Loop distribution in GCC

Adding a new optimization pass at the GIMPLE SSA level in GCC

Georges-André SILBER
Centre de recherche en informatique
École nationale supérieure des mines de Paris

Georges-Andre.Silber@ensmp.fr

May 2006
Grenoble, France
Outline

- Getting and compiling GCC
- How to add an optimization pass in GCC
- Case study: loop distribution
Getting and compiling GCC
Getting GCC

- Main website: http://gcc.gnu.org
- Use Subversion: http://subversion.tigris.org
- `svn co svn://gcc.gnu.org/svn/gcc/trunk gcc`
- `svn up` in the directory `gcc` to get in sync
Welcome to the GCC home page!

GCC, the GNU Compiler Collection, includes front ends for C, C++, Objective-C, Fortran, Java, and Ada, as well as libraries for these languages (libstdc++, libgcj,...).

We strive to provide regular, high quality releases, which we want to work well on a variety of native and cross targets (including GNU/Linux), and encourage everyone to contribute changes and help testing GCC. Our sources are readily and freely available via SVN and weekly snapshots.

Major decisions about GCC are made by the steering committee, guided by the mission statement.

Current release series: GCC 4.1.0 (released 2006-02-28)
Branch status: 2006-04-16 (open for regression and documentation fixes only). Serious regressions. All known regressions.

Previous release series: GCC 4.0.3 (released 2006-03-10)
Branch status: 2006-03-10 (open for regression and documentation fixes only). Serious Regressions.

Previous release series: GCC 3.4.5 (released 2005-11-30)
Branch status: GCC 3.4.6 is the last release from the 3.4 series; the branch has been closed after the release.

Active development (mainline): will become GCC 4.2.0 (current changes)
Status: Stage 3; 2006-04-19 (open for bugfixes).

News/Announcements

March 10, 2006
GCC 4.0.3 has been released.

March 9, 2006
Richard Henderson, Jakub Jelinek and Diego Novillo of Red Hat Inc, and Dmitry Kurochkin have contributed an implementation of the OpenMP v2.5 parallel programming interface for C, C++ and Fortran.

February 28, 2006
GCC 4.1.0 has been released.

November 30, 2005
GCC 3.4.5 has been released.

October 26, 2005
GCC has moved from CVS to SVN for revision control.

September 28, 2005
GCC 4.0.2 has been released.
SVN checkout
Compiling GCC

• Create a directory, for instance ‘gcc-obj’
• This is the target of the compilation
• cd gcc-obj
• $GCCSRC/configure
• Use ‘--enable-languages=c’ for C only
• Use ‘--disable-bootstrap’ for slow machines
• make
Configure

```
chamonix:~/temp gasilber$ mkdir gcc-obj
chamonix:~/temp gasilber$ cd gcc-obj/
chamonix:~/temp/gcc-obj gasilber$ ../configure --disable-bootstrap --enable-languages=c
```

```
-bash: ../configure: No such file or directory
chamonix:~/temp/gcc-obj gasilber$ ../gcc/configure --disable-bootstrap --enable-languages=c
creating cache .config.cache
checking host system type... powerpc-apple-darwin8.6.0
checking target system type... powerpc-apple-darwin8.6.0
checking build system type... powerpc-apple-darwin8.6.0
checking for a BSD compatible install... /usr/bin/install -c
checking whether ln works... yes
checking whether ln -s works... yes
checking for gcc... gcc
checking whether the C compiler (gcc) works... yes
checking whether the C compiler (gcc) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for gnatbind... no
checking whether compiler driver understands Ada... no
checking how to compare bootstrapped objects... cmp --ignore-initial=16 $f1 $f2
checking for correct version of gmp.h... no

*** This configuration is not supported in the following subdirectories:
   target-libmudflap target-libada gnattools target-libstdc++-v3 target-libgfortran target-libgfortran-3.0
   et-libffi target-boehm-gc target-zlib target-libjava zlib target-libobjc target-libgcc-math

checking whether to enable maintainer-specific portions of Makefiles... no
updating cache .config.cache
creating .config.status
creating Makefile
chamonix:~/temp/gcc-obj gasilber$
```
Make

```
chamonix:~/Work/GCC/gcc-obj gasilber$ make CFLAGS='-g' -j2
Makefile:10935: warning: overriding commands for target `restrap'
Makefile:10273: warning: ignoring old commands for target `restrap'
Makefile:10935: warning: overriding commands for target `restrap'
Makefile:10273: warning: ignoring old commands for target `restrap'
rm -f stamp-h1
/bin/sh ./config.status config.h
make[3]: Nothing to be done for `all'.
make[3]: Nothing to be done for `all'.
make[2]: Nothing to be done for `all'.
config.status: creating config.h
make[2]: Nothing to be done for `all'.
make[2]: Nothing to be done for `all'.
config.status: config.h is unchanged
test -f config.h || (rm -f stamp-h1 && make stamp-h1)
test -d po || mkdir po
test -d po || mkdir po
  : --statistics -o po/be.gmo ../../../gcc/libcpp/po/be.po
  : --statistics -o po/ca.gmo ../../../gcc/libcpp/po/ca.po
test -d po || mkdir po
test -d po || mkdir po
  : --statistics -o po/de.gmo ../../../gcc/libcpp/po/de.po
test -d po || mkdir po
  : --statistics -o po/da.gmo ../../../gcc/libcpp/po/da.po
make all-recursive
Making all in testsuite
make[8]: Nothing to be done for `all'.
true DO=all multi-do # make
chamonix:~/Work/GCC/gcc-obj gasilber$
```
Using GCC

- Considering ‘loop.c’ is an example code
- The ‘cc1’ (cc one) executable is in ‘gcc-obj/gcc’
- `gcc-obj/gcc/cc1 -O2 -fdump-tree-ivopts loop.c`
- It generates ‘loop.s’
- Dump after ‘ivopts’ pass in ‘loop.c.086t.ivopts’
- Note: use ‘make install’ for full installation
Example

```c
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#define N 10000

int main (int argc, char const* argv[]) {
    unsigned int i;
    int a[N], b[N], c[N], d[N];
    int k;

    assert (argc > 1);
    k = atoi (argv[1]);
    a[0] = k;
    a[3] = k+1;
    c[1] = k*2;
    for (i = 2; i < (N-1); i ++) {
        a[i] = k * i;
        b[i] = a[i-2] + k;
        c[i] = b[i] + a[i+1];
        d[i] = c[i-1] + k + i;
    }
    printf ("%d %d %d %d\n", a[N-2], b[N-1], c[N-2], d[N-2]);
    return 0;
}
```
Execute `cc1`

```
chamonix:~:/Work/GCC gasilber$ gcc-obj/gcc/cc1 -02 -fdump-tree-ivopts loop.c

Analyzing compilation unit
Performing intraprocedural optimizations

Assembling functions:

main

Execution times (seconds)

<table>
<thead>
<tr>
<th>Function</th>
<th>User</th>
<th>System</th>
<th>Wall</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias analysis</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>7 KB</td>
</tr>
<tr>
<td>preprocessing</td>
<td>0.03</td>
<td>0.00</td>
<td>0.04</td>
<td>104 KB</td>
</tr>
<tr>
<td>lexical analysis</td>
<td>0.00</td>
<td>0.06</td>
<td>0.05</td>
<td>0 KB</td>
</tr>
<tr>
<td>parser</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>279 KB</td>
</tr>
<tr>
<td>tree VRP</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>9 KB</td>
</tr>
<tr>
<td>tree SSA incremental</td>
<td>0.01</td>
<td>0.00</td>
<td>0.01</td>
<td>0 KB</td>
</tr>
<tr>
<td>complete unrolling</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0 KB</td>
</tr>
<tr>
<td>tree STMT verifier</td>
<td>0.02</td>
<td>0.00</td>
<td>0.02</td>
<td>20 KB</td>
</tr>
<tr>
<td>expand</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>3 KB</td>
</tr>
<tr>
<td>varconst</td>
<td>0.01</td>
<td>0.00</td>
<td>0.02</td>
<td>3 KB</td>
</tr>
<tr>
<td>global CSE</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>3 KB</td>
</tr>
<tr>
<td>scheduling</td>
<td>0.00</td>
<td>0.04</td>
<td>0.04</td>
<td>3 KB</td>
</tr>
</tbody>
</table>
| TOTAL         | 0.11 | 0.10   | 0.26  | 1315 KB

Extra diagnostic checks enabled; compiler may run slowly.
Configure with `--disable-checking` to disable checks.
```
Assembly code

.chasm ppc
cstring
.align 2

LC0:
  .ascii "%s:%u: failed assertion \%s\"12\""
  .align 2

LC1:
  .ascii "loop.c\""
  .align 2

LC2:
  .ascii "argc > 1\""
  .align 2

LC3:
  .ascii "%d %d %d %d\"12\""
.text
  .align 2
  .globl _main

_main:
  mfllr r0
  stw r0,8(r1)
  lis r0,0xffffd
  ori r0,r0,36544
  lwz r0,8(r1)
  mtlr r0
  blr
  .subsections_via_symbols

chamonix:~/Work/GCC gasilber$
GIMPLE dump

GIMPLE dump

i_62 = ivtmp.61_8;
k.26_32 = pretmp.49_43;
D.2627_31 = (int) ivtmp.65_60;
D.2627_34 = D.2627_31;
D.2704_36 = (int *) ivtmp.62_4;
MEM[base: D.2704_36] = D.2627_34;
D.2705_14 = (int *) ivtmp.62_4;
D.2629_38 = MEM[base: D.2705_14, offset: 42949672888];
D.2631_39 = D.2629_38 + k_13;
D.2706_67 = (int *) ivtmp.61_8;
D.2707_68 = D.2706_67 * 4B;
MEM[base: &b[0], index: D.2707_68] = D.2631_39;
D.2708_69 = (int *) ivtmp.62_4;
D.2633_45 = MEM[base: D.2708_69, offset: 4B];
D.2634_46 = D.2631_39 + D.2633_45;
D.2709_70 = (int *) ivtmp.71_56;
MEM[base: D.2709_70] = D.2634_46;
D.2710_71 = (int *) ivtmp.71_56;
D.2636_50 = MEM[base: D.2710_71, offset: 42949672928];
D.2637_51 = D.2636_50 + k_13;
D.2638_52 = (unsigned int) D.2637_51;
Debugging GCC

A lot of people frequently have questions about debugging GCC. In particular, how to debug the compiler itself, instead of the driver.

Here is a quick rundown:

Assuming you've produced preprocessed source (see the bug reporting directions for how to do this), and have the compiler built somewhere, you can simply do

```
gdb --args <location of cc1, cc1plus, or whatever compiler>
for the language the preprocessed source file is in> <flags passed to compiler>
```

This will enable you to debug the compiler itself, instead of the driver.

You can also use the driver's `-###` option which writes the commands that the driver would execute. For example,

```
gdb --args $(./xgcc -### <parameters to the driver> 2>&1 | fgrep cc1)
```

There are scripts that automate all of this for you [here](#) that make debugging a front-end much simpler.

While stepping through a front-end within a debugger, you can use the `debug_tree()` and `debug_rtx()` functions to print out the structure of a tree node or RTL expression respectively.

GCC itself is normally compiled at `-O2` which makes stepping through code a bit difficult. You should use GDB 6.3 (or a newer version), which can work properly with location lists generated by newer GCCs that help in debugging in such cases. Another useful trick is to only compile the particular module you are interested in at a lower optimisation level. For example, if you are debugging `parse.y` in the Java front-end, you can use:

```
$ touch $GCC_SRC_DIR/gcc/java/parse.y
$ make BOOT_CFLAGS=-00 -g3 bubblestrap
```

If you use GDB to debug GCC and you run the debugger from within the `$GCC_BUILD_DIR/gcc` folder, you get to automatically use the `.gdbinit` file created there by the build process. It defines a few handy macros to help debug GCC. See the file `$GCC_SRC_DIR/gcc/gdbinit.in` for details.

Randomization

You may want to read up on Randomization and disable it if you would like reproducible results.
How to add an optimization pass in GCC
Adding a pass: checklist

✓ New pass in file ‘gcc/gcc/mynewpass.c’
✓ Edit ‘gcc/gcc/passes.c’ (new pass)
✓ Edit ‘gcc/gcc/tree-flow.h’ (prototype)
✓ Edit ‘gcc/gcc/tree-pass.h’ (pass prototype)
✓ Edit ‘gcc/gcc/common.opt’ (new option)
✓ Edit ‘gcc/gcc/doc/invoke.texi’ (doc)
✓ Edit ‘gcc/gcc/timevar.def’ (timing)
✓ Edit ‘gcc/gcc/Makefile.in’ (new target)
New pass file

- New C file in gcc/gcc
- Name: tree-loop-distribution.c
- Pass gate
- Pass function
- Pass structure describing the pass
- The pass is executed for each function
Pass function and gate

```c
static unsigned int
tree_loop_distribution (void)
{
    fprintf (stderr, "Hello %x\n", current_loops);
    return 0;
}

static bool
gate_tree_loop_distribution (void)
{
    return flag_tree_loop_distribution != 0;
}
```
Pass structure

```c
struct tree_opt_pass pass_loop_distribution =
{
    "ldist", /* name */
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution, /* execute */
    NULL, /* sub */
    NULL, /* next */
    0, /* static_pass_number */
    TV_TREE_LOOP_DISTRIBUTION, /* tv_id */
    PROP_cfg | PROP_ssa, /* properties_required */
    0, /* properties_provided */
    0, /* properties_destroyed */
    0, /* todo_flags_start */
    TODO_dump_func | TODO_verify_loops, /* todo_flags_finish */
    0 /* letter */
};
```
Terse name

const char *name;

Terse name of the pass used as a fragment of the dump file name.
Gate function

```c
struct tree_opt_pass pass_loop_distribution = {
    "ldist",          /* name */
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution,    /* execute */
    NULL,        /* options */
    NULL,        /* id */
    0,           /* prop */
    TV_TREE_LOOP_DIST,  /* is_root */
    PROP_cfg | PROP_s,
    0,           /* prop */
    0,
    0,
    TODO_dump_func | 0
};
```

```c
bool (*gate) (void)
```

If non-null, this pass and all sub-passes are executed only if the function returns true.
Function for execution

```c
struct tree_opt_pass pass_loop_distribution = {
    "ldist",
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution,         /* execute */
    NULL,
    NULL,
    0,
    TV_TREE_LOOP_DISTRIBUTION,
    PROP_cfg | PROP_ssd,
    0,
    0,
    0,
    TODO_dump_func | TODO_dump_ssd,
    0
};
```

`unsigned int (*execute) (void)`

This is the code to run. If null, then there should be sub-passes otherwise this pass does nothing.
Hierarchy of passes

Passes are chained and can have sub-passes.

```c
struct tree_opt_pass pass_loop_distribution =
{
    "ldist",          /* name */
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution,     /* execute */
    NULL,                /* sub */
    NULL,                /* next */
    0,                   /* static pass number */
    TV_TREE_LOOP_DIST,   /* prop */
    PROP_CFG | PROP_PASS
};
```
Pass number

```c
struct tree_opt_pass pass_loop_distribution = {
    "ldist",
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution, /* execute */
    NULL, /* sub */
    NULL, /* next */
    0, /* static_pass_number */
    TV_TREE_LOOP_DISTRIBUTION, /* tv_id */
    PROP_cfg | PROP_addr, /* properties required */
    0,
    0,
    0,
    TODO_dump_func,
    0
};
```

`int static_pass_number;` Used as a fragment of the dump file name.
Variable for timing

```c
struct tree_opt_pass pass_loop_distribution = {
    "ldist",
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution, /* execute */
    NULL, /* sub */
    NULL, /* next */
    TV_TREE_LOOP_DISTRIBUTION, /* tv_id */
    PROP_cfg | PROP_ssa, /* properties_required */
    0,
    0,
    0,
    0,
    TODO_dump_func | TODO,
};
```

```c
unsigned int tv_id;
```

The timevar id associated with this pass.
Properties

Passes can require, provide and/or destroy some properties.
Things to do...

```c
struct tree_opt_pass pass_loop_distribution =
{
    "ldist",               /* name */
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution,   /* execute */
    NULL,
    NULL,                     /* properties_destroyed */
    0,
    TV_TREE_LOOP_DISTRIBUTION, /* todo_flags_start */
    PROP_cfg | PROP_ssc |
    0,
    0,                        /* todo_flags_finish */
    TODO_dump_func | TODO_verify_loops, /* letter */
};
```

unsigned int ...;

Things to do before and after the pass.
Letter for RTL dumps

```c
struct tree_opt_pass pass_loop_distribution =
{
    "ldist",
    gate_tree_loop_distribution, /* gate */
    tree_loop_distribution, /* execute */
    NULL, /* sub */
    TV.Tree_LOOP, /* next */
    0,
    NULL,
    TV.Tree_LOOP, /* name */
    PROP_cfg | PROP_tree,
    0,
    0,
    0,
    0,
    0,
    /* todoFlags_finish */
};
```

char letter;

Letter used for RTL dumps.
GCC file: passes.c

Adding the pass in the pass hierarchy

```c
init_optimization_passes()
```

```c
p = &pass_tree_loop.sub;
NEXT_PASS (pass_tree_loop_init);
NEXT_PASS (pass_copy_prop);
NEXT_PASS (pass_lim);
NEXT_PASS (pass_tree_unswitch);
NEXT_PASS (pass_scev_cprop);
NEXT_PASS (pass_empty_loop);
NEXT_PASS (pass_record_bounds);
NEXT_PASS (pass_linear_transform);
NEXT_PASS (pass_loop_distribution);
NEXT_PASS (pass_iv_canon);
NEXT_PASS (pass_if_conversion);
NEXT_PASS (pass_vectorize);
NEXT_PASS (pass_complete_unroll);
NEXT_PASS (pass_loop_prefetch);
NEXT_PASS (pass_iv_optimize);
NEXT_PASS (pass_tree_loop_done);
*p = NULL;
```
tree-flow.h / tree-pass.h

Prototypes for pass function and structure.
common.opt

Command line option and internal flag.

```plaintext
ftree-loop-linear
Common Report Var(flag_tree_loop_linear)
Enable linear loop transforms on trees

ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution)
Enable loop distribution on trees

ftree-loop-ivcanon
Common Report Var(flag_tree_loop_ivcanon) Init(1)
Create canonical induction variables in loops

ftree-loop-optimize
Common Report Var(flag_tree_loop_optimize) Init(1)
Enable loop optimizations on tree level
```
Documenting the pass for the GCC manual.
Timing: timevar.def

Variable used for timing and identification in the timing report.
Testsuite Conventions

Every language or library feature, whether standard or a GNU extension, and every warning GCC can give, should have testcases thoroughly covering both its specification and its implementation. Every bug fixed should have a testcase to detect if the bug recurs.

The testsuite READMEs discuss the requirement to use `abort()` for runtime failures and `exit(0)` for success. For compile-time tests, a trick taken from autoconf may be used to evaluate expressions: a declaration `extern char x[(EXPR) ? 1 : -1];` will compile successfully if and only if `EXPR` is nonzero.

Where appropriate, testsuite entries should include comments giving their origin: the people who added them or submitted the bug report they relate to, possibly with a reference to a PR in our bug tracking system. There are some copyright guidelines on what can be included in the testsuite.

If a testcase itself is incorrect, but there's a possibility that an improved testcase might fail on some platform where the incorrect testcase passed, the old testcase should be removed and a new testcase (with a different name) should be added. This helps automated regression-checkers distinguish a true regression from an improvement to the test suite.
Test

gcc-obj/gcc/cc1 -O -ftree-loop-distribution -fdump-tree-ldist loop.c
Preparing a patch

- In `gcc/gcc` issue a `svn diff > mypatch`
- Edit the patch to add a ‘Changelog’
- Apply: `patch -p0 < mypatch`
Configure an external diff utility

Our patch guidelines suggest that patches be submitted using the `-p` option to get function names printed into the context surrounding changes. Subversion's internal diff library does not support `-p`, so doing this requires configuring Subversion to use an external diff utility. To configure Subversion to use an external diff utility, create a file containing the diff command, and mark it as executable.

An example:

```bash
#!/bin/bash
diff=/usr/bin/diff
args="-up"

exec ${diff} ${args} "@"
```

Then edit `~/.subversion/config`, and specify this script as your diff command. Other information can be found in the tricks page.
tree-loop-distribution.c: New.
doc/invoke.texi: Add new option -ftree-loop-distribution.
tree-pass.h (pass_loop_distribution): Declare.
timevar.def (TV_TREE_LOOP_DISTRIBUTION): New.
tree-flow.h (distribute_loops): Declared.
Makefile.in (tree-loop-distribution.o): New target.
passes.c (init_optimization_passes): Add new pass pass_loop_distribution.

Index: doc/invoke.texi
--- doc/invoke.texi (revision 113325)
+++ doc/invoke.texi (working copy)
@@ -341,7 +341,7 @@
    -fsplit-ivs-in-unroller -funswitch-loops @gol
    -fvariable-expansion-in-unroller @gol
    -ftree-pre -ftree-ccp -ftree-dce -ftree-loop-optimize @gol
---ftree-loop-linear -ftree-loop-im -ftree-loop-ivcanon -fivopts @gol
+ftree-loop-linear -ftree-loop-distribution -ftree-loop-im -ftree-loop-ivcanon -fivops @gol
   -ftree-dominator-opts -ftree-dse -ftree-copyrename -ftree-sink @gol
   -ftree-ch -ftree-sra -ftree-ter -ftree-lrs -ftree-fre -ftree-vectorize @gol
   -ftree-vec-loop-version -ftree-salias -fipa-pta -fweb @gol
@@ -5090,6 +5090,11 @@

Perform linear loop transformations on tree. This flag can improve cache
performance and allow further loop optimizations to take place.
Case study: loop distribution

```
DO i=2,N
  S1
  ENDDO
  S2
  DO i=2,N
    S2
    ENDDO
  S1
  ENDDO
```

```
DO i=2,N
  S1
  ENDDO
  S2
  DO i=2,N
    S2
    ENDDO
  S1
  ENDDO
```

```
DO i=2,N
  S1
  ENDDO
  S2
  DO i=2,N
    S2
    ENDDO
  S1
  ENDDO
```
Why loop distribution?

• Typical pass in compiling technology
  • Especially for the source-to-source community
• Can increase parallelism and cache hits
• (Can decrease performance and cache hits)
• Goal: help the vectorizer of GCC
How to distribute?

Algorithm by Allen, Callahan, and Kennedy (simplified version for a loop nest of depth one)

- Build a data dependence graph with levels
- Find the Strongly Connected Components
- Rewrite the new loops according to a topological sort of the SCCs
- Produces the maximal number of parallel loops for a data dependence graph with levels
How in GCC?

- Use of existing GCC infrastructure
  - Loops + Dependences + SSA graph + GIMPLE
- New algorithms and data structures in GCC
  - Data dependence graph + SCC computation
- Manipulating GCC trees for code generation
  - Distributed loops
Example 1: C code

```c
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#define N 10000

int main (int argc, char const* argv[])
{
    unsigned int i;
    int a[N], b[N], c[N], k, z;
    assert (argc > 1);
    k = atoi (argv[1]); a[0] = k;
    for (i = 1; i < N; i++)
    {
        a[i] = c[i];
        b[i] = a[i-1]+1;
    }
    printf ("%d %d\n", a[N-1], b[N-1]);
    return 0;
}
```
Example 1: GIMPLE dump

```c
# i_19 = PHI \langle i\_31(6), 1(4)\rangle;

<\text{L3}>:;
 D.2623_24 = c[i_19];
 a[i_19] = D.2623_24;
 D.2624_27 = i_19 - 1;
 D.2625_28 = a[D.2624_27];
 D.2626_29 = D.2625_28 + 1;
 b[i_19] = D.2626_29;
 i_31 = i_19 + 1;
 if (i_31 \leq 9999) goto <\text{L9}>; else goto <\text{L5}>;

<\text{L9}>:;
 goto \langle bb\ 5\rangle\ (<\text{L3}>);

<\text{L5}>:;
 D.2627_17 = a[9999];
 D.2628_18 = b[9999];
 printf ("\%d\ %d\n"[0], D.2627_17, D.2628_18);
 return 0;
```

Example 1: RDG

S0 : D.2623_24 = c[i_19]

S1 : a[i_19] = D.2623_24

S2 : D.2624_27 = i_19 - 1

S3 : D.2625_28 = a[D.2624_27]

S4 : D.2626_29 = D.2625_28 + 1

S5 : b[i_19] = D.2626_29

S6 : i_31 = i_19 + 1
Example 1: RDG

Partition 1

S0 : D.2623_24 = c[i_19]
S1 : a[i_19] = D.2623_24
S2 : D.2624_27 = i_19 - 1
S3 : D.2625_28 = a[D.2624_27]
S4 : D.2626_29 = D.2625_28 + 1
S5 : b[i_19] = D.2626_29
S6 : i_31 = i_19 + 1
S0 : D.2623_24 = c[i_19]
S1 : a[i_19] = D.2623_24
S2 : D.2624_27 = i_19 - 1
S3 : D.2625_28 = a[D.2624_27]
S4 : D.2626_29 = D.2625_28 + 1
S5 : b[i_19] = D.2626_29
S6 : i_31 = i_19 + 1

Example 1: RDG
Example 1: partition graph

P1(0): S0; S1; S6;

P1(0)

f: 1

P2(0): S2; S3; S4; S5; S6;

P2(0)
Example 1: SCC graph

Strongly Connected Components

P1(1): S0; S1; S6;  P1(1)

P2(2): S2; S3; S4; S5; S6;  P2(2)

Two parallel loops
Example 2: C code

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define N 100000
int main (int argc, char const* argv[])
{
    unsigned int i;
    int a[N], b[N], c[N], d[N];
    int k, z;
    assert (argc > 1);
    k = atoi (argv[1]); a[0] = k; a[3] = k * 2;
    for (i = 2; i < (N-1); i++) {
        a[i] = k * i;
        b[i] = a[i-2] + k;
        c[i] = b[i-1] + a[i+1];
        d[i] = c[i-1] + b[i+1] + k + i;
    }
    printf (“%d %d %d %d
”, a[N-2], b[N-1], c[N-2], d[N-2]);
    return 0;
}
```
S0: \( k_{26,30} = (\text{unsigned int}) k_{13} \)

S1: \( D_{26,31} = i_{34} \times k_{26,30} \)

S2: \( D_{26,32} = \text{(int)} D_{26,31} \)

S3: \( a[i_{34}] = D_{26,32} \)

S4: \( D_{26,35} = i_{34} - 2 \)

S5: \( D_{26,36} = a[D_{26,35}] \)

S6: \( D_{26,37} = D_{26,36} + k_{13} \)

S7: \( b[i_{34}] = D_{26,37} \)

S8: \( D_{26,40} = i_{34} - 1 \)

S9: \( D_{26,41} = b[D_{26,40}] \)

S10: \( i_{42} = i_{34} + 1 \)

S11: \( D_{26,43} = a[i_{42}] \)

S12: \( D_{26,44} = D_{26,43} + D_{26,41} \)

S13: \( c[i_{34}] = D_{26,44} \)

S14: \( D_{26,48} = c[D_{26,40}] \)

S15: \( D_{26,50} = b[i_{42}] \)

S16: \( D_{26,51} = D_{26,48} + D_{26,50} \)

S17: \( D_{26,52} = D_{26,51} + k_{13} \)

S18: \( D_{26,53} = (\text{unsigned int}) D_{26,52} \)

S19: \( D_{26,54} = D_{26,53} + i_{34} \)

S20: \( D_{26,55} = \text{(int)} D_{26,54} \)

S21: \( d[i_{34}] = D_{26,55} \)
Example 2: RDG (2/3)
Example 2: RDG (3/3)
Exemple 2: RDG
S0 : k.26_30 = (unsigned int) k_13
S1 : D.2626_31 = i_34 * k.26_30
S2 : D.2627_32 = (int) D.2626_31
S3 : a[i_34] = D.2627_32
S4 : D.2628_35 = i_34 - 2
S5 : D.2629_36 = a[D.2628_35]
S6 : D.2630_37 = D.2629_36 + k_13
S7 : b[i_34] = D.2630_37
S8 : D.2631_40 = i_34 - 1
S9 : D.2632_41 = b[D.2631_40]
S10 : i_42 = i_34 + 1
S11 : D.2634_43 = a[i_42]
S12 : D.2635_44 = D.2634_43 + D.2632_41
S13 : c[i_34] = D.2635_44
S14 : D.2636_48 = c[D.2631_40]
S15 : D.2637_50 = b[i_42]
S16 : D.2638_51 = D.2636_48 + D.2637_50
S17 : D.2639_52 = D.2638_51 + k_13
S18 : D.2640_53 = (unsigned int) D.2639_52
S19 : D.2641_54 = D.2640_53 + i_34
S20 : D.2642_55 = (int) D.2641_54
S21 : d[i_34] = D.2642_55
Exemple 2: RDG

Partition 1

Partition 2
S0 : k.26_30 = (unsigned int) k_13
S1 : D.2626_31 = i_34 * k.26_30
S2 : D.2627_32 = (int) D.2626_31
S3 : a[i_34] = D.2627_32
S4 : D.2628_35 = i_34 - 2
S5 : D.2629_36 = a[D.2628_35]
S6 : D.2630_37 = D.2629_36 + k_13
S7 : b[i_34] = D.2630_37
S8 : D.2631_40 = i_34 - 1
S9 : D.2632_41 = b[D.2631_40]
S10 : i_42 = i_34 + 1
S11 : D.2633_43 = a[i_42]
S12 : D.2634_43 = a[i_42]
S13 : D.2635_44 = D.2634_43 + D.2632_41
S14 : D.2636_48 = c[D.2631_40]
S15 : D.2637_50 = b[i_42]
S16 : D.2638_51 = D.2636_48 + D.2637_50
S17 : D.2639_52 = D.2638_51 + k_13
S18 : D.2640_53 = (int) D.2639_52
S19 : D.2641_54 = D.2640_53 + i_34
S20 : D.2642_55 = (int) D.2641_54
S21 : d[i_34] = D.2642_55

Example 2: RDG
Exemple 2: RDG

Scalars can be recomputed.

Partition 1

Partition 2

Partition 3

Partition 4
Example 2: partition graph

P1(0): S0; S1; S2; S3;

P2(0): S4; S5; S6; S7;

P3(0): S8; S9; S10; S11; S12; S13;

P4(0): S8; S10; S14; S15; S16; S17; S18; S19; S20; S21;
Example 2: SCC graph

One sequential loop
Example 3: C code

```c
int main (int argc, char const* argv[])
{
    unsigned int i;
    int a[N], b[N], c[N], d[N];
    int k;
    assert (argc > 1);
    k = atoi (argv[1]);
    a[0] = k; a[3] = k*2;
    c[1] = k+1;
    for (i = 2; i < (N-1); i ++)
    {
        a[i] = k * i; /* S1 */
        b[i] = a[i-2] + k; /* S2 */
        c[i] = b[i] + a[i+1]; /* S3 */
        d[i] = c[i-1] + k + i; /* S4 */
    }
    printf ("%d %d %d %d\n", a[N-2], b[N-1], c[N-2], d[N-2]);
    return 0;
}
```
Exemple 3: RDG
Exemple 3: partition graph

P1(0): S₀; S₁; S₂; S₃;

P₂(0): S₄; S₅; S₆; S₇; S₈; S₉; S₁₀; S₁₁;

P₃(0): S₁₂; S₁₃; S₁₄; S₁₅; S₁₆; S₁₇; S₁₈;
Exemple 3: SCC graph

One sequential loop
One parallel loop
Sketch of the implementation

- The code presented here is simplified
- This is not part of GCC now
- The complete patch can be found on:
  - http://www.hipeac.net
tree\_loop\_distribution ()

tree\_loop\_distribution is called for each function

current\_loops contains the loops of the function
Building dependence graphs

```c
static void
do_distribution (struct loop *loop_nest)
{
    struct rdg *rdg;
    struct rdg_partition *rdgp, *sccg;

    rdg = build_rdg (loop_nest);
    if (dump_file)
        dump_rdg (dump_file, rdg);

    rdgp = build_rdg_partition (rdg);
    if (dump_file)
        dump_rdg_partition (dump_file, rdgp);

    sccg = build_scc_graph (rdgp);
    if (dump_file)
        dump_rdg_partition (dump_file, sccg);
}
```
Loop structure of GCC

See gcc/gcc/cfgloop.h
RDG computation

- One vertex per statement of the loop body
- One edge for each scalar dependence (SSA)
- One edge for each data dependence (SCEV)
static struct rdg *
build_rdg (struct loop *loop_nest)
{
    ...
    /* Check whether a RDG can be build for this loop nest or not */
    if (!loop_is_good_p (loop_nest))
        return NULL;

    /* Compute array data dependence relations */
    dependence_relations = VEC_alloc (ddr_p, heap, RDG_VS * RDG_VS);
    datarefs = VEC_alloc (data_reference_p, heap, RDG_VS);
    compute_data_dependences_for_loop (loop_nest,
        false,
        &datarefs,
        &dependence_relations);

    /* Check if all the array dependences are known (computable) */
    if (!known_dependences_p (dependence_relations))
        return NULL;

VEC_alloc () : gcc/gcc/vec.h
compute_data_dependences_for_loop () : gcc/gcc/tree-data-ref.c
loop_is_good_p ()

```c
static bool loop_is_good_p (struct loop *loop_nest)
{
    if (!loop_nest)
        return false;
    else if (loop_nest->inner)
        return false;
    else if (!loop_nest->single_exit)
        return false;
    else if (!get_loop_exit_condition (loop_nest))
        return false;
    else if (loop_nest->num_nodes != 2)
        return false;
    else if (number_of_phi_nodes (loop_nest->header) > 1)
        return false;
    else if (!check_statements (loop_nest))
        return false;
    ...
    return true;
}
```
number_of_phi_nodes ()

phi_nodes () : gcc/gcc/tree-flow-inline.h
PHI_CHAIN () : gcc/gcc/tree.h
is_gimple_reg () : gcc/gcc/tree-gimple.c
check_statements ()

```c
static bool
check_statements (struct loop *loop_nest)
{
    basic_block *bbs;
    basic_block bb;
    unsigned int i;
    block_stmt_iterator bsi;
    bbs = get_loop_body (loop_nest);
    for (i = 0; i < loop_nest->num_nodes; i++)
    {
        bb = bbs[i];
        for (bsi = bsi_start (bb); !bsi_end_p (bsi); bsi_next (&bsi))
        {
            tree stmt = bsi_stmt (bsi);
            if (TREE_CODE (stmt) == MODIFY_EXPR)
                if (!correct_modify_expr_p (stmt))
                    return false;
        }
    }
    free (bbs);
    return true;
}
```
correct_modify_expr_p ()

```c
static bool correct_modify_expr_p (tree stmt)
{
  tree lhs;

  if (TREE_CODE (stmt) != MODIFY_EXPR)
    return false;

  lhs = TREE_OPERAND (stmt, 0);

  switch (TREE_CODE (lhs))
  {
    case SSA_NAME:
    case ARRAY_REF:
    case INDIRECT_REF:
      return true;
    default:
      return false;
  }
}
```
/* OK, now we know that we can build our Reduced Dependence Graph where each vertex is a statement and where each edge is a data dependence between two references in statements. */

rdg = XNEW (struct rdg);
rdg->loop_nest = loop_nest;
rdg->loop_exit_condition = get_loop_exit_condition (loop_nest);
rdg->loop_index = get_loop_index (loop_nest);
rdg->loop_index_update = SSA_NAME_DEF_STMT (rdg->loop_index);
rdg->loop_index_phi_node = get_index_phi_node (loop_nest);

rdg->dependence_relations = dependence_relations;
rdg->datarefs = datarefs;
create_vertices (rdg);
create_edges (rdg);
```c
static tree
get_loop_index (struct loop *loop_nest)
{
    tree expr = get_loop_exit_condition (loop_nest);
    tree ivarop;
    tree test;

    if (expr == NULL_TREE)
        return NULL_TREE;
    if (TREE_CODE (expr) != COND_EXPR)
        return NULL_TREE;
    test = TREE_OPERAND (expr, 0);
    if (!COMPARISON_CLASS_P (test))
        return NULL_TREE;

    if (expr_invariant_in_loop_p (loop_nest, TREE_OPERAND (test, 0)))
        ivarop = TREE_OPERAND (test, 1);
    else if (expr_invariant_in_loop_p (loop_nest, TREE_OPERAND (test, 1)))
        ivarop = TREE_OPERAND (test, 0);
    else
        return NULL_TREE;
    if (TREE_CODE (ivarop) != SSA_NAME)
        return NULL_TREE;
    return ivarop;
}
```
static int
number_of_lvalue_immediate_uses (struct rdg *rdg, tree stmt)
{
  tree lhs;

  lhs = TREE_OPERAND (stmt, 0);
  if (TREE_CODE (lhs) == SSA_NAME)
  {
    use_operand_p imm_use_p;
    imm_use_iterator iterator;
    int n = 0;

    FOR_EACH_IMM_USE_FAST (imm_use_p, iterator, lhs)
      if (find_vertex_with_stmt (rdg, USE_STMT (imm_use_p)))
        n++;

    return n;
  }

  return 0;
}
static void
dump_rdg (FILE *outf, struct rdg *rdg)
{
    unsigned int i;
    rdg_vertex_p vertex;

    fprintf (outf, "<graphviz><![CDATA[\n");
    fprintf (outf, "digraph ");
    print_generic_expr (outf, rdg->loop_index, \0);
    fprintf (outf, " {\n");
    for (i = \0; i < rdg->nb_vertices; i++)
    {
        fprintf (outf, " v%d [ label = \"", rdg->vertices[i].number);
        fprintf (outf, "S%d : ", rdg->vertices[i].number);
        print_generic_expr (outf, rdg->vertices[i].stmt, \0);
        fprintf (outf, ";\n");
        if (rdg->vertices[i].has_dd_p)
            fprintf (outf, " shape=rect style=filled color=\".7 .3 1.0\"]");
        else
            fprintf (outf, " shape=rect");
        fprintf (outf, ";\n");
    }
    fprintf (outf, "];\n");
}
Next?

- Topological sort of SCC graph
- For each SCC, create a loop
- Mark it parallel or not according to loop carried dependences
More

- http://gcc.gnu.org
- gcc@gcc.gnu.org
- gcc-patches@gcc.gnu.org