ 710 —
Muscle of IBM POWER5™ processor,
mainframe-inspired reliability,
astounding price.

→ Learn more

BladeCenter JS20



IBM eServer BladeCenter





IBM and Open Source - GCC

IBM and GCC: why

Part of the “Power everywhere” and Linux initiatives
improve support for IBM platforms
available for research in academia

IBM's contributions to GCC (few examples):

Communication and coordination:

Maintainers, steering committee, GCC summit
bugzilla, wiki

New frontend (PL8) and backend (zSeries)

Optimizations

More



Putting Haifa on the Free-Software Map: GCC Development at IBM Research - Talk Layout



Introduction

GCC structure



Haifa's recent contributions to GCC

Vectorization (GCC 4.0)

Modulo scheduling (GCC 4.0)

Inter-Procedural constant propagation (GCC 4.1)

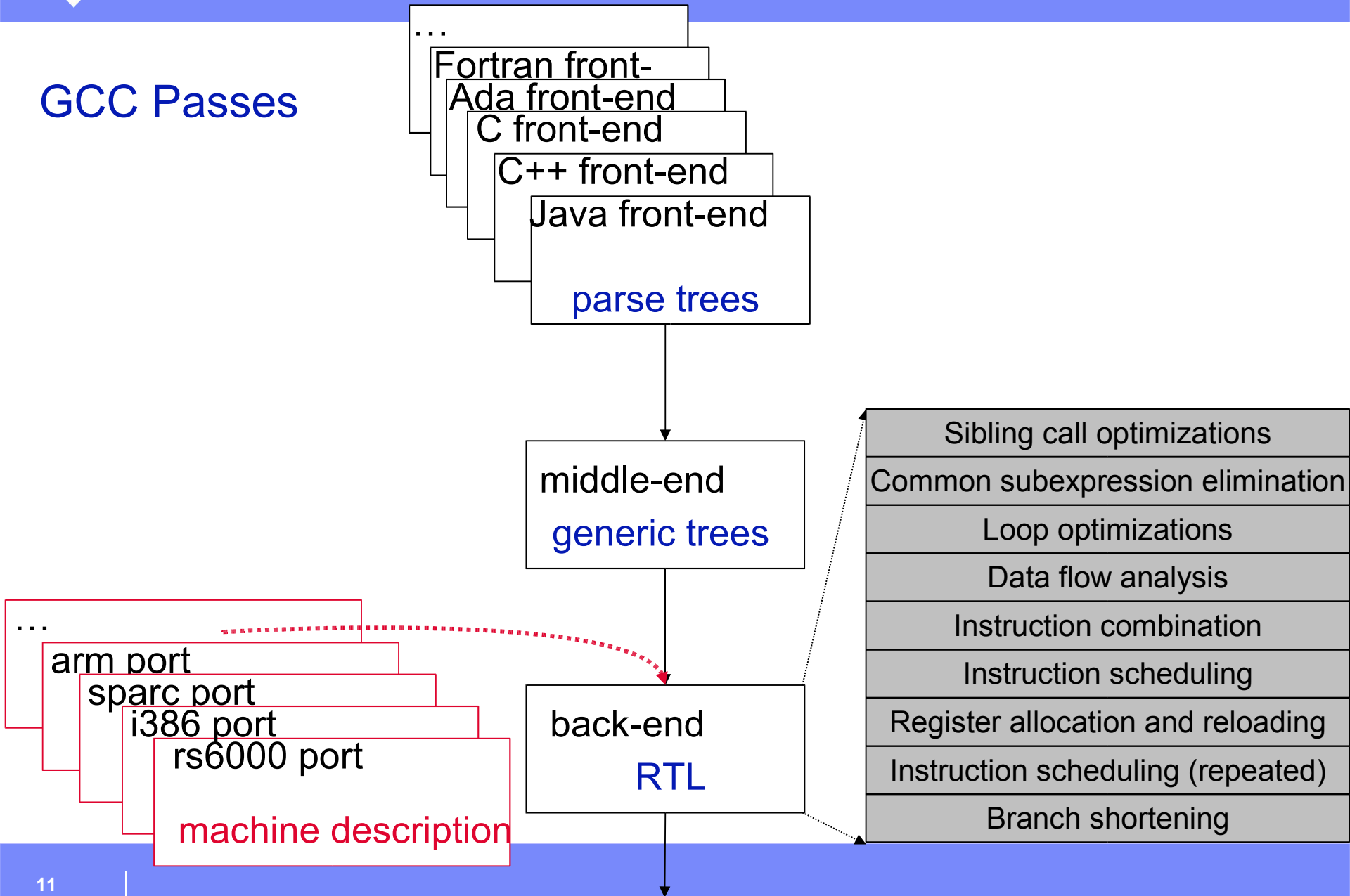
Data-layout optimizations (GCC 4.x ?)

Concluding Remarks

Working with the GCC community



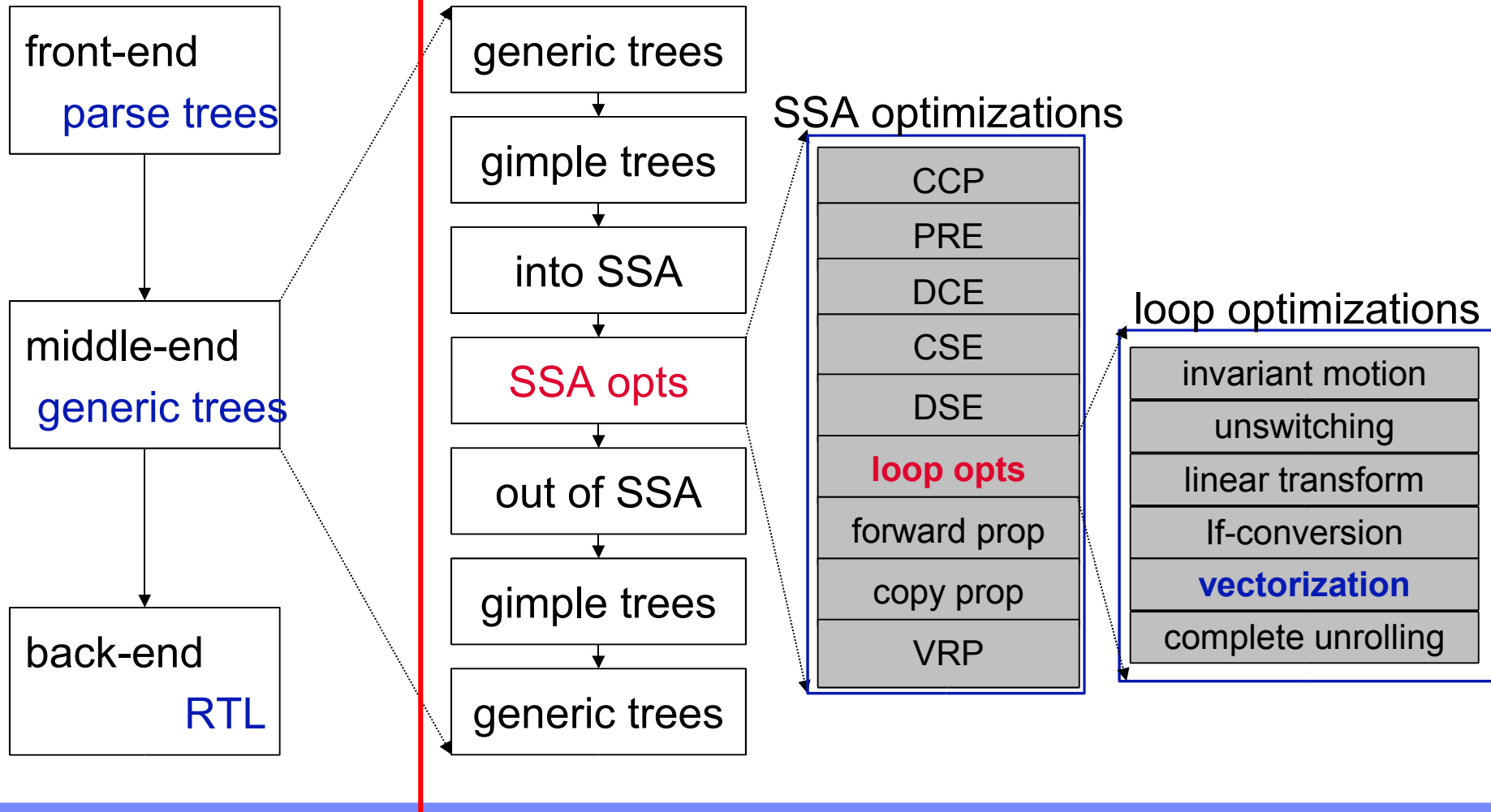
GCC Passes

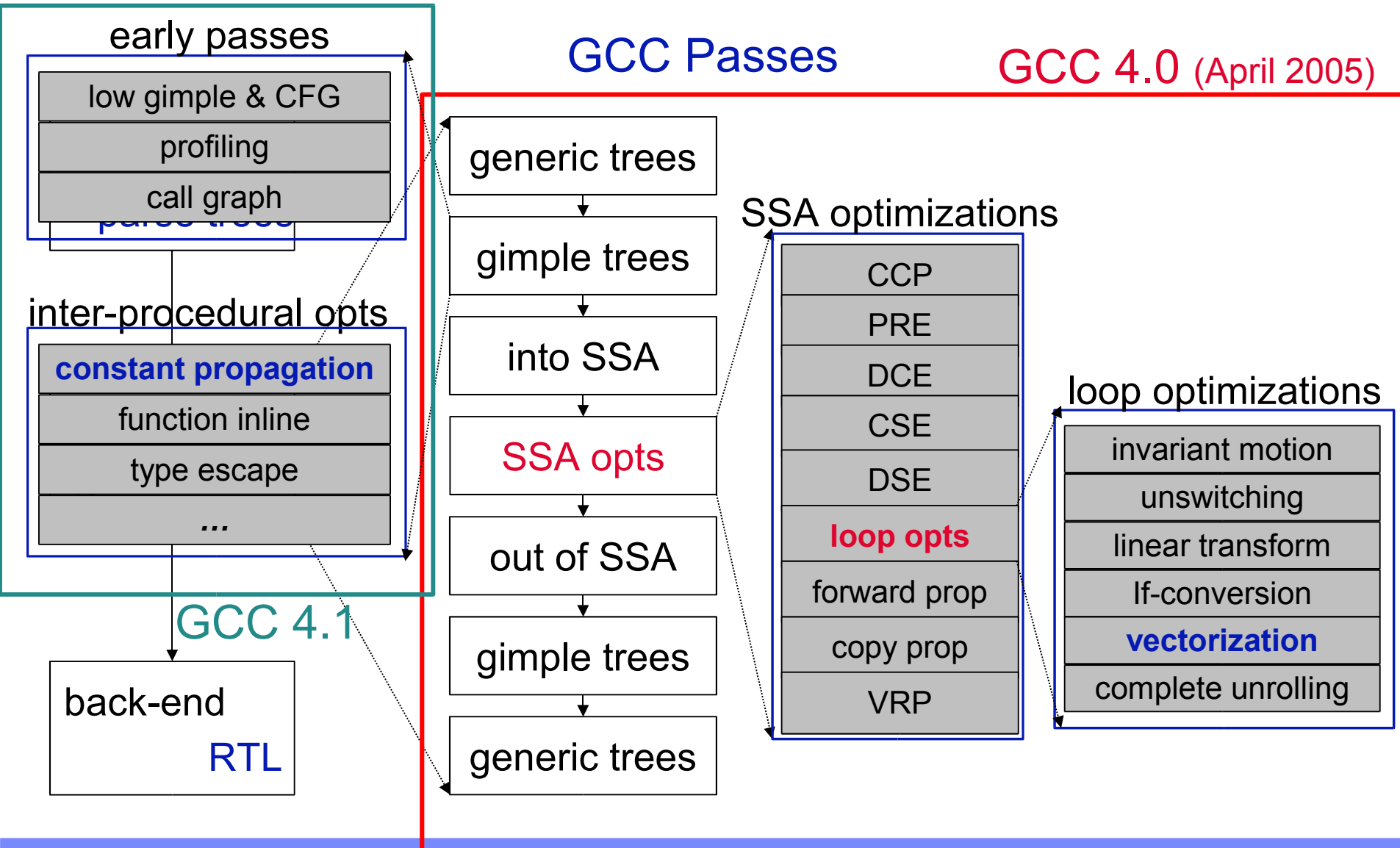




GCC Passes

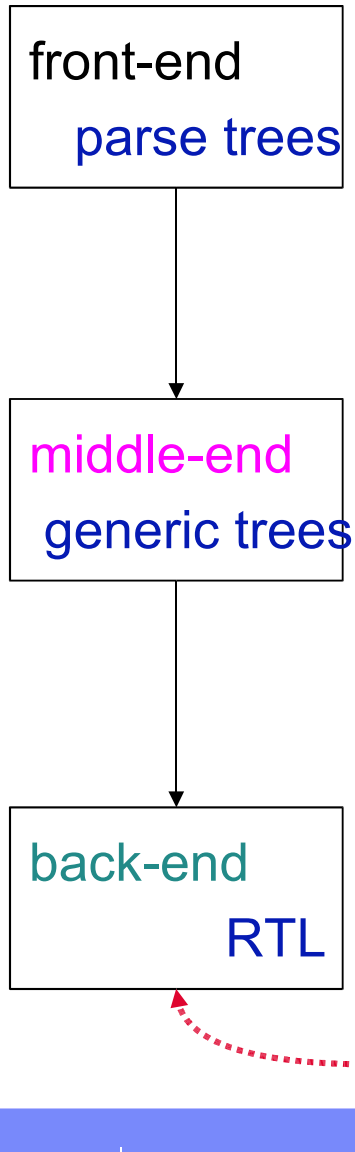
GCC 4.0 (April 2005)







Haifa Contributions



Fortran 95 front-end (4.0)

Vectorization (4.0) **1**

Inter procedural optimizations:

Constant Propagation (4.1) **3**

Aliasing (4.x)

Data layout (4.x) **4**

Modulo Variable expansion (4.0)

scheduling enhancements (4.0)

Modulo Scheduling (4.0) **2**

Power5 (4.0)

machine
description



Putting Haifa on the Free-Software Map: GCC Development at IBM Research - Talk Layout



Introduction

GCC structure

Haifa's recent contributions to GCC ←

Vectorization (GCC 4.0)

Modulo scheduling (GCC 4.0)

Inter-Procedural constant propagation (GCC 4.1)

Data-layout optimizations (GCC 4.x ?)

Concluding Remarks

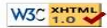
Working with the GCC community



GCC Developers' Summit 2004

Navigation

[Registration](#)
[Paper Presentations](#)
[BOFS/Meetings](#)
[Sponsors](#)
[Venue](#)
[FAQ](#)
[Travel & Hotel](#)
[OpenPGP Keysigning](#)
[Home](#)
[2003 Home](#)



Developers' Summit

Paper Presentations

- | | | |
|--|---------------------------------------------------------------------------------------|----------------------------------------------------------|
| | Árpád Beszédes | University of Szeged |
| | CSIBE Benchmark : One Year Perspective and Plans | |
| | Paolo Carlini | SUSE |
| | Performance work in the libstdc++-v3 project | |
| | Zdenek Dvorak | SuSE |
| | Declarative world inspiration | |
| | David Edelsohn | IBM Research |
| | High-Level Loop Optimizations for GCC | |
| | Mostafa Hagog | IBM Research Lab in Haifa |
| | Swing Modulo Scheduling for GCC | |
| | Jan Hubicka | SUSE CR |
| | Call graph module in GCC (framework for intraprocedural optimization) | |
| | Janis Johnson | IBM Linux Technology Center |
| | GCC Binary Compatibility Test Framework | |
| | Gábor Lóki | University of Szeged |
| | Code Factoring in GCC | |
| | Vladimir N. Makarov | Red Hat |
| | Fighting register pressure in GCC | |
| | Dorit Naishlos | IBM Haifa Labs |
| | Autovectorization in GCC | |
| | Diego Novillo | Red Hat |
| | Design and Implementation of Tree SSA | |
| | Mukta Punjani | Cadence Design Systems India Pvt Limited |
| | Register Rematerialization In GCC | |
| | Naveen Sharma | HCL Technologies Ltd |
| | Addressing Mode Selection in GCC | |
| | Nathan Sidwell | CodeSourcery |
| | Statically Typed Trees | |
| | Tom Tromey | Red Hat, Inc. |
| | Gc: the new ABI and its implications | |



Haifa's contributions to GCC:

(1) Vectorization (4.0) – layout:

What is vectorization

Vectorization in GCC

Overview

Alignment

Issues

Results

Status



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				

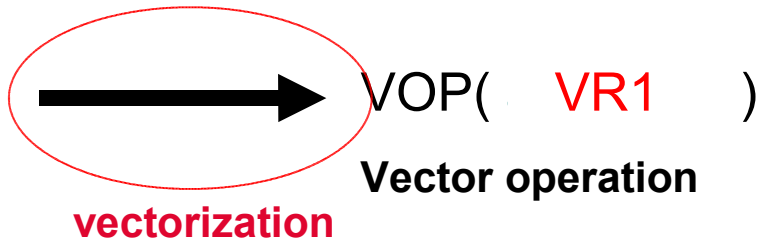
Vector Registers

OP(a)

OP(b)

OP(c)

OP(d)



Data elements packed into vectors

Vector length \rightarrow 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; i+=VF){  
    a[i:i+VF] = a[i:i+VF] + b[i:i+VF];  
}
```

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
```

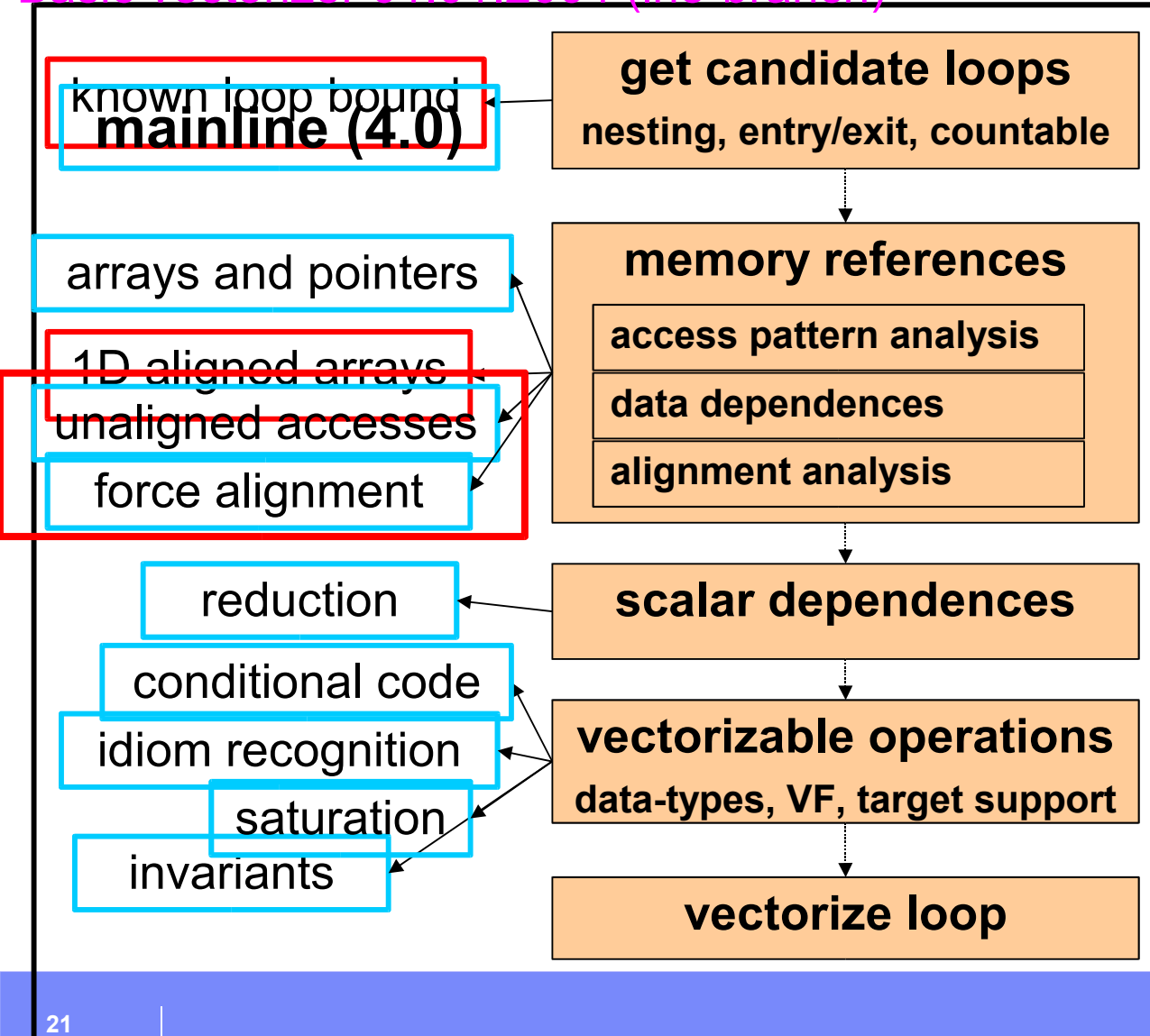
Loop based vectorization

No dependences between iterations



Vectorizer Skeleton

Basic vectorizer 01.01.2004 (Ino-branch)



```
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
```



Alignment

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

Vector Registers

Alignment support in a multi-platform compiler

General (new trees: realign_load)

Efficient (new target hooks: mask_for_load)

Hide low-level details

$$OP(c)$$
$$\text{OP}(d)$$
$$OP(e)$$
$$\text{OP}(\mathbf{f})$$

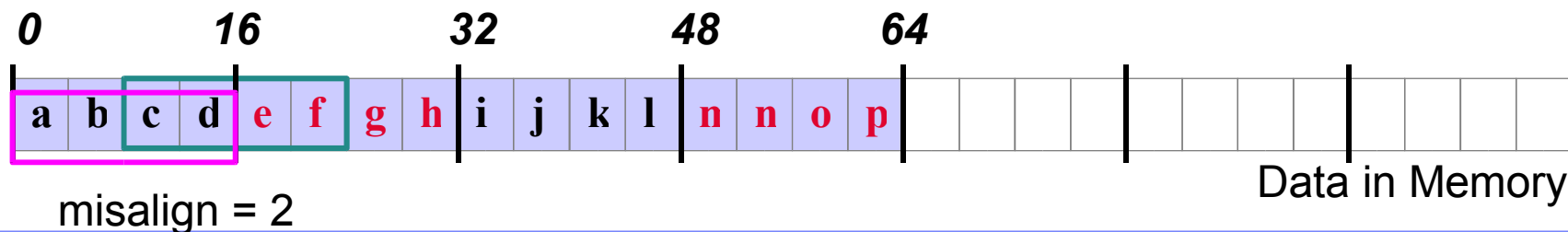
→ VOP(VR3)

(VR1,VR2) ← vload (mem)

mask $\leftarrow (0,0,1,1,1,1,0,0)$

VR3 \leftarrow pack (VR1,VR2),mask

VOP(VR3)





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



Vectorizer Status

Part of the GCC 4.0 release

4.1 will include:

- Reduction

- Versioning for alignment

New development branch (autovect-branch). In the works:

- Special idioms – saturation, dot product

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

Current Vectorizer Developers:

- Dorit Naishlos (IBM Haifa)

- Ira Rosen (IBM Haifa)

- 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] *  $\alpha$ ) >> 8 + (input2[i] * ( $\alpha$ -1)) >> 8 );  
}
```

SPEC gzip – **9%** improvement

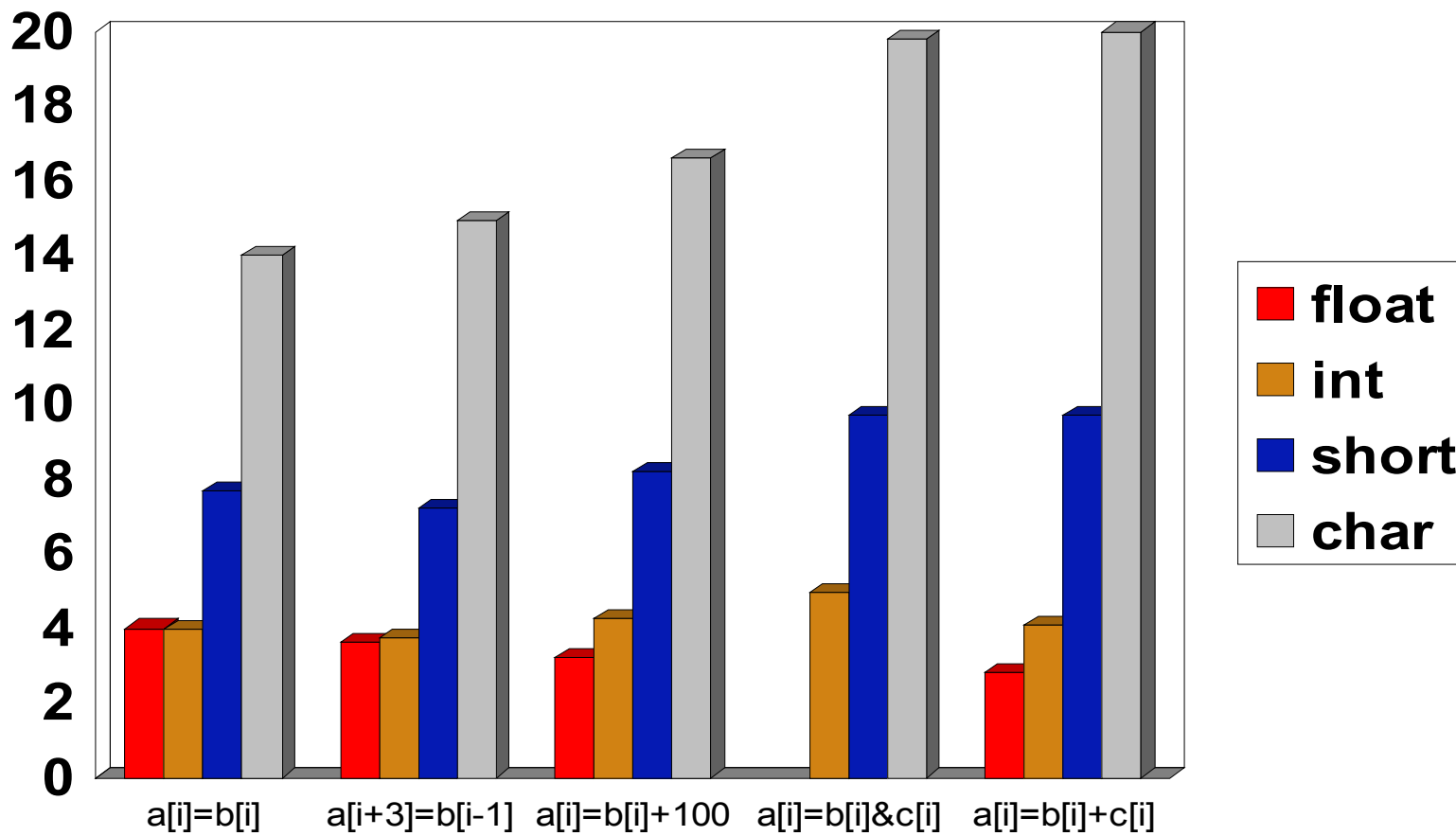
```
for (n = 0; n < SIZE; n++) {  
    m = head[n];  
    head[n] = (unsigned short)(m >= WSIZE ? m-WSIZE : 0);  
}
```

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

Kernels:

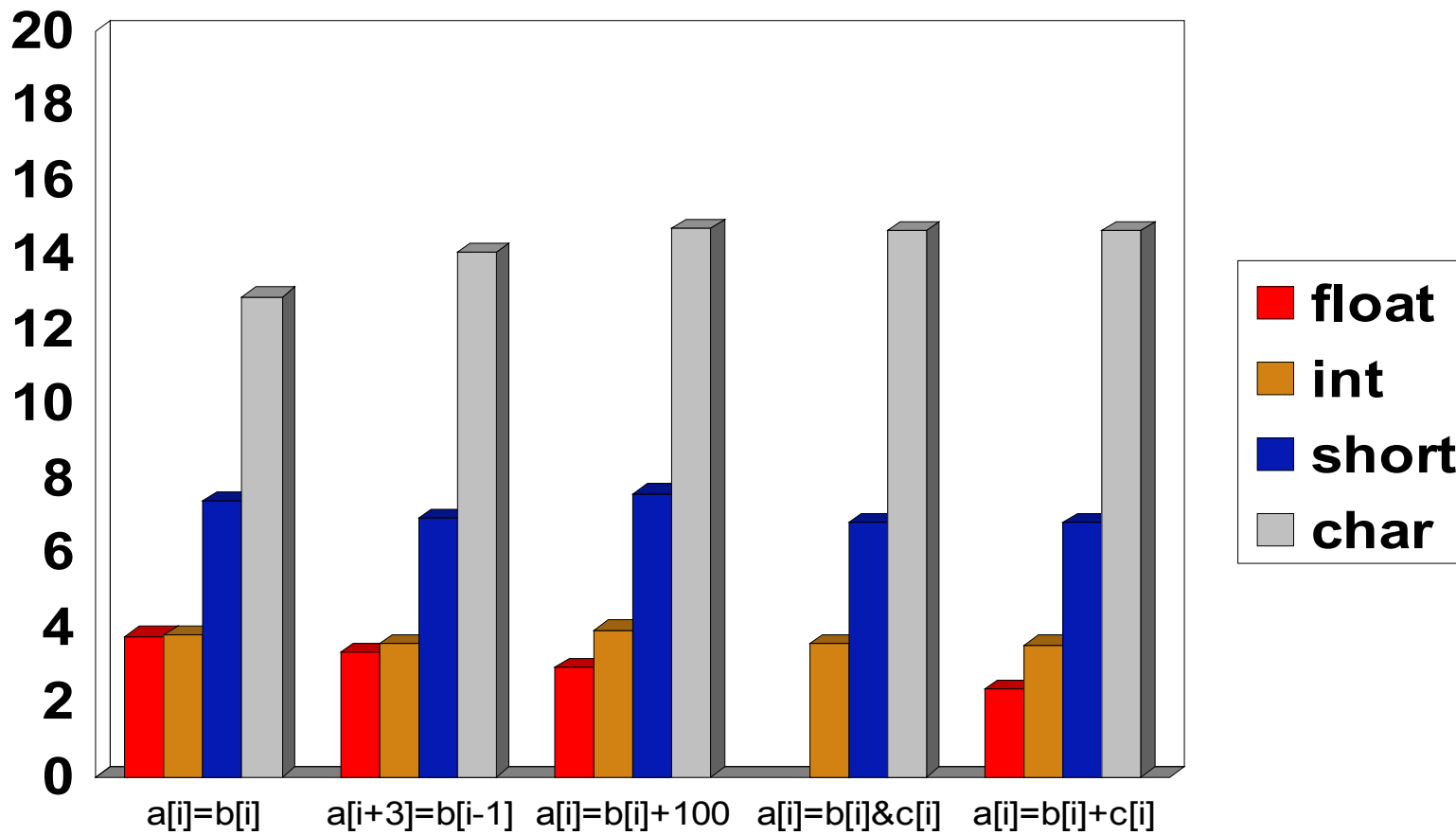


Performance improvement (aligned accesses)





Performance improvement (unaligned accesses)





Putting Haifa on the Free-Software Map: GCC Development at IBM Research - Talk Layout



Introduction

GCC structure

Haifa's recent contributions to GCC

Vectorization (GCC 4.0)

Modulo scheduling (GCC 4.0) ←

Inter-Procedural constant propagation (GCC 4.1)

Data-layout optimizations (GCC 4.x ?)

Concluding Remarks

Working with the GCC community



Haifa's contributions to GCC: (2) Modulo Scheduling (4.0)

Mustafa Hagog, Ayal Zaks

Based on:

- [1] E. Ayguade M. Valero. J. Llosa, A. Gonzalez. **Swing modulo scheduling: A lifetime sensitive approach.** In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)*, pages 80–87, Boston – Massachussets - USA, 1996.
- [2] E. Ayguade M. Valero. J. Llosa, A. Gonzalez and J. Eckhardt. **Lifetime-sensitive modulo scheduling in a production environment.** *IEEE Trans. on Comps.*, 50:3, 2001.



Scheduling

Current scheduling of loops in GCC – cycle schedule the loop body

source code

```
float *a, *b;  
for (i = 0 ; i < N ; i++)  
    sum += a[i] + b[i];
```

pre-scheduled instructions

```
L5:  
2. slwi r0,r2, 2  
3. lfsx f13,r4,r0  
4. lfsx f0,r3,r0  
5. fadds f0,f0,f13  
6. fadds f1,f1,f0  
7. addi r2,r2,1  
8. bdnz L5
```

schedule
the body

Cycle	op
[0]:	1, 6
[1]:	2, 3
[2]:	-
[3]:	4
[4]:	-
[5]:	-
[6]:	5,7

Time = 7*N cycles

resources: 4 issue, 2 alu, 2 ldstu , 2 fpu
latencies: add & slwi = 1, ld = 2, addfp = 3,



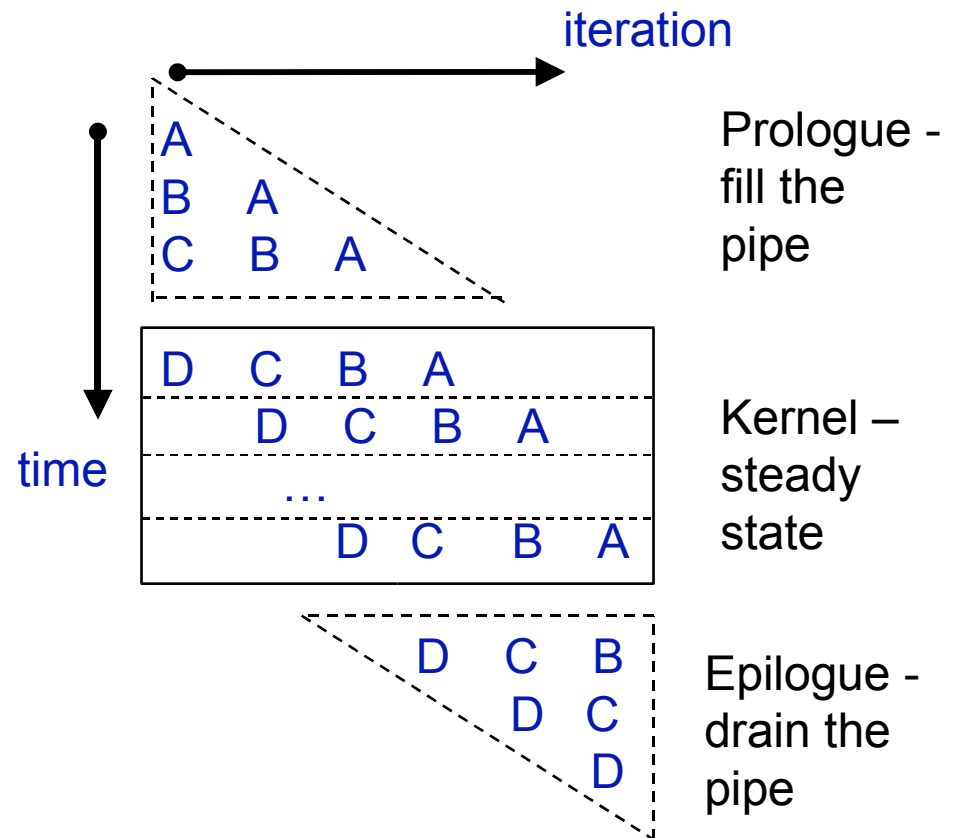
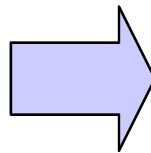
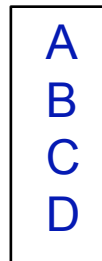
Software Pipelining

Unrolling by larger factor can increase ILP, but

1. hard to determine best unroll factor, and
2. increases code size.

Software Pipelining – gives the same effect with much less code bloat

Loop body
with 4 ops



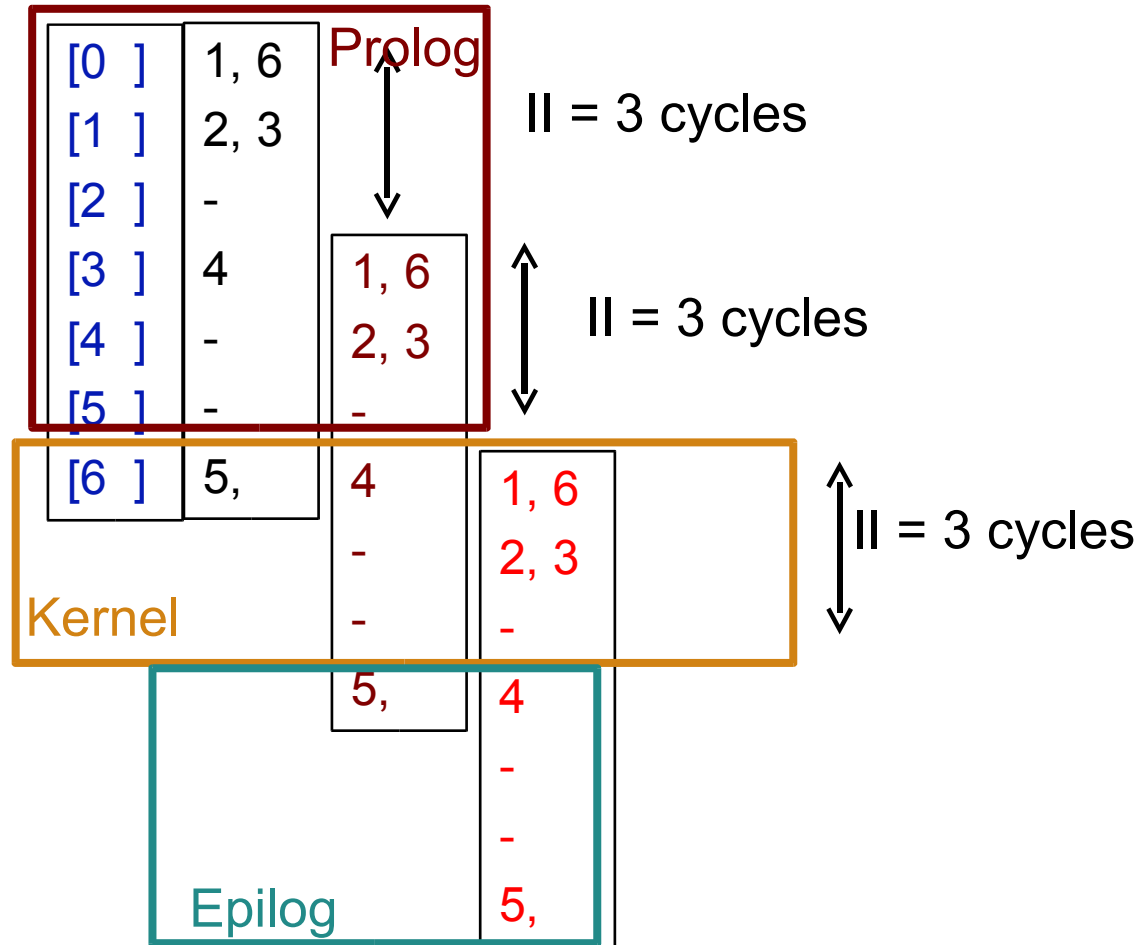


Modulo Scheduling (MS)

pre-scheduled instructions

L5:

2. slwi r0,r2,2
3. lfsx f13,r4,r0
4. lfsx f0,r3,r0
5. fadds f0,f0,f13
6. fadds f1,f1,f0
7. addi r2,r2,1
8. bdnz L5





Modulo scheduling – current status

First implementation in GCC 4.0

Improvements in GCC 4.1:

- Improved insertion of register-moves (Haifa)

 - Gives 25% speedup on an accumulation loop

- Using RTL level loop utilities (Haifa)

In the works (GCC 4.x?):

- Propagate dependencies to the RTL level (Haifa)

- Make Modulo Scheduling work with rotating registers (for itanium, China)

- Remove restrictions on loop form (for MIPS, India)



GCC Developers' Summit 2005



Registration

Schedule

Paper Presentations

Special Events

Sponsors

Travel & Hotel

Venue

FAQ

Home

Paper Presentations

Andrey Belevantsev **ISP RAS**
Improving GCC instruction scheduling for Itanium

Daniel Berlin **IBM**
Structure aliasing in GCC

Paolo Carlini **SUSE**
"The C++ library is being enhanced"

Zdenek Dvorak **SuSE**
Gimplification Improvements

Mostafa Hagog **IBM Research Lab in Haifa**
Cache Aware Data Layout Reorganization Optimization in GCC

Olivier Hainque **AdaCore**
Compile-time evaluation of stack size requirements with GCC. Motivation, Development and Experiments feedback.

Jan Hubicka **SUSE CR**
Profile driven optimizations in GCC

Janis Johnson **IBM Linux Technology Center**
Decimal Floating Point Extension for C via decNumber

Geoff Keating **Apple Computer, Inc.**
Inter-Module Analysis in GCC

Razya Ladelsky **IBM Labs**
Interprocedural constant propagation in GCC

Vladimir N Makarov **Red Hat**
Yet another GCC register allocator

Toon Moene
Gfortran: A case study compiling a 1,000,000+ line Numerical Weather Forecasting System.

David L. Moore **Intel**
C/C++ Compatibility on Linux. Requirements, Achievements and Prospects

Diego Novillo **Red Hat**
A Propagation Engine for GCC

Ulrich Weigand **IBM**
Porting the GNU Tool Chain to the Cell Architecture

Canqun Yang **National University of Defense Technology**
Performance Improvements for GCC Using Architecture Features on IA-64

F. Kenneth Zadeck **NaturalBridge, Inc**
Compilation wide alias analysis

Copyright © 2004 ~ 2005 GCCSummit.org. All rights reserved.



Haifa's contributions to GCC:

(3) Inter-Procedural constant propagation (4.1)

Razya Ladelsky, Mircea Namolaru, Revital Erez

Based on the paper Interprocedural Constant Propagation
by David Callahan, Keith D.Cooper, Ken Kennedy, Linda Toczon,
ACM SIGPLAN '86 Symposium on Compiler Construction



Inter Procedural Constant Propagation (IPCP)

Determine which formal parameters (of a method) are known to be constant in all invocations

Use method versioning in the presence of extern methods / indirect calls

Create a version with the constant parameter propagated into the method body

Update the call graph

```
f1(7);
```

```
f2(7);
```

```
f1 (int a) {  
    g (a);  
}
```

```
f2 (int b) {  
    g (7);  
}
```

```
g (int c) {  
    ...  
}
```



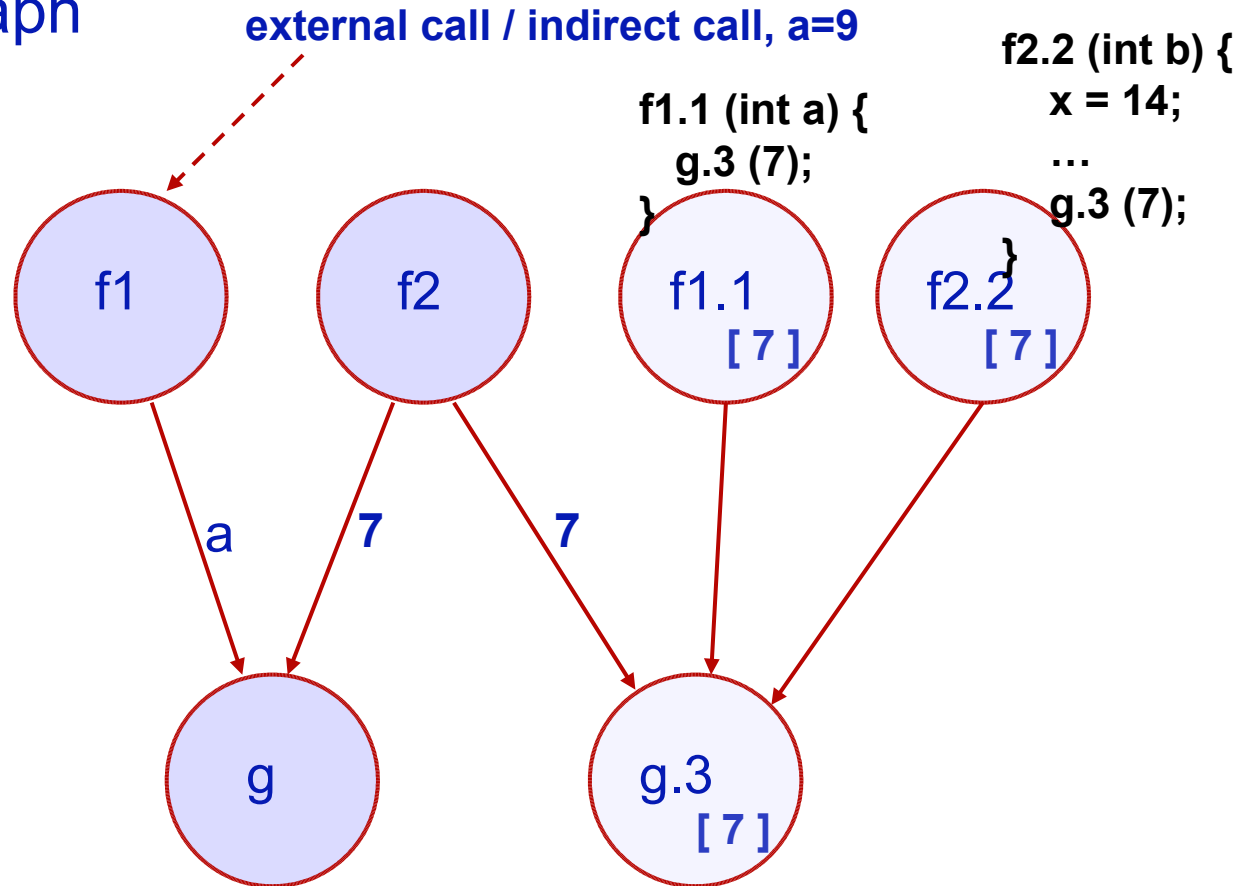
Updating the Callgraph

```
f1(7);  
f2(7);
```

```
f1 (int a) {  
  g (a);  
}
```

```
f2 (int b) {  
  x = b * 2;  
  ...  
  g (7);  
}
```

```
g (int c) {  
  ...  
}
```





Inter Procedural Constant Propagation – Status:

First implementation will be part of GCC 4.1

Versioning can serve many IPA optimizations

IPCP as an engine for propagating other formal properties

Inter Procedural alias analysis (Olga Golovanevsky)

Future Work:

Don't always version

Multiple versioning (multiple constant values)

Profile-guided



Haifa's contributions to GCC:

(4) Data Layout Optimizations (GCC 4.x?)

Mostafa Hagog, Caroline Tice (Apple)

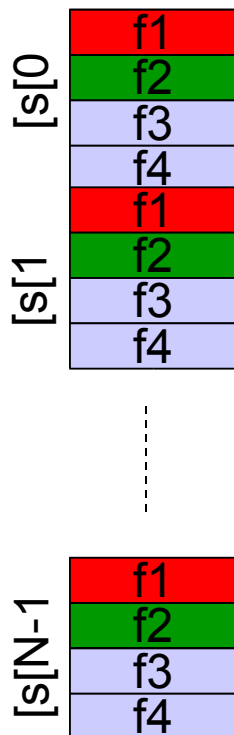


Introduction - Structure Splitting (Peeling) Example

```
struct s {  
    int f1, f2, f3, f4;  
} s[N];
```

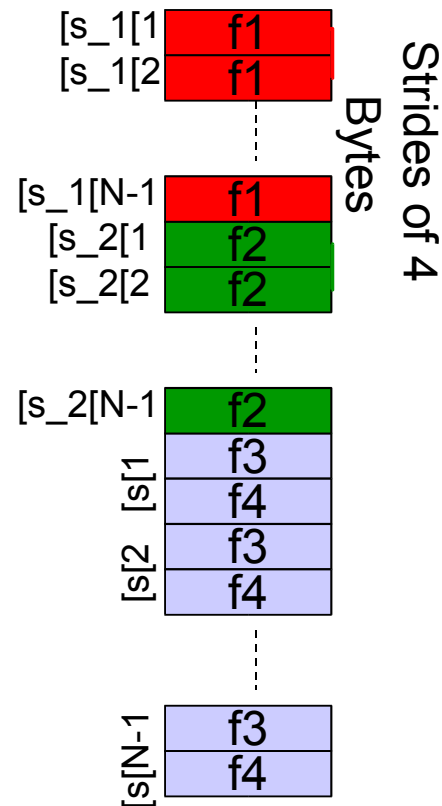
```
foo1 () {  
    for (i = 1 to N)  
        access (S_1[i].f1)  
}
```

```
foo2 () {  
    for (i = 1 to N)  
        access (S_2[i].f1)  
}
```



Strides of 16 Bytes

```
struct s_1 {  
    int f1;  
} s_1[N];  
  
struct s_2 {  
    int f2;  
} s_2[N];  
  
struct s {  
    int f3;  
    int f4;  
} s[N];
```



Bytes
Strides of 4

Data locality



A field access by any other name...

Field name	<code>foo.bar</code> <code>foo->bar</code>
Base address + offset	<code>(&foo) + 16</code>
Nearby field + offset	<code>&(foo->baz) + 8</code>
Typecast/alias	<code>dummy = (void *) foo;</code> <code>&(dummy) ...</code>
Other	<code>< your favorite C idiom here! ></code>



Data Layout Optimizations – Status:

Currently in struct-reorg branch

Soon: will start submitting patches for review



Putting Haifa on the Free-Software Map: GCC Development at IBM Research - Talk Layout



Introduction

GCC structure

Haifa's recent contributions to GCC

Vectorization (GCC 4.0)

Modulo scheduling (GCC 4.0)

Inter-Procedural constant propagation (GCC 4.1)

Data-layout optimizations (GCC 4.x ?)

Concluding Remarks



Working with the GCC community



Working with the GCC Community - Issues

It's a shock

“project management” ??

- No control, who's doing what?

- Main GCC code base is under rapid, active development

- Using an infrastructure as it is being developed

Culture shock

- Language

- Work **with** GCC community!

- Long-term maintenance of private forks very expensive

- Infrastructure changes need to be coordinated with community

- Multiplatform aspects need to be considered

- Expose your work early



Working the GCC Community - Benefits

World Wide Collaboration

Distribution of work

Testing

World Wide Exposure

The Community

Highly skilled developers



Concluding Remarks

IBM leading industrial backer of OSS movement
and a catalyst for the GCC community

Haifa and GCC:

GCC vectorizer

Developing a generic vectorizer in a multi-platform compiler

Modulo scheduling, Haifa scheduler

Inter Procedural optimizations (constant-propagation, data-layout)

GCC:

Evolving - new SSA framework, new IPA infrastructure

Open – gcc.gnu.org



References

gcc.gnu.org (including wiki pages)

www.gccsummit.org (including paper proceedings)

“Contributions to the GNU Compiler Collection GCC”, IBM’s Systems Journal, issue on Open Source, volume 44, number 2, 2005

thanks !



Backup slides



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  
    }
```

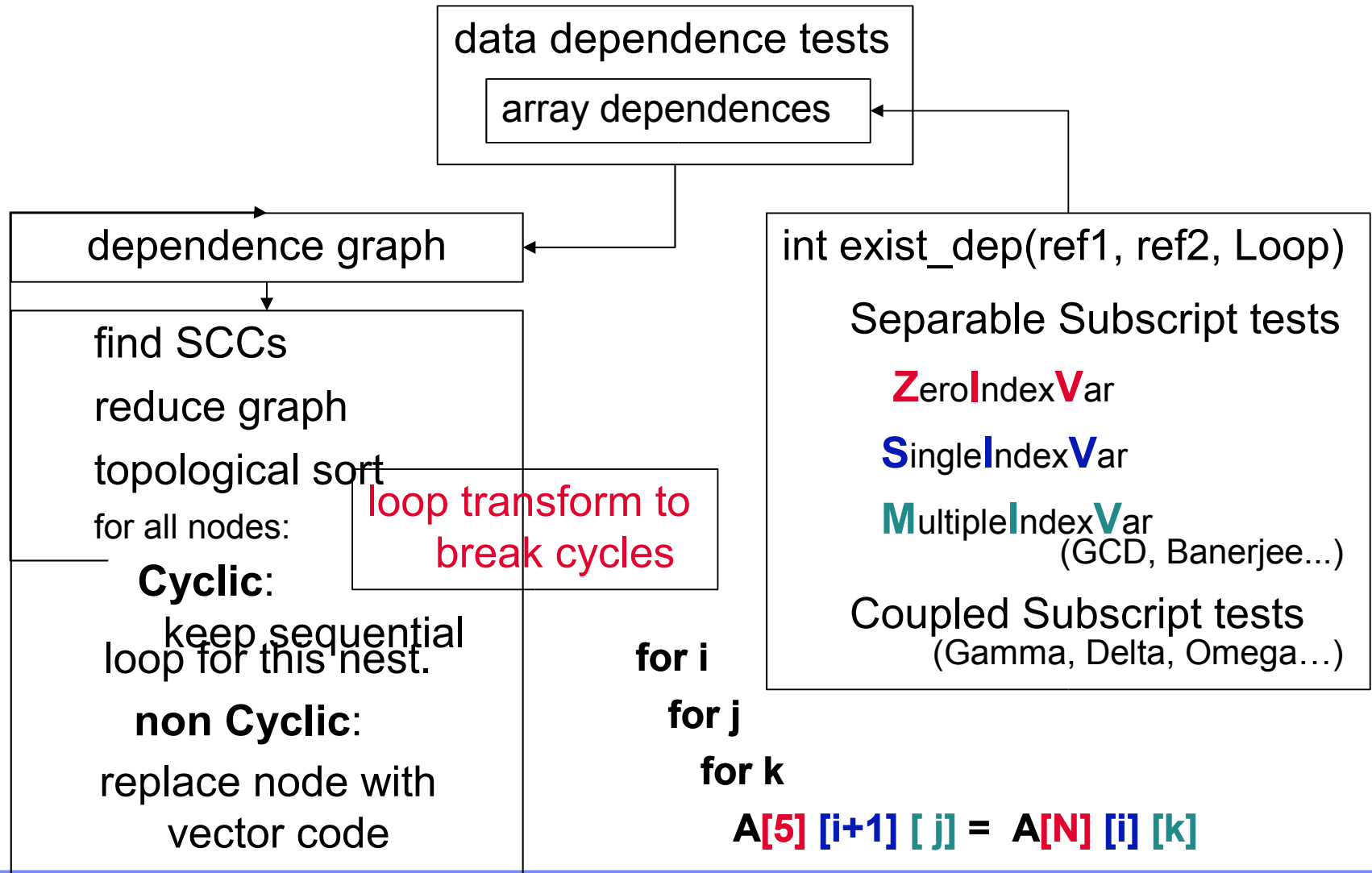
```
for (i=0; i<N; i++){  
    D[i] = A[i] + Y  
    A[i+1] = B[i] + X  
}
```

```
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





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];
```

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”

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;

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;



Handling Alignment

Alignment analysis

Transformations to force alignment

loop versioning

loop peeling

Efficient misalignment support

vector *vp_q1 = q, *vp_q2 = &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
}

```

Peeling for p[i] and versioning:

Loop versioning (for access p[i]):

```

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

```



Reduction

Cross iteration dependence

Prolog and epilog

Partial sums

s1,s2,s3,s4

0	0	0	0
---	---	---	---

0	1	2	3
---	---	---	---

4	6	8	10
---	---	---	----

+	0	1	2	3
---	---	---	---	---

+	4	5	6	7
---	---	---	---	---

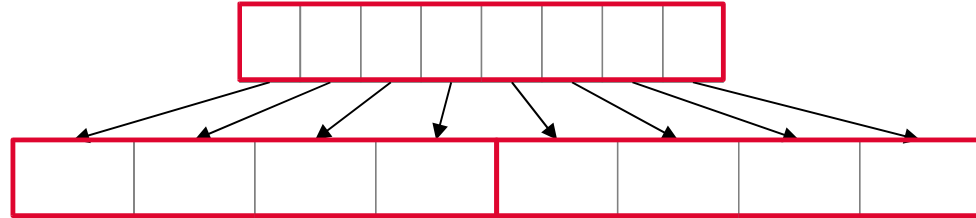
28

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



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

	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)

$A[i], i=\{0,5,10,15,\dots\}; \text{access_fn}(i) = (0,+,5)$

→ VOP(**VR5**)

$(VR1,\dots,VR4) \leftarrow \text{vload}(\text{mem})$

$\text{mask} \leftarrow (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1)$

$VR5 \leftarrow \text{pack}(VR1,\dots,VR4),\text{mask}$

VOP(VR5)

Data in Memory:

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



Stage 3: Applying the transformations

Three possible transformations

Field reordering: If not safe to split.

Struct splitting: If safe to split & not appropriate for peeling.

Struct peeling: If safe to split & using an array to access fields.



Field Reordering Example

```
struct network {  
    char    clustfile[]  
    char    inputfile[]  
    long int n;  
    long int n_trips;  
    long int primal_unbounded;  
    double  optcost;  
    long int ignore_impl;  
    node    *nodes;  
    arc     *arcs;  
    long int bound_exchanges;  
    long int checksum;  
};
```

Original Struct

```
struct network {  
    long int checksum;  
    long int bound_exchanges;  
    long int ignore_impl;  
    double  optcost;  
    long int primal_unbounded;  
    char    clustfile[]  
    char    inputfile[]  
    node    *nodes;  
    arc     *arcs;  
    long int n_trips;  
    long int n;  
};
```

Reordered Fields

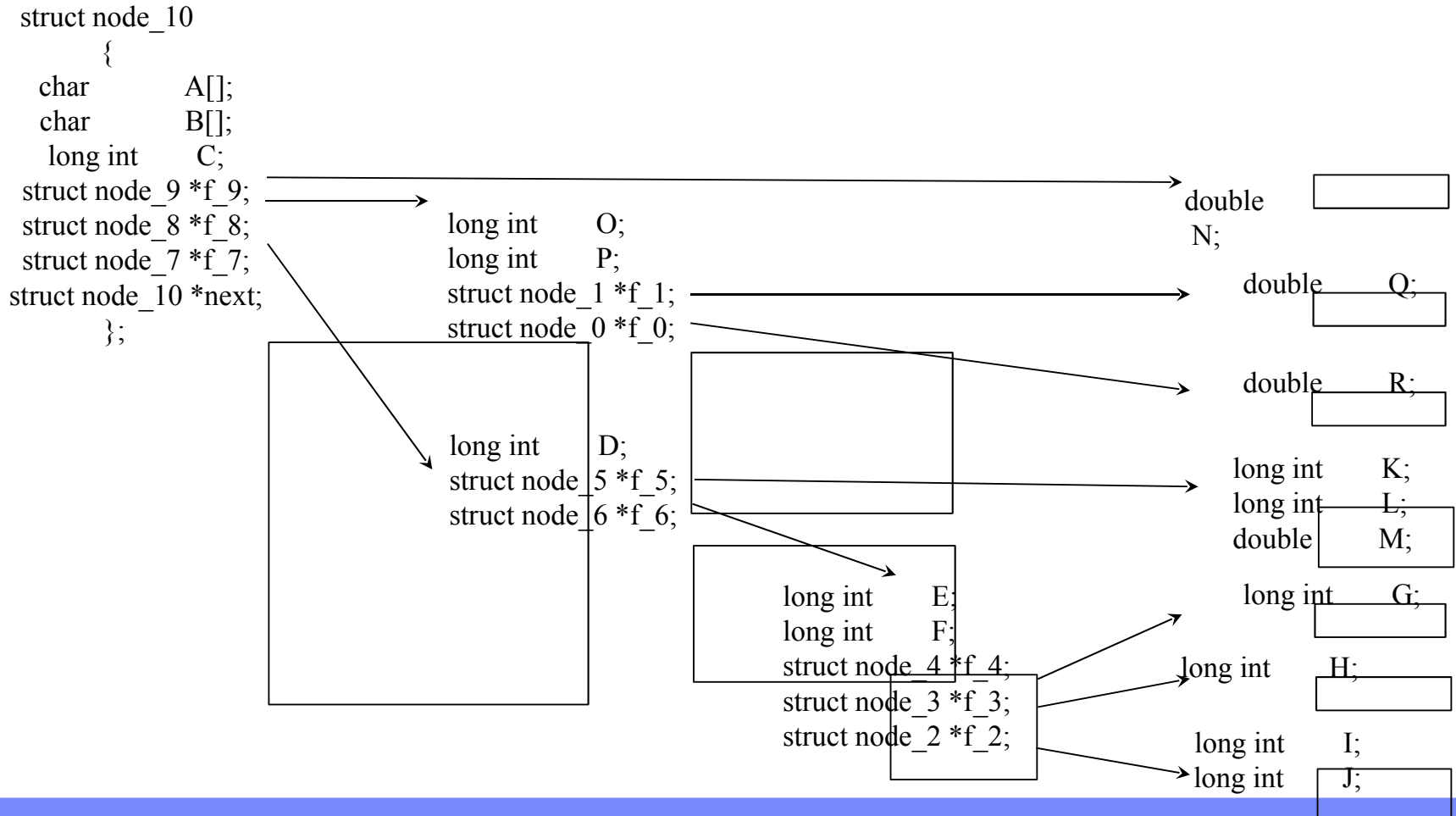


Struct Splitting Example

```
struct node
{
    char    A[];
    char    B[];
long int    C, D, E, F, G;
long int    H, I, J, K, L;
    double  M;
    double  N;
    long int O;
    long int P;
    double  Q;
    double  R;
};
```



Struct Splitting Example (cont.)





Struct Peeling Example

```
struct student
{
  char *name;
  char *address;
  int age;
  class_rec classes[7];
  char grades[7];
  double gpa;
}
```

name: "Jill"
address: "1 Main St."
age: 9
classes; ...
grades; ...
gpa: 3.14
[0]

name: "John"
address: "35 Elm St."
age: 10
classes; ...
grades; ...
gpa: 2.75
[1]

name: "Alan"
address: "78 South St."
age: 9
classes; ...
grades; ...
gpa: 3.0
[2]

```
struct student
{
  char *name;
  class_rec classes[7];
  char grades[7];
  double gpa;
}
```

name: "Jill"
classes; ...
grades; ...
gpa: 3.14
[0]

name: "John"
classes; ...
grades; ...
gpa: 2.75
[1]

name: "Alan"
classes; ...
grades; ...
gpa: 3.0
[2]

```
struct student_sub_1
{
  char *address;
  int age;
}
```

address: "1 Main St." age: 9 [0]
address: "35 Elm St." age: 10 [1]
address: "78 South St." age: 9 [2]



Basic Steps for Applying Transformations

Build new record definition(s). {R,S,P}

Update field mapping data concurrently. {S,P}

Update types for vars . {R}

Create new vars with new types. {S,P}

Replace uses of old vars with uses of new vars

Update all field accesses. {S,P}

Update array element calculations. {P}

Fix up calls to *malloc*. {S,P}

{R = reordering, S = Splitting, P = Peeling}