Have Fun with GCC

爱好者 prolj@163.com

February 9, 2009

摘要

文中尝试探讨了GCC 的结构，调试，测试，遍编写，移植等方面，目的是作为一个 hand by hand 的入门读物，以后...结合 internal 看代码？优化理论？Arch？...自要自己思考就好
个人能力有限，对 GCC 的一点粗浅认知，欢迎指正。

文中使用了来自 google 的一个调试方法的示例，来自印度技术学院孟买分校的一个源代码示例的中间表示的教学示例，以及来自 HiPEAC 的循环优化的例子。当然，我还拷贝了 gcc internal 很多列举描述性的东西。

目录

第 1 章

Get Build and Test

1.1 获取源代码

我们是要开发 GCC ，不是使用 GCC 做开发，当然要从源码构建了。去 GCC http://gcc.gnu.org/ 主页下载最新版本的源代码或者运行下面命令：

- svn co svn://gcc.gnu.org/svn/gcc/trunk

也许你会觉得你需要 GMP 和 MPFR ，下载之。

1.2 检查系统需求

哦，什么需求呢？你最起码要有个 POSIX 环境吧？哈哈，这条不是认真的，下面是构建 GCC 的系统需求：

- ANSI/ISO C90 编译器
  你编译 GCC 也需要一个编译器啊，对于这一条我表示怀疑， GCC 代码里面那么多 GNU 扩展，别的编译器可以编译 GCC ？所以推荐用 GCC 编译 GCC ，这可不是自举哦。

- GNU binutils
  这个不用说， GCC 仅仅是把代码转变成 ASM ， binutils 才是把 ASM 转变成 bin 的东西。目前 GCC 所支持的机器都是运行 bin 而不是 ASM 的。

- GNAT GNU Ada编译器
  如果你需要编译 Ada 前端，这是不可缺少的，这个...我目前还不关心。

- POSIX 标准的 shell 或者 GNU bash
  不要抱怨你怎么无法在流行的 Windows 下编译 GCC 啊， GCC 是在 UNIX 下开发使用的，所以，你至少需要一个 POSIX 的 shell ，或者 GNU bash ， 最流行的 UNIX 变种 Linux 下最流行的 shell 。

- POSIX 标准的 awk 或者 SVR4 标准的也可以
  其实 GNU awk 就可以了，你的 Linux 套件中会有的。 GCC 在编译时所生成的代码就是由这个脚本按照模板生成的。

- GNU make
  土人不会用 Makefile，编译 GCC 需要 make ，我们不是土人。

- GNU Multiple Precision Library （GMP） 和 MPFR Library
  前者就是 GNU 多精度库啦，配置的时候有三个相关选项：

  ```
  --with-gmp
  --with-gmp-lib
  --with-gmp-include
  ```

  MPFR 和 GMP 是配合，编译 GCC 的时候同时需要 GMP 和 MPFR 。配置的时候同样有三个相关选项：

  ```
  --with-mpfr
  --with-mpfr-lib
  --with-mpfr-include
  ```

- Parma Polyhedra Library （PPL）和 CLooG-PPL
  这个选项在编译 GCC 的时候启用 Graphite 循环优化，这个 Graphite 我不知道叫什么。

  ```
  --with-ppl
  ```

  CLooG-PPL 应该被上面的参数一起指定了，如果你系统的默认路径中没有包含 CLooG-PPL ，那么就需要指定下面的参数。

  ```
  --with-cloog
  ```

- jar 或者 zip & unzip
  这是构建 GCJ （GNU Java 编译器）所需要的运行时系统， libgcj 所需要的。

- GNU tar & gzip & bzip2 | svn
  显然这是我们获取代码的方式。

下面来看我们修改 GCC 所需要的软件包。

- autoconf & automake & GNU m4
  这个 auto 系列都不陌生，在修改 configure.ac， aclocal.m4 等用于生成 configure 和 config.in 的文件的时候 m4 就用上了。在修改用于生成 Makefile.in 文件的 Makefile.am 文件的时候 automake 就派上用场了。像 gcc， intl， libcpp， libiberty， libobjc 这些不需要 automake 的目录，你就得自己修改 Makefile.in 了，很痛苦，是不?

- gettext
  gcc.opt 就是他生成的

- gperf
  有些文件，比如 gcc/cp/cfns.gperf 就需要 gperf 去生成 gcc/cp/cfns.h

- DejaGnu & Expect & Tcl
  跑 testcase 必须的。

- guile

  从 fixinc/inclhack.def 和 fixinc/*.tpl 中生成 fixinc/fixincl.x 用到。

  运行 make check 命令的时候用到。

  从 Makefile.tpl 和 Makefile.def 中生成顶层的 Makefile.in 。

- Flex

  有些前端的词法描述文件是要用 lex 来处理的。

  开发版本的 GCC 有些文件需要自己用 lex 生成，发布版本的时候有人替你生成了。

- Texinfo & TeX

  修改了 *.texi 跑 makeinfo 看看文档效果的时候需要 Texinfo。

  make dvi 和 make pdf 即需要 Texinfo 也需要 TeX。

  同样，发布版本有的 doc 在开发版本中没人给你 make ，自己 make 吧。

- SVN SSH
  svn 取得最新的代码，有些讨厌的网站需要“安全”，不得不 ssh 。

- perl

  生成和 libiberty 目录有联系的 Makefile 。

生成 libiberty/functions.texi 。

生成 manpages 。

在 Darwin （Mac OS X 的核心系统） 下面编译 libstdc++ 用，并且千万别用 --disable-symvers 参数。

被很多脚本使用。

- diff & patch
  补丁，不用我说了吧?
- ecjl & gjavah
  Java...不关心
- antlr.jar 或者 antlr
  Java...同样不关心

1.3 配置

该配置了，创建一个构建目录，在那里面进行构建，我不希望把源代码目录搞乱，因为那会造成阅读的时候莫名的恼怒。

```
# mkdir objdir
# cd objdir
# srcdir/configure [options] [target]
```

options ？运行下面命令，看看自己需要什么 options 。

```
configure –help
```

target ？默认是本机环境，如果你想构建交叉编译器，就需要指定这个参数了。

```
–target=target
```

target 是 m68k-coff ， m68k-linux-elf 这样的格式，指定 cpu 类型，二进制文件格式，甚至 OS 环境。

1.4 构建

```
make
```

So easy. Isn't it? Yeah. You want more? Yeah... But... You need more CPUs or Cores.

```
make -jN
```

N 指定并行进行编译的 GCC 的数目。如果你没有足够的 CPU 或者 Cores 这个参数不会起到好的作用。

或者用这种方法，反馈编译，优化编译器本身，提高的当然是运行速度了，不过这个特性尚在实验之中，自己负责后果哦。和标准编译流程不同的是在编译 stage1 的时候需要 64 位的 int 。

```
make profiledbootstrap
```

现在说一下简单的一个 make 背后的故事

- 构建编译 GCC 需要的工具，说实话，这一步理解真不深，也许是指构建那些需要的脚本，和一些 utils 。
- 构建 GCC 要编译 3 遍，每一遍都构建一套完整的 GCC 二进制文件。如果已经有 binutils 就不再构建，如果没有，需要先构建 binutils ，这个需要把 binutils 的代码放在 GCC 代码的顶层目录中，并在 configure 的时候指定。构建 GCC 的时候有个叫作 bootstrap 的名词，意思就是编译一遍并且编译出来二进制文件。
- 比较第二遍和和第三遍的二进制文件是否相同。因为第一遍用本机的编译器构建出来一个 GCC ，先叫其 GCC1 ，这个 GCC1 不知道是否正确，再用 GCC1 编译出来 GCC2，如果能编译出来 GCC2 就说明 GCC1 是可以用的，GCC2 便是我们想得到“最终版本”了，可以还要用 GCC2 再去构建 GCC3 ，然后比较 GCC2 和 GCC3 是否完全一样，这是为了防止有人在编译器里面放后门，毕竟他们可能不像 Dennis 那样无恶意的 Kidding 。
- 为第三遍的二进制文件构建运行时库。

硬盘空间不足？使用下面的命令编译，第一遍和第二遍的 obj 则会在不用的时候立即被删除。

```
make bootstrap-lean
```

嫌编译太慢？默认优化是 -O2 ，下面编译命令可以改成不优化的。

```
make BOOT_CFLAGS='-O' bootstrap
```

使用下面命令增加编译次数。

```
make bootstrap4
```

如果你确定自己代码没问题，不用编译 3 遍，或者你在测试自己的代码，不想等待那漫长的编译，可以把 bootstrap 禁用了，那样就相当于只编译了 bootstrap1， bootstrap2 和 bootstrap3 就没有了。

```
–disable-bootstrap
```

在编译交叉编译器的时候是不使用 bootstrap 的，但是，如果你需要并想要启用 bootstrap ，比如在 powerpc64-unknown-linux-gnu 环境下构建一个 powerpc-unknown-linux-gnu 的交叉编译器，可以用下面的命令编译。

```
–enable-bootstrap
```

这样指定编译的语言。

```
–enable-languages=...
```

我们喜欢构建交叉编译器？因为目标机的配置跑 GCC 实在太可怜了，或者你不想在目标机跑 GCC ，这是需要，不是喜欢。构建交叉编译器的 make 背后是这样的。

- 准备工作，准备工具，这一步和构建本机使用的编译器是一样的。
- 如果没有，就去构建目标机使用的 binutils ，也需要把 binutils 的代码放在 GCC 代码顶层目录 configure 的时候指定。
- 构建目标机的 GCC ，只有一遍。
- 构建目标机 GCC 使用的运行时库。

交叉编译通过 --host 和 --target 参数指定。

1.5 测试

你需要保证系统中有如下两个系统变量。

```
TCL_LIBRARY = /usr/local/share/tcl8.0
DEJAGNULIBS = /usr/local/share/dejagnu
```

在 objdir 中运行下面命令就可以测试了，好运。

```
make -k check
```

或者你可以运行

```
make check-gcc
```

只测试 gcc。
运行

```
make check-g++
```

只测试 g++。

```
make check-gcc RUNTESTFLAGS="execute.exp other-options"
```

进行 gcc 的 execute 测试。

```
make check-g++ RUNTESTFLAGS="old-deja.exp=9805* other-options"
```

只使用以 9805 开头的测试文件进行 g++ 的 old-deja 测试，。
可以使用 Dejagnu 的 --target_board 选项来向测试用例传递多个参数。比如

```
make check-g++ RUNTESTFLAGS="–target_board=unix/-O3/-fmerge-constants"
```

unix 是目标机器名，这个命令会运行表准 g++ 测试，给每个测试用例都传递 -O3和-fmerge-constants 两个参数。

还可以给定一系列的参数，比如

```
"–target_board=arm-sim\{-mhard-float,-msoft-float\}\{-O1,-O2,-O3,\}"
```

相当于

```
–target_board=arm-sim/-mhard-float/-O1
–target_board=arm-sim/-mhard-float/-O2
–target_board=arm-sim/-mhard-float/-O3
–target_board=arm-sim/-mhard-float
–target_board=arm-sim/-msoft-float/-O1
–target_board=arm-sim/-msoft-float/-O2
–target_board=arm-sim/-msoft-float/-O3
–target_board=arm-sim/-msoft-float
```

也可以用 -Wextra 参数指定任意的组合

```
–target_board=unix/-Wextra\{-O3,-fno-strength\}\{-fomit-frame,\}
```

还记得前面编译的时候 make -jN 么？现在测试的时候也可以，当然硬件要足够哦。

```
make -jN check-testsuite//test-target/option1/option2/...
```

1.6 安装

```
make DESTDIR=path-to-rootdir install
```

我们直接看 make install 背后的故事吧。

- 逐行运行 srcdir/config.guess ，并输出其反馈内容。拷贝动作在这时候进行的。
- 输出你新编译 GCC 版本的 gcc -v 信息，表明这是新版本。

OK，结束了。但是千万不要以为这就是 GCC 开发了。

第 2 章

Arch

2.1 gcc 编译流程

先看一下 GCC 的编译流程吧，这是 gcc 的。 g++ ， gfortran 等的类似，不做赘述。这个图放在开始没什么意思，就是让你有个概念。 c 文件被 cpp

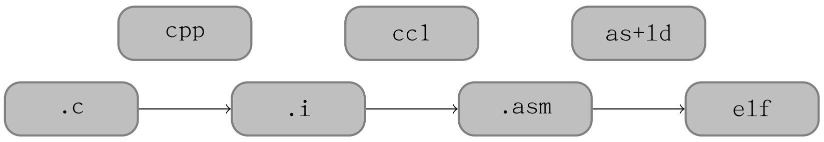

图 2.1: gcc 的编译流程

预处理以后变成 i 文件，cc1编译出来 ASM ， as 汇编 ASM 出来 OBJ ， ld 以后才 elf 的。从图中可以看出其实 gcc 只是一个 driver ，编译起是 cc1 。

2.2 GCC 代码目录结构

下面该开始看代码了，好几百 M 呢！不要怕，这一章我们不会开始真正的代码之旅，真正开始的时候你会不知不觉的发现原来很简单。这一章就是从表面谈 GCC 的一些概念，后面的内容才是这些东西的代码实现。首先看一下 GCC 的代码目录结构吧。

- boehm-gc
  Java 的运行时支持，谁让 Java 非得 GC 呢。
- conifg
  配置文件啊。

- contrib
  别人贡献的代码。
- fixincludes
  头文件，修正的。
- gcc
  前端、后端、优化，都在这里面呢。
- gnattools
  Ada 的工具。
- include
  头文件。
- INSTALL
  安装指南。
- intl
  国际化。
- libada
  Ada 的运行时支持。
- libcpp
  预处理器。
- libdecnumber
  十进制算术库。
- libffi
  Java 的运行时支持，不清楚干什么的。
- libgcc
  这个可不是什么运行时支持，这是 gcc 用软件模拟某些 CPU 不支持的函数库。比如有的 CPU 不支持乘法指令，这里面就有软件实现的乘法。
- libgfortran
  Fortran 的运行时支持。
- libgomp
  OpenMP 的运行时支持。
- libiberty
  通用数据结构和一些支持函数。
- libjava
  Java 的运行时支持。
- libmudflap
  指针/存储检查的运行时支持。

- libobjc
  Objective-C 的运行时支持。

- libssp
  栈溢出保护的运行时支持。

- libstdc++-v3
  C++ 的运行时支持。就是 C++ 标准库。

- maintainer-scripts
  维护脚本。

- zlib
  zlib 是一个压缩解压库， Java 运行时系统要用的。 jar 是压缩文件，zlib 支持内存中解压， jar 就是在内存中解压然后运行的。

哎妈呀，目录不少啊，随便看看吧，下面主要对付 gcc 目录，就是那个“前端、后段、优化”的那个。其实这才是 GCC 的核心所在。

- ada
  Ada 语言前端。

- config
  平台相关的都在里面了。

- cp
  C++ 语言前端。

- doc
  文档。

- fortran
  Fortran 语言前端。

- ginclude
  头文件。

- java
  Java 语言前端。

- objc
  Objctive-C 语言前端。

- objcp
  Objctive-C++ 语言前端。

- po
  翻译，多语言信息。

- testsuite
  测试包。

- treelang
  一种简单的语言前端，用于演示怎么编写 GCC 前端。

2.3 GCC 内部结构

下面我们来看看 GCC 的内部结构吧。

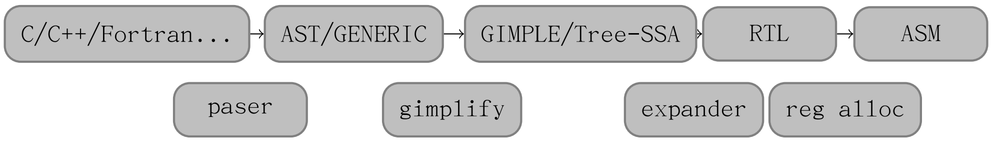

图 2.2: GCC 内部结构

GCC 的内部编译流程大体上是这样的。

- 高级语言先被 GCC 的前端 paser 分析生成 GENERIC ，有的前端可能在 GENERIC 之前使用 AST ，然后做一个 AST 到 GENERIC 的转换。 GENERIC 是所有语言共用的前端中间表示，引入 GENERIC 的原因是 GCC 要加入 SSA 支持，但是 SSA 没有增加在 RTL 上，而是想增加在 Tree 上，可是 GCC 不同的前端使用不同的 Tree ，如果直接给各个语言前端增加 SSA 支持的话就太繁琐了，一个 SSA 算法要为不同语言的前端 Tree 实现多次。所以， GCC 引入了 GENERIC Tree ，作为各种语言前端的通用 Tree ，但是 SSA 变换不在这里做，继续往下看。

- GENERIC 被一个叫做 gimplify 的 lower 简化成 GIMPLE ，因为优化在经过简化的 GIMPLE 上更容易进行。哦，这里还没有 SSA ，继续往下看。GENERIC 到 GIMPLE 的 lower 主要是
  - 指令都被转化为三地址的形式
  - 并且所有变量使用同一个命名空间
  - 控制流降低，指令序列＋jump ，指令序列仍然是被组织成 Tree 的

- GIMPLE 被 SSA 化，叫做 Tree-SSA ， SSA 全称 Static Single Assignment，翻译过来就是静态单一赋值。 Tree-SSA 就是在 Tree 上进行 SSA 变化形成的 SSA 形式的 Tree 。我要强调的是 SSA 不是优化，而是 IR 的一种表现形式，至于 SSA 是什么，请查阅相关资料。

- Tree/SSA 被 unSSA 并且转化为 RTL ，控制流被降低。到 RTL 已经是机器相关的了，意思是这些 RTL 代码本身虽然是机器无关的，但是此时产生 RTL 代码已经对应目标机代码了。

- 在 RTL 上数据流分析，指令选择，寄存器分配，emit ASM。

下面再来看一下 GCC 的优化，虽然 lower 也是优化，但是我们在这里更加关注那种在一种 IR 上的优化

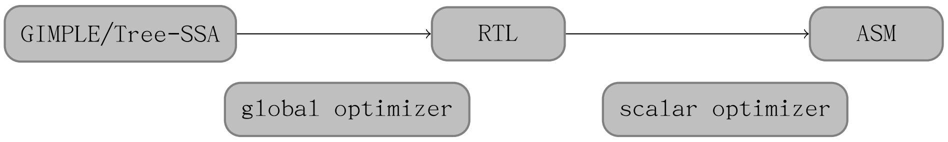

图 2.3: GCC 优化示意图

GCC 的优化大致分两部分

- 作用于 GIMPLE/Tree-SSA 上的全局优化，大约100遍
- 作用于 RTL 上的标量优化，大约70遍

OK ，最后看一下 GCC 的内部调用流程吧。我就不解释了，要知道，画图也是很麻烦的，你根据函数名就能猜出来大概意思，要想明白细节，看代码去吧。

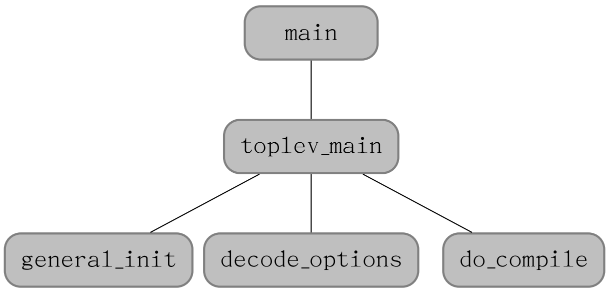

图 2.4: GCC 内部调用流程 driver 部分

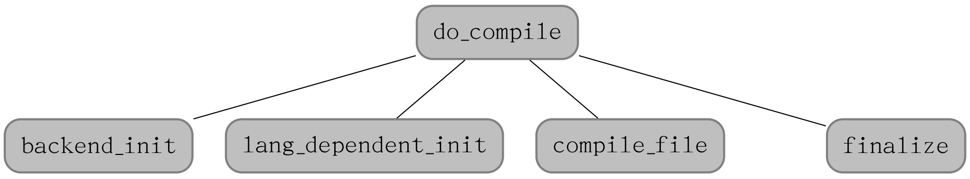

图 2.5: GCC 内部调用流程 初始化部分

finalize 这个函数是编译结束之后清理现场的

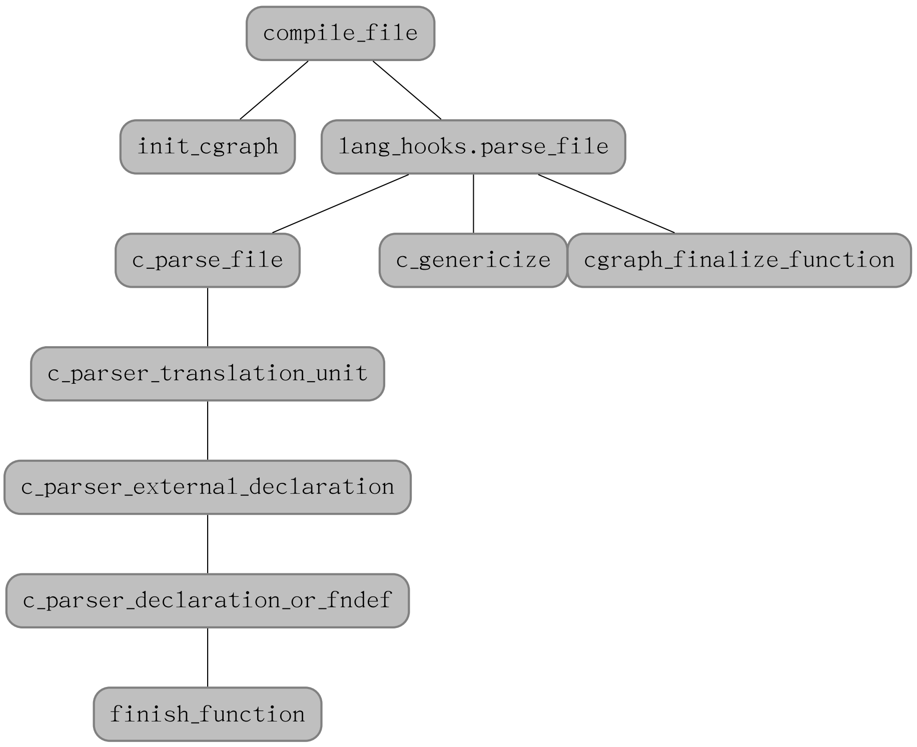

图 2.6: GCC 内部调用流程 C 语言前端部分

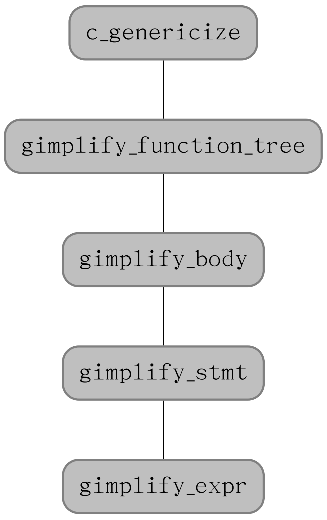

图 2.7: GCC 内部调用流程 gimplify 部分

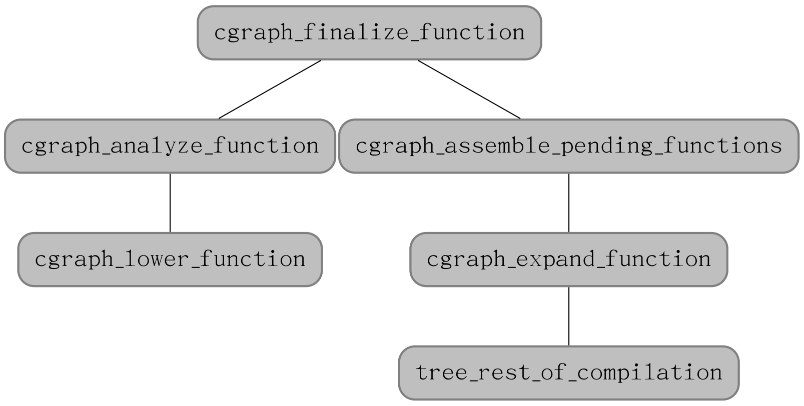

图 2.8: GCC 内部调用流程 作用在 gimple 上的 lowering 部分

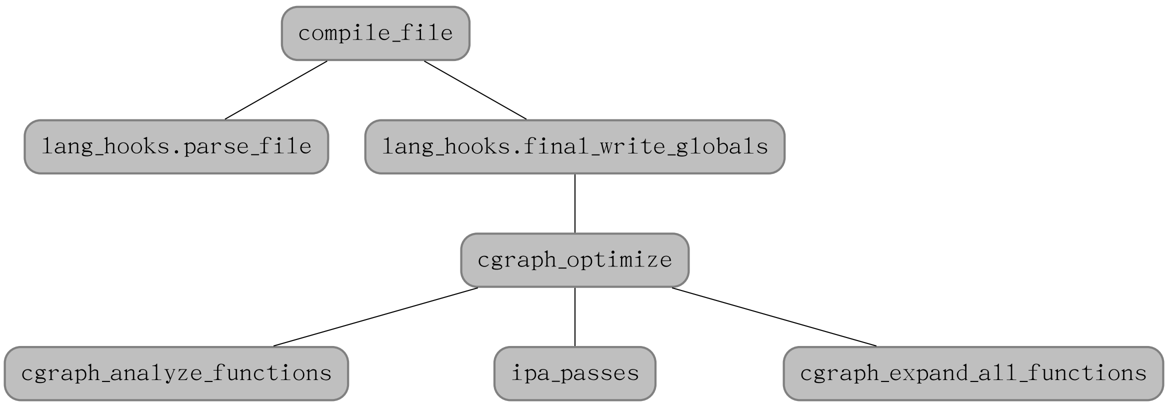

图 2.9: GCC 内部调用流程 作用在 gimple 上的优化部分

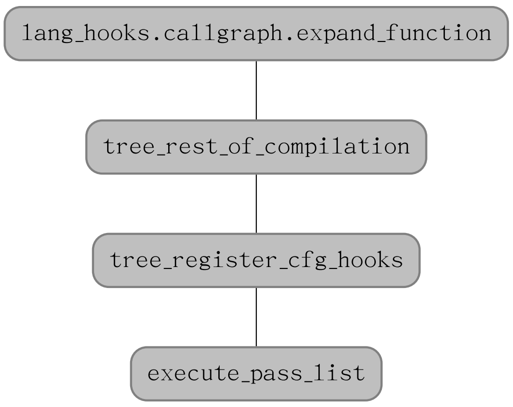

图 2.10: GCC 内部调用流程 gimple/tree-ssa 优化部分

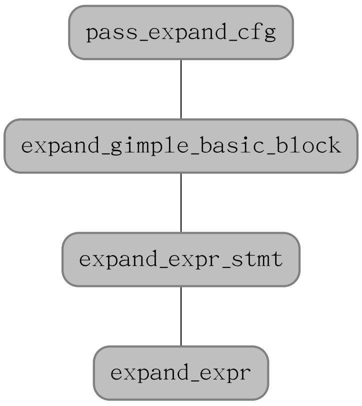

图 2.11: GCC 内部调用流程 gimple-to-rtl 的展开部分

下面我们来看一下 tree-optimize.c 中的 passes ，作用在 gimple/tree-ssa 上的优化，也就是 execute pass list 要调用那些 passes 。

- pass_gimple
- pass_remove_useless_stmts //无用表达式删除
- pass_mudflap_1 //Mudflap pass 1
- pass_lower_cf //控制流降低
- pass_lower_eh //异常降低
- pass_build_cfg //构造控制流图
- pass_pre_expand //向量和复数的展开
- pass_tree_profile //Tree profiler
- pass_init_datastructures //初始化数据结构
- pass_all_optimizations //所有优化的列表

    pass_referenced_vars
    pass_build_ssa
    pass_may_alias
    pass_rename_ssa_copies
    pass_early_warn_uninitialized
    pass_dce //死代码删除
    pass_dominator

```
pass_redundant_phi
pass_dce
pass_merge_phi
pass_forwprop //前向传播
pass_phiopt
pass_may_alias
pass_tail_recursion
pass_ch //循环起点复制
pass_profile
pass_sra
pass_may_alias
pass_rename_ssa_copies
pass_dominator
pass_redundant_phi
pass_dce
pass_dse //无用store消除
pass_may_alias
pass_forwprop
pass_phiopt
pass_ccp //有条件的常量传播
pass_redundant_phi
pass_fold_builtins
pass_may_alias
pass_split_crit_edges
pass_pre //部分冗余删除
pass_loop
    pass_loop_init
    pass_lim //循环不变外提
    pass_unswitch
    pass_record_bounds
    pass_linear_transform
    pass_iv_canon //Canonical induction variable creation
    pass_if_conversion
    pass_vectorize
    pass_complete_unroll
    pass_iv_optimize
```

```
        pass_loop_done
    pass_dominator
    pass_redundant_phi
    pass_late_warn_uninitialized
    pass_cd_dce
    pass_dse
    pass_forwprop
    pass_phiopt
    pass_tail_calls
    pass_rename_ssa_copies
    pass_del_ssa
    pass_nrv //高级语言相关的返回值优化
    pass_remove_useless_vars
    pass_mark_used_blocks
    pass_cleanup_cfg_post_optimizing
```

- pass_warn_function_return
- pass_mudflap_2
- pass_free_datastructures
- pass_expand //展开成 RTL
- pass_rest_of_compilation //RTL 遍

然后紧接着来看 RTL 上的优化，注意哦，和上面优化遍的最后一遍是衔接的哦。并且，精确到函数了哦。

- remove_unnecessary_notes ()
- init_function_for_compilation ()
- rest_of_handle_jump ()
- rest_of_handle_eh ()
- emit_initial_value_sets ()
- unshare_all_rtl ()
- instantiate_virtual_regs ()

- rest_of_handle_jump2 ()
- rest_of_handle_cse ()
- rest_of_handle_gcse ()
- rest_of_handle_loop_optimize ()
- rest_of_handle_jump_bypass ()
- rest_of_handle_cfg ()
- rtl_register_profile_hooks ()
- rtl_register_value_prof_hooks ()
- rest_of_handle_branch_prob ()
- rest_of_handle_value_profile_transformations ()
- count_or_remove_death_notes (NULL, 1)
- rest_of_handle_if_conversion ()
- rest_of_handle_tracer ()
- rest_of_handle_loop2 ()
- rest_of_handle_web ()
- rest_of_handle_cse2 ()
- rest_of_handle_life ()
- rest_of_handle_combine ()
- rest_of_handle_if_after_combine ()
- rest_of_handle_partition_blocks ()
- rest_of_handle_regmove ()
- split_all_insns (1)
- rest_of_handle_mode_switching ()
- recompute_reg_usage ()
- rest_of_handle_sms ()
- rest_of_handle_sched ()
- rest_of_handle_old_regalloc ()
- rest_of_handle_postreload ()

- rest_of_handle_gcse2 ()
- rest_of_handle_flow2 ()
- rest_of_handle_peephole2 ()
- rest_of_handle_if_after_reload ()
- rest_of_handle_regrename ()
- rest_of_handle_reorder_blocks ()
- rest_of_handle_branch_target_load_optimize ()
- rest_of_handle_sched2 ()
- rest_of_handle_stack_regs ()
- compute_alignments ()
- duplicate_computed_gotos ()
- rest_of_handle_variable_tracking ()
- free_bb_for_insn ()
- rest_of_handle_machine_reorg ()
- purge_line_number_notes (get_insns ())
- cleanup_barriers ()
- rest_of_handle_delay_slots ()
- split_all_insns_noflow ()
- convert_to_eh_region_ranges ()
- rest_of_handle_shorten_branches ()
- set_nothrow_function_flags ()
- rest_of_handle_final ()
- rest_of_clean_state ()

最后看看 RTL 是怎么变成 ASM 的吧，这一章就这样结束了。

- assemble_start_function ()
- final_start_function ()

- `final ()`
- `final_end_function ()`
- `assemble_end_function ()`

第 3 章

IRs

如果你看到过我的书的初稿的话，你会发现我很喜欢 IR ，以至于这个文章我都要单独拿出来一章详细的讲解 IR ，其实在我看来，一个编译器，搞明白它的 IR 之后就基本搞定了它。

3.1 对应源代码看 IR ：来自印度技术学院孟买分校的教学示例

本节示例来自印度技术学院孟买分校的教学示例。
看这段代码

```
int f(char *a)
{
  int n = 10;
  int i, g;

  i = 0;
  while (i < n) {
    a[i] = g * i + 3;
    i = i + 1;
  }
  return i;
}
```

预处理之后是这样的

```
int f(char *a)
{
  int n = 10; int i, g;

  i = 0;
  while (i < n) {
    a[i] = g * i + 3;
    i = i + 1;
  }
  return i;
}
```

语法之后呢？ AST 是这个样子的

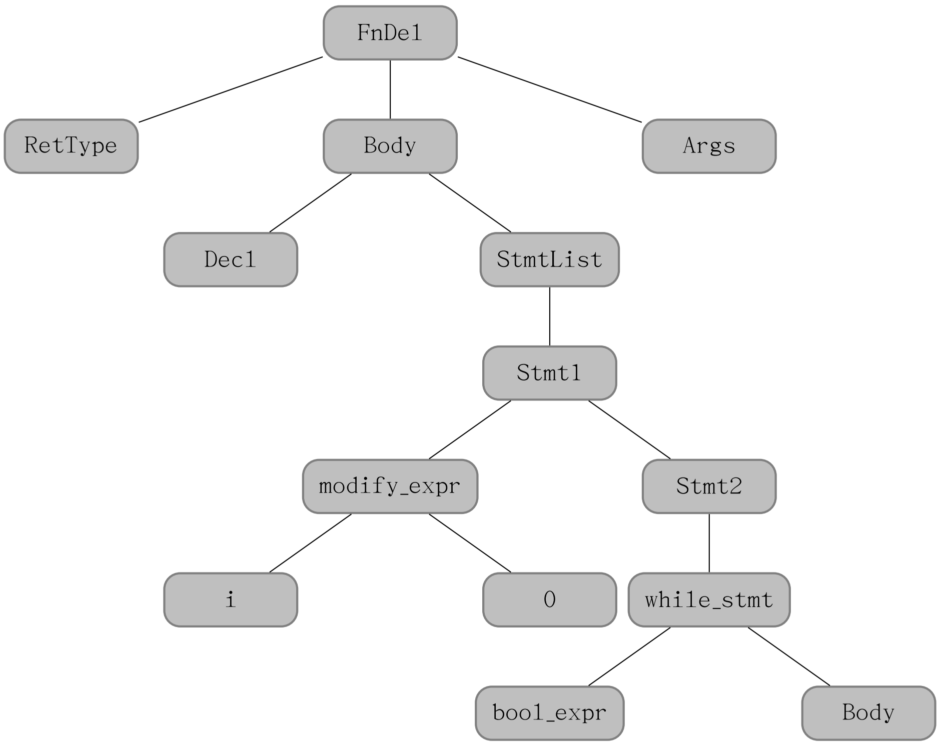

图 3.1: 源代码的 AST 表示

然后呢？ GIMPLE 。

```
f (a)
{
  unsigned int i.0; char * i.1;
  char * D.1140;     int D.1141;
```

```
    ...
  goto <D1136>;
<D1135>:;
    ...
  D.1140 = a + i.1; D.1141 = g * i;
    ...
<D1136>:;
  if (i < n) {goto <D1135>;}
    ...
}
```

下面是什么？ SSA 变换。

```
f (a)
{
  ... int D.1144; ...
<bb 0>:
  n_2 = 10;   i_3 = 0;
  goto <bb 2> (<L1>);
<L0>:;
    ...
  D.1140_9 = a_8 + i.1_7;
  D.1141_11 = g_10 * i_1;
    ...
<L1>:;
  if (i_1 < n_2) goto <L0>; else goto <L2>
    ...
}
```

下面呢？ RTL

```
(insn 21 20 22 2 (parallel [
  (set (reg:SI 61 [ D.1141 ])
    (mult:SI (reg:SI 66)
      (mem/i:SI
        (plus:SI
          (reg/f:SI 54 ...)
          (const_int -8 ...)))))
  (clobber (reg:CC 17 flags))
  ]) -1 (nil)
  (nil))
```

然后呢？ ASM 啦

```
  .file "sample.c"
  ...
f:
pushl %ebp
  ...
movl -4(%ebp), %eax
imull -8(%ebp), %eax
addb $3, %al
  ...
leave
ret
  ...
```

OK ，其实就是让你有这么个大概的概念。

3.2 调试

3.2.1 调试 GCC，最后一个例子来自 google

gdb ？ gdb 。这样

```
gdb –args <location of cc1, cc1plus...> <flags passed to compiler>
```

或者这样

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

当然你编译 GCC 的时候要这样才可以用 gdb 调试

```
BOOT_CFLAGS='-O0 -g3'
```

其实吧，这个我没用过，因为我觉得像我这么笨的人只适合用 print 。给你一个来自 google 的例子

```
gdb –args <path>/cc1 -fpreprocessed a.i -quiet -dumpbase a.c -mtune=generic -auxbase a -O2 -version -o a.s
```

3.2.2 GCC 调试

这可不是教你调试程序的，这是用调试程序的方法观察 GCC 工作是否正常的。基本上就是这样的

```
-fdump-<ir>-<pass>[-<flag1>[-<flag2>]...]
```

不明白？不明白就对了，详细情况要看这里，看了才明白 http://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html

3.3 AST/GENERIC 简介

就是 Tree 嘛， Tree 关键就是节点啦。看 tree.h 里面的 tree_node

```
union tree_node GTY ((ptr_alias (union lang_tree_node),
      desc ("tree_node_structure (&%h)")))
{
  struct tree_base GTY ((tag ("TS_BASE"))) base;
  struct tree_common GTY ((tag ("TS_COMMON"))) common;
  struct tree_int_cst GTY ((tag ("TS_INT_CST"))) int_cst;
  struct tree_real_cst GTY ((tag ("TS_REAL_CST"))) real_cst;
  struct tree_fixed_cst GTY ((tag ("TS_FIXED_CST"))) fixed_cst;
  struct tree_vector GTY ((tag ("TS_VECTOR"))) vector;
  struct tree_string GTY ((tag ("TS_STRING"))) string;
  struct tree_complex GTY ((tag ("TS_COMPLEX"))) complex;
  struct tree_identifier GTY ((tag ("TS_IDENTIFIER"))) identifier;
  struct tree_decl_minimal GTY((tag ("TS_DECL_MINIMAL"))) decl_minimal;
  struct tree_decl_common GTY ((tag ("TS_DECL_COMMON"))) decl_common;
  struct tree_decl_with_rtl GTY ((tag ("TS_DECL_WRTL"))) decl_with_rtl;
  struct tree_decl_non_common GTY ((tag ("TS_DECL_NON_COMMON"))) decl_non_common;
  struct tree_parm_decl  GTY  ((tag ("TS_PARM_DECL"))) parm_decl;
  struct tree_decl_with_vis GTY ((tag ("TS_DECL_WITH_VIS"))) decl_with_vis;
  struct tree_var_decl GTY ((tag ("TS_VAR_DECL"))) var_decl;
  struct tree_field_decl GTY ((tag ("TS_FIELD_DECL"))) field_decl;
  struct tree_label_decl GTY ((tag ("TS_LABEL_DECL"))) label_decl;
  struct tree_result_decl GTY ((tag ("TS_RESULT_DECL"))) result_decl;
  struct tree_const_decl GTY ((tag ("TS_CONST_DECL"))) const_decl;
  struct tree_type_decl GTY ((tag ("TS_TYPE_DECL"))) type_decl;
  struct tree_function_decl GTY ((tag ("TS_FUNCTION_DECL"))) function_decl;
  struct tree_type GTY ((tag ("TS_TYPE"))) type;
  struct tree_list GTY ((tag ("TS_LIST"))) list;
  struct tree_vec GTY ((tag ("TS_VEC"))) vec;
  struct tree_exp GTY ((tag ("TS_EXP"))) exp;
  struct tree_ssa_name GTY ((tag ("TS_SSA_NAME"))) ssa_name;
  struct tree_phi_node GTY ((tag ("TS_PHI_NODE"))) phi;
  struct tree_block GTY ((tag ("TS_BLOCK"))) block;
  struct tree_binfo GTY ((tag ("TS_BINFO"))) binfo;
  struct tree_statement_list GTY ((tag ("TS_STATEMENT_LIST"))) stmt_list;
  struct gimple_stmt GTY ((tag ("TS_GIMPLE_STATEMENT"))) gstmt;
  struct tree_value_handle GTY ((tag ("TS_VALUE_HANDLE"))) value_handle;
  struct tree_constructor GTY ((tag ("TS_CONSTRUCTOR"))) constructor;
  struct tree_memory_tag GTY ((tag ("TS_MEMORY_TAG"))) mtag;
  struct tree_struct_field_tag GTY ((tag ("TS_STRUCT_FIELD_TAG"))) sft;
  struct tree_omp_clause GTY ((tag ("TS_OMP_CLAUSE"))) omp_clause;
  struct tree_memory_partition_tag GTY ((tag ("TS_MEMORY_PARTITION_TAG"))) mpt;
};
```

其实 Tree 节点被分了很多类，有点儿类继承的感觉吧？反正我有。
找节点有多少类型？去 tree.def 里面找吧。

3.4 GIMPLE/Tree-SSA 简介

GIMPLE 只是简化的 AST/GENERIC ， Tree-SSA 只是 SSA 化的 GIMPLE ，所以，GIMPLE/Tree-SSA 的 node 是共用 AST/GENERIC 的。

3.5 RTL 简介

RTL ？ expr.c 中的 expand_expr_* 系列函数把 GIMPLE 展开成 RTL ，一系列的判断， if-else 也好 swicth-case 也好，判断到了就返回相应的 rtx 。
optabs.h 里面通过这个表可以“查”到相应的 RTL 指令。
看 rtl.h 里面的 rtx_def

```

struct rtx_def GTY((chain_next ("RTX_NEXT (&%h)"),
    chain_prev ("RTX_PREV (&%h)")))
{
  /* The kind of expression this is.  */
  ENUM_BITFIELD(rtx_code) code: 16;

  /* The kind of value the expression has.  */
  ENUM_BITFIELD(machine_mode) mode : 8;

  /* 1 in a MEM if we should keep the alias set for this mem unchanged
     when we access a component.
     1 in a CALL_INSN if it is a sibling call.
     1 in a SET that is for a return.
     In a CODE_LABEL, part of the two-bit alternate entry field.  */
  unsigned int jump : 1;
  /* In a CODE_LABEL, part of the two-bit alternate entry field.
     1 in a MEM if it cannot trap.  */
  unsigned int call : 1;
  /* 1 in a REG, MEM, or CONCAT if the value is set at most once, anywhere.
     1 in a SUBREG if it references an unsigned object whose mode has been
     from a promoted to a wider mode.
     1 in a SYMBOL_REF if it addresses something in the per-function
     constants pool.
     1 in a CALL_INSN, NOTE, or EXPR_LIST for a const or pure call.
     1 in a JUMP_INSN, CALL_INSN, or INSN of an annulling branch.  */
  unsigned int unchanging : 1;
    /* 1 in a MEM or ASM_OPERANDS expression if the memory refer-
ence is volatile.
     1 in an INSN, CALL_INSN, JUMP_INSN, CODE_LABEL, BARRIER, or NOTE
     if it has been deleted.
     1 in a REG expression if corresponds to a variable declared by the user,
     0 for an internally generated temporary.
     1 in a SUBREG with a negative value.
     1 in a LABEL_REF, REG_LABEL_TARGET or REG_LABEL_OPERAND note for a
     non-local label.
     In a SYMBOL_REF, this flag is used for machine-specific purposes.  */
  unsigned int volatil : 1;
  /* 1 in a MEM referring to a field of an aggregate.
     0 if the MEM was a variable or the result of a * operator in C;
     1 if it was the result of a . or -> operator (on a struct) in C.
     1 in a REG if the register is used only in exit code a loop.
     1 in a SUBREG expression if was generated from a variable with a
     promoted mode.
     1 in a CODE_LABEL if the label is used for nonlocal gotos
     and must not be deleted even if its count is zero.
     1 in an INSN, JUMP_INSN or CALL_INSN if this insn must be scheduled
     together with the preceding insn.  Valid only within sched.
     1 in an INSN, JUMP_INSN, or CALL_INSN if insn is in a delay slot and
     from the target of a branch.  Valid from reorg until end of compilation;
     cleared before used.  */
  unsigned int in_struct : 1;
  /* At the end of RTL generation, 1 if this rtx is used.  This is used for
     copying shared structure.  See `unshare_all_rtl'.
     In a REG, this is not needed for that purpose, and used instead
     in `leaf_renumber_regs_insn'.
     1 in a SYMBOL_REF, means that emit_library_call
     has used it as the function.  */
  unsigned int used : 1;
  /* 1 in an INSN or a SET if this rtx is related to the call frame,
     either changing how we compute the frame address or saving and
     restoring registers in the prologue and epilogue.
     1 in a REG or MEM if it is a pointer.
     1 in a SYMBOL_REF if it addresses something in the per-function
     constant string pool.  */
  unsigned frame_related : 1;
  /* 1 in a REG or PARALLEL that is the current function's return value.
     1 in a MEM if it refers to a scalar.
     1 in a SYMBOL_REF for a weak symbol.  */
  unsigned return_val : 1;

  /* The first element of the operands of this rtx.
     The number of operands and their types are controlled
     by the `code' field, according to rtl.def.  */
  union u {
    rtunion fld[1];
    HOST_WIDE_INT hwint[1];
    struct block_symbol block_sym;
    struct real_value rv;
    struct fixed_value fv;
  } GTY ((special ("rtx_def"), desc ("GET_CODE (&%0)"))) u;
};
```

然后就应该接着看 rtl.def 里面的东西啊。
继续 rtl.h 里的代码

```

/* Common union for an element of an rtx.  */

union rtunion_def
{
  int rt_int;
  unsigned int rt_uint;
  const char *rt_str;
  rtx rt_rtx;
  rtvec rt_rtvec;
  enum machine_mode rt_type;
  addr_diff_vec_flags rt_addr_diff_vec_flags;
  struct cselib_val_struct *rt_cselib;
  struct bitmap_head_def *rt_bit;
  tree rt_tree;
  struct basic_block_def *rt_bb;
  mem_attrs *rt_mem;
  reg_attrs *rt_reg;
  struct constant_descriptor_rtx *rt_constant;
};
typedef union rtunion_def rtunion;
```

你需要下面的代码理解上面的代码

```
/* The flags and bitfields of an ADDR_DIFF_VEC.  BASE is the base label
   relative to which the offsets are calculated, as explained in rtl.def.  */
typedef struct
{
  /* Set at the start of shorten_branches - ONLY WHEN OPTIMIZING - : */
  unsigned min_align: 8;
  /* Flags: */
  unsigned base_after_vec: 1; /* BASE is after the ADDR_DIFF_VEC.  */
  unsigned min_after_vec: 1;  /* minimum address target label is
 after the ADDR_DIFF_VEC.  */
  unsigned max_after_vec: 1;  /* maximum address target label is
 after the ADDR_DIFF_VEC.  */
  unsigned min_after_base: 1; /* minimum address target label is
 after BASE.  */
  unsigned max_after_base: 1; /* maximum address target label is
 after BASE.  */
  /* Set by the actual branch shortening process - ONLY WHEN OPTIMIZING - : */
  unsigned offset_unsigned: 1; /* offsets have to be treated as unsigned.  */
  unsigned : 2;
  unsigned scale : 8;
} addr_diff_vec_flags;

/* Structure used to describe the attributes of a MEM.  These are hashed
   so MEMs that the same attributes share a data structure.  This means
   they cannot be modified in place.  If any element is nonzero, it means
   the value of the corresponding attribute is unknown.  */
/* ALIGN and SIZE are the alignment and size of the MEM itself,
   while EXPR can describe a larger underlying object, which might have a
   stricter alignment; OFFSET is the offset of the MEM within that object.  */
typedef struct mem_attrs GTY(())
{
  alias_set_type alias; /* Memory alias set.  */
  tree expr; /* expr corresponding to MEM.  */
  rtx offset; /* Offset from start of DECL, as CONST_INT.  */
  rtx size; /* Size in bytes, as a CONST_INT.  */
  unsigned int align; /* Alignment of MEM in bits.  */
} mem_attrs;
```

第 4 章

GIMPLE pass

4.1 简介

想想远古的黄金时代吧，写完 paser 以后就生成 RTL ，然后就是 md 一下 emit ASM 。遍？分析和优化遍在哪里？都在 RTL 上。 RTL 非常适合进行标量优化，在 RTL 上遍总共积累了 70 个遍左右，不要误会哦，很多都是在 md 文件上做手脚，其实源代码到 RTL 已经是相对于某个目标机的 RTL 了，一般认为进入了 RTL 就是后端了，后端优化和 CG 混在一起。

后来呢？在今天的黑铁时代， GIMPLE/Tree-SSA 出现了，干什么的？用于全局优化的，不仅全局标量优化，还有循环优化呢。在全局优化中有一个重要的东西叫 IPA ，过程间分析。要强调 IPO 么？过程间优化，不，强调分析略带过优化即可。那么这个该死的叫做 GIMPLE 的东西带来了什么呢？ 100 个遍，对， 100 个遍。

4.2 来自 HiPEAC 的 GIMPLE/Tree-SSA 循环优化遍

结合例子说吧，我已经厌倦了那些“抽象”的代码。这个循环优化的例子是来自 HiPEAC 的，可不是我弄的哦，我只是拿过来用来说明问题的。来看看添加一个循环优化遍是个什么流程。

- 在 $GCC_SRC/gcc/ 下面增加 pass-example.c
  你的遍在这个文件里面实现
- 修改 $GCC_SRC/gcc/passes.c
  加入你的遍，有宏，添加一个名字就可以了
- 修改 $GCC_SRC/gcc/tree-flow.h
  函数原型啊，你的遍的函数原型啊
- 修改 $GCC_SRC/gcc/tree-pass.h
  你的 pass 的结构名字添进去

- 修改 $GCC_SRC/gcc/common.opt 选项，调用你的 pass 总得有个选项吧
- 修改 $GCC_SRC/gcc/doc/invoke.texi doc ， man 的时候看到的信息，就不写 doc ，对，就不写，早晚你的代码会迅速的让人遗忘，没人维护
- 修改 $GCC_SRC/gcc/timevar.def 计时器啊， GCC 自己有个计时器，专门计算自己每个遍用的时间
- 修改 $GCC_SRC/gcc/Makefile.in Makefile ，我想不用说这个 in 是干什么的了

看 tree-loop-distribution.c 文件中的俩函数，全部内容见附录

```
static unsigned int
tree_loop_distribution (void)
{
  if (!current_loops)
    return 0;

  if (dump_file)
    {
      fprintf (dump_file, "<distribute_loops>\n");

      if (current_function_decl)
        {
          fprintf (dump_file, "<function name=\"%s\"><![CDATA[\n",
                   lang_hooks.decl_printable_name (current_function_decl,
                                                   2));
          dump_function_to_file (current_function_decl,
                                 dump_file, dump_flags);
          fprintf (dump_file, "]]></function>\n");
        }
    }

  distribute_loops (current_loops);

  if (dump_file)
    {
      fprintf (dump_file, "</distribute_loops>\n");
    }

  return 0;
}

static bool
```

```
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

看了下面的代码就知道上面的代码是干什么的了

```
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_verify_loops,            /* todo_flags_finish */
  0                         /* letter */
};
```

这个 struct 被抽象出来了，然后根据你的遍的特性实例化一个这样的 struct ，就像 LLVM 的 pass 都是从 class pass 里面继承出来的那样。

- "ldist", /* name */
  这个遍的名字叫 ldist
- gate_tree_loop_distribution, /* gate */
  大门，gate ，啥意思？就是这个函数返回 true 的话这个遍及其子遍才会被执行
- tree_loop_distribution, /* execute */
  tree_loop_distribution 就是这个遍被运行的时候被运行的函数，对应于 LLVM RunOn*函数
- NULL, /* sub */
  指向其子遍的 tree_opt_pass 的指针
- NULL, /* next */
  指向其兄弟遍的 tree_opt_pass 的指针

- 0, /* static_pass_number */
  计数器
- TV_TREE_LOOP_DISTRIBUTION, /* tv_id */
  GCC 自己有一个计时器, 用来追踪各个遍耗费的时间
- PROP_cfg | PROP_ssa, /* properties_required */
  这个遍需要 cfg 和 ssa , 别说你不知道什么是 cfg 和 ssa
- 0, /* properties_provided */
  不提供
- 0, /* properties_destroyed */
  不销毁
- 0, /* todo_flags_start */
  这一遍开始之前要做的事情
- TODO_verify_loops, /* todo_flags_finish */
  这一遍结束之后要做的事情
- 0 /* letter */
  用于转储 RTL 的 letter

下一步该干什么了? 在 passes.c 里面增加 NEXT_PASS (pass_loop_distribution); 也就是增加我们自己的遍, 位置如下, 就像 LLVM 遍的注册一样

```
NEXT_PASS (pass_linear_transform);
NEXT_PASS (pass_loop_distribution);
NEXT_PASS (pass_iv_canon);
```

在 tree-flow.h 里面添加 extern void distribute_loops (struct loops *); 如下

```
 /* In tree-loop-linear.c  */
 extern void linear_transform_loops (struct loops *);

/* In tree-loop-distribution.c  */
extern void distribute_loops (struct loops *);

 /* In tree-ssa-loop-ivopts.c  */
 bool expr_invariant_in_loop_p (struct loop *, tree);
 bool multiplier_allowed_in_address_p (HOST_WIDE_INT);
```

tree-pass.h 中添加 extern struct tree_opt_pass pass_loop_distribution;

```
extern struct tree_opt_pass pass_if_conversion;
extern struct tree_opt_pass pass_loop_distribution;
extern struct tree_opt_pass pass_vectorize;
```

common.opt 里面添加 ftree-loop-distribution

```
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
```

invoke.texi 里面

```
-ftree-loop-distribution
...
@item -ftree-loop-distribution
Perform loop distribution on tree.  This flag can improve cache
performance (on big loop bodies) and allow further loop optimizations
(like parallelization) to take place.
```

timevar.def 里面添加上 DEFTIMEVAR (TV_TREE_LOOP_DISTRIBUTION, "tree loop distribution")

```
DEFTIMEVAR (TV_TREE_LINEAR_TRANSFORM , "tree loop linear")
DEFTIMEVAR (TV_TREE_LOOP_DISTRIBUTION, "tree loop distribution")
DEFTIMEVAR (TV_TREE_PREFETCH       , "tree prefetching")
```

Makefile.in 里面加入规则

```
tree-loop-distribution.o
...
tree-loop-distribution.o: tree-loop-distribution.c $(CONFIG_H) $(SYSTEM_H) core-
types.h \
   $(TM_H) $(GGC_H) $(OPTABS_H) $(TREE_H) $(RTL_H) $(BASIC_BLOCK_H) \
    $(DIAGNOSTIC_H) $(TREE_FLOW_H) $(TREE_DUMP_H) $(TIMEVAR_H) $(CF-
GLOOP_H) \
   tree-pass.h $(TREE_DATA_REF_H) $(SCEV_H) $(EXPR_H) \
   $(TARGET_H) tree-chrec.h
```

第 5 章

RTL and port

先说废话吧，刚开始学习的时候我看见 GCC 的 md 文件都头疼，看不懂啊。后来我对比了 LLVM 的 port 和 GCC 的 port ，再后来又了解 ORC 的 port ，才发现，原来三个编译器的移植都是差不多的，是，形式上是不一样的，但实际上的内容都是一样，没什么不同的。

曾经我以为不同的 port 很不同，是的，很不同， ISA不同，调用规范不同。但是对比了一下 GCC 中 x86 ， MIPS ， OpenRISC 的 port ，发现原来形式都是一样的，都是 define_* 系列。

为什么说这个废话，因为我觉得 port 是编译器的边缘，就像驱动是内核的边缘一样。

5.1 porting GCC 简介

$GCC_SRC/gcc/config/<arch> ？人人都知道

- <arch>.md
  匹配模式啊，什么样的 ASM 配什么样的 RTL insn ，门当户对啊
- <arch>.h
  机器特性描述（寄存器，调用规范，类型支持...）
- <arch>.c
  CG的支持函数，你 md 文件里面那些使用到的函数都放在这里

其实就是 md 文件， md 里面呢？主要分两种

- 简单的，一个 RTL 对应一个 ASM 的 define_insn
- 复杂的，没有对应 ASM 的 RTL ， 要先映射成其他 RTL 的 define_expand

简单的是什么呢？看这个被各个文档广泛使用的 i386 的例子吧

```
(define_insn "addsi3"
  [(set (match_operand:SI 0 "integer_register_operand" "=d")
        (plus:SI (match_operand:SI 1 "integer_register_operand" "%d")
                 (match_operand:SI 2 "gpr_or_int12_operand" "dNOPQ")))]
  ""
  "add%I2 %1,%2,%0"
  [(set_attr "length" "4")
   (set_attr "type" "int")])
```

- define_insn "addsi3"
  是模式的名字，用于生成对应的 RTL，其中有些名字被代码生成器直接使用，有些名字是强制性的，比如 mov 操作，这个不是我规定的，我只是遵循，有些名字被替换成函数调用。并且这个名字不是必须的， insn combination 的时候就没有名字
- 匹配模式，整个文件的主体就是这个

```
[(set (match\_operand:SI 0 "integer\_register\_operand" "=d")
      (plus:SI (match\_operand:SI 1 "integer\_register\_operand" "\%d")
           (match\_operand:SI 2 "gpr\_or\_int12\_operand" "dNOPQ")))]
```

- "add%I2 %1,%2,%0"
  汇编模板，输出汇编全靠他， %n 被实际的操作数替换，输出可能是字符串或者调用一个 C 函数，这个函数在哪里？在你的 arch.c 里。
- 属性，这也是可选的

```
[(set\_attr "length" "4")
 (set\_attr "type" "int")]
```

5.2 RTL 简介

RTL 是什么啊？ RTL 就是一个有无限寄存器的虚拟机，就像 LLVM 一样，有自己的指令集。 RTL 表达式叫 insn ，这个 insn 是 rtx 的实例， rtx 是被 typedef 的 struct 。 看六种 insn

- INSN 一般指令，不分支那种
- JUMP_INSN JUMP 呗
- CALL_INSN 函数调用
- CODE_LABEL JUMP 得有个 Label 啊
- BARRIER 控制流到这里就停了
- NOTE 调试信息

RTL 的类型和操作都在 rtl.def 里面，自己看看吧
RTL 的匹配模板里面不是有 N 多这个模式么？你得指定你的指令的操作数是啥样的啊。

- QImode 1字节
- HImode 2字节
- SImode 4字节
- DImode 8字节

machmode.def 里面还有更多呢，一般来说这几个够用了
md 文件里面都是什么呢？其实就是下面这些东西的集合，你要是跟我一开始那样害怕就不敢看了，其实静下心来一看就明白了

```
(define_insn 名字 RTL模板 条件 输出模板 属性)
(define_expand 名字 RTL模板 条件 准备语句)
(define_split RTL模板 条件 [RTL模板_1 RTL模板_2 ...] 准备语句)
(define_peephole [RTL模板_1 RTL模板_2 ...] 条件 输出模板 属性)
(define_asm_attributes 属性)
(define_delay 测试 [
                延迟槽测试_1
                分支真删除测试_1
                        分支假删除测试_1
                延迟槽测试_2
                分支真删除测试_2
                       分支假删除测试_2
          ...
          ...
          ])
```

```
(define_function_unit 名字 部件数 可并行指令数 测试结果到达时间 流出起步时
间 冲突表)
(define_attr 名字 值表 缺省值)
(define_combine [...] "" "" )
```

此外还有用于机器描述的一些东西也是经常用到的

```
(match_operand: M 操作数号 谓词 约束)
(match_scratch: M 操作数号 约束)
(match_operator: M 操作数号 谓词 [操作数_1 操作数_2 ...])
(match_parallel: M 操作数号 [rtx _1 rtx _2 ...])
(match_dup: M 操作数号)
(match_op_dup: M 操作数号 [操作数_1 操作数_2 ...])
(match_par_dup: M 操作数号 [操作数_1 操作数_2 ...])
(address rtx)
(eq_attr 名字 值)
(attr_flag 名字)
(attr 名字)
(set_attr 名字 属性)
(set_attr_alternative 名字 值表)
```

5.3 port 一句话

其实学习最好的方法就是去看现有的 GCC port，我给你白白半天起的作用不大。有的时候你写的 md 文件可以用，但是 md 文件里面道道多啊，窥空优化啥的都是在 md 里面的，写不好出错就好，不好的是不出错，但是性能打折扣，慢慢跑 testcase 玩儿吧...

附录 A

来自 HiPEAC 循环优化的 patch

```
2006-05-19  Georges-Andre Silber  <silber@cri.ensmp.fr>

        * doc/invoke.texi (Optimization Options): Document.
        * tree-loop-distribution.c: New file.
        * tree-pass.h (pass_loop_distribution): New.
        * timevar.def (TV_TREE_LOOP_DISTRIBUTION): New.
        * tree-data-ref.c (initialize_data_dependence_relation):
        Initialize and set the new structure element reverse_p.
        * tree-data-ref.h (data_dependence_relation): New element.
        * common.opt (-ftree-loop-distribution): New.
        * tree-flow.h (distribute_loops): Declare.
        * Makefile.in: Add tree-loop-distribution.o.
        * passes.c (init_optimization_passes): Schedule loop distribution.

Index: doc/invoke.texi
===================================================================
— doc/invoke.texi (revision 113325)
+++ doc/invoke.texi (working copy)
@@ -341,7 +341,8 @@ Objective-C and Objective-C++ Dialects}.
 -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 @gol
+-ftree-loop-ivcanon -fivopts @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-vect-loop-version -ftree-salias -fipa-pta -fweb @gol
@@ -5090,6 +5091,11 @@ at @option{-O} and higher.
 Perform linear loop transformations on tree.  This flag can improve cache
 performance and allow further loop optimizations to take place.

+@item -ftree-loop-distribution
+Perform loop distribution on tree.  This flag can improve cache
+performance (on big loop bodies) and allow further loop optimizations
+(like parallelization) to take place.
+
 @item -ftree-loop-im
 Perform loop invariant motion on trees.  This pass moves only invariants that
 would be hard to handle at RTL level (function calls, operations that expand to
Index: tree-loop-distribution.c
===================================================================
— tree-loop-distribution.c (revision 0)
+++ tree-loop-distribution.c (revision 0)
@@ -0,0 +1,1860 @@
+/* Loop Distribution
+   Copyright (C) 2006 Free Software Foundation, Inc.
+   Contributed by Georges-Andre Silber <Georges-Andre.Silber@ensmp.fr>.
+
+This file is part of GCC.
+
+GCC is free software; you can redistribute it and/or modify it under
+the terms of the GNU General Public License as published by the Free
+Software Foundation; either version 2, or (at your option) any later
+version.
+
+GCC is distributed in the hope that it will be useful, but WITHOUT ANY
+WARRANTY; without even the implied warranty of MERCHANTABILITY or
+FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
+for more details.
+
+You should have received a copy of the GNU General Public License
+along with GCC; see the file COPYING.  If not, write to the Free
+Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA
+02110-1301, USA.  */
+
+/* This pass performs loop distribution using a reduced dependence graph
+   (RDG) build using the data dependence analyzer.  This RDG is then used
+   to build a "partition" RDG graph without taking into account data
+   dependences.  Data dependences are then used to build a graph of
+   Strongly Connected Components.  This graph is then topologically sorted
+   to obtain a code generation plan for the distributed loops.
+
+   When using the -fdump-tree-ldist option, it outputs each graph in
+   "dotty" (used by graphviz), surrounded by some XML structure.
```

```
+
+   TODO: Replace arrays by vectors for edges and vertices of RDG.
+   TODO: Generate new distributed loops.  */
+
+#include "config.h"
+#include "system.h"
+#include "coretypes.h"
+#include "tm.h"
+#include "ggc.h"
+#include "tree.h"
+#include "target.h"
+
+#include "rtl.h"
+#include "basic-block.h"
+#include "diagnostic.h"
+#include "tree-flow.h"
+#include "tree-dump.h"
+#include "timevar.h"
+#include "cfgloop.h"
+#include "expr.h"
+#include "optabs.h"
+#include "tree-chrec.h"
+#include "tree-data-ref.h"
+#include "tree-scalar-evolution.h"
+#include "tree-pass.h"
+#include "lambda.h"
+#include "langhooks.h"
+
+#ifdef USE_MAPPED_LOCATION
+typedef source_location LOC;
+#define UNKNOWN_LOC UNKNOWN_LOCATION
+#define EXPR_LOC(e) EXPR_LOCATION(e)
+#define LOC_FILE(l) LOCATION_FILE (l)
+#define LOC_LINE(l) LOCATION_LINE (l)
+#else
+typedef source_locus LOC;
+#define UNKNOWN_LOC NULL
+#define EXPR_LOC(e) EXPR_LOCUS(e)
+#define LOC_FILE(l) (l)->file
+#define LOC_LINE(l) (l)->line
+#endif
+
+/* Initial value for VEC data structures used in RDG.  */
+# define RDG_VS 10
+
+/* Values used in DFS algorithm.  */
+# define VERTEX_WHITE 0
+# define VERTEX_GRAY 1
+# define VERTEX_BLACK 2
+
+typedef struct rdg *rdg_p;
+typedef struct rdg_vertex *rdg_vertex_p;
+typedef struct rdg_edge *rdg_edge_p;
+typedef struct prdg *prdg_p;
+typedef struct prdg_vertex *prdg_vertex_p;
+typedef struct prdg_edge *prdg_edge_p;
+
+DEF_VEC_P(rdg_vertex_p);
+DEF_VEC_ALLOC_P(rdg_vertex_p, heap);
+DEF_VEC_P(rdg_edge_p);
+DEF_VEC_ALLOC_P(rdg_edge_p, heap);
+DEF_VEC_P(prdg_vertex_p);
+DEF_VEC_ALLOC_P(prdg_vertex_p, heap);
+DEF_VEC_P(prdg_edge_p);
+DEF_VEC_ALLOC_P(prdg_edge_p, heap);
+DEF_VEC_I(int);
+DEF_VEC_ALLOC_I(int, heap);
+
+/* A RDG (Reduced Dependence Graph) represents all data dependence
+   constraints between the statements of a loop nest. */
+struct rdg
+{
+  /* The loop nest represented by this RDG.  */
+  struct loop *loop_nest;
+
+  /* The SSA_NAME used for loop index.  */
+  tree loop_index;
+
+  /* The MODIFY_EXPR used to update the loop index.  */
+  tree loop_index_update;
+
+  /* The COND_EXPR that is the exit condition of the loop.  */
+  tree loop_exit_condition;
+
+  /* The PHI_NODE of the loop index.  */
+  tree loop_index_phi_node;
+
+  /* The vertices of the graph.  There is one vertex per
+     statement of the basic block of the loop.  */
+  unsigned int nb_vertices;
+  rdg_vertex_p vertices;
+
+  /* The edges of the graph.  There is one edge per data dependence (between
+     memory references) and one edge per scalar dependence.  */
+  unsigned int nb_edges;
+  rdg_edge_p edges;
+
+  /* Vertices that contain a statement containing an ARRAY_REF.  */
+  VEC (rdg_vertex_p, heap) *dd_vertices;
+
+  /* Data references and array data dependence relations.  */
+  VEC (ddr_p, heap) *dependence_relations;
+  VEC (data_reference_p, heap) *datarefs;
+};
+
+#define RDG_LOOP(G)   (G)->loop_nest
+#define RDG_IDX(G)         (G)->loop_index
+#define RDG_IDX_UPDATE(G) (G)->loop_index_update
+#define RDG_EXIT_COND(G)  (G)->loop_exit_condition
+#define RDG_IDX_PHI(G)    (G)->loop_index_phi_node
+#define RDG_NBV(G)         (G)->nb_vertices
+#define RDG_NBE(G)         (G)->nb_edges
+#define RDG_V(G)           (G)->vertices
+#define RDG_VERTEX(G,i)   &((G)->vertices[i])
+#define RDG_E(G)           (G)->edges
+#define RDG_EDGE(G,i)      &((G)->edges[i])
+#define RDG_DDV(G)         (G)->dd_vertices
+#define RDG_DR(G)          (G)->datarefs
+#define RDG_DDR(G)         (G)->dependence_relations
+
+/* A RDG vertex representing a statement.  */
+struct rdg_vertex
+{
+  /* This color is used for graph algorithms.  */
+  int color;
+
+  /* The number of the basic block in the loop body.  */
+  unsigned int bb_number;
+
+  /* The number of the vertex.  It represents the number of
+     the statement in the basic block.  */
+  unsigned int number;
+
+  /* The statement represented by this vertex.  */
+  tree stmt;
```

```
+
+  /* True when this vertex contains a data reference
+     that is an ARRAY_REF.  */
+  bool has_dd_p;
+
+  /* Vertex is the sink of those edges.  */
+  VEC (rdg_edge_p, heap) *in_edges;
+
+  /* Vertex is the source of those edges. */
+  VEC (rdg_edge_p, heap) *out_edges;
+
+  /* Partitions the vertex is in.
+     If 'has_dd_p' is true, the vertex can only be in one partition.
+     If not, the vertex can be duplicated in several partitions.  */
+  VEC (int, heap) *partition_numbers;
+
+  /* Strongly connected components the vertex is in.
+     If 'has_dd_p' is true, the vertex can only be in one SCC.
+     If not, the vertex can be in several SCCs.  */
+  VEC (int, heap) *scc_numbers;
+};
+
+#define RDGV_COLOR(V)      (V)->color
+#define RDGV_BB(V)         (V)->bb_number
+#define RDGV_N(V)          (V)->number
+#define RDGV_STMT(V)       (V)->stmt
+#define RDGV_DD_P(V)       (V)->has_dd_p
+#define RDGV_IN(V)         (V)->in_edges
+#define RDGV_OUT(V)        (V)->out_edges
+#define RDGV_PARTITIONS(V) (V)->partition_numbers
+#define RDGV_SCCS(V)       (V)->scc_numbers
+
+/* Data dependence type.  */
+enum rdg_dep_type
+{
+  /* Read After Write (RAW) (source is W, sink is R).  */
+  flow_dd = 'f',
+
+  /* Write After Read (WAR) (source is R, sink is W).  */
+  anti_dd = 'a',
+
+  /* Write After Write (WAW) (source is W, sink is W).  */
+  output_dd = 'o',
+
+  /* Read After Read (RAR) (source is R, sink is R).  */
+  input_dd = 'i'
+};
+
+/* An edge of the RDG with dependence information.  */
+struct rdg_edge
+{
+  /* Color used for graph algorithms.  */
+  int color;
+
+  /* The vertex source of the dependence.  */
+  rdg_vertex_p source;
+
+  /* The vertex sink of the dependence.  */
+  rdg_vertex_p sink;
+
+  /* The reference source of the dependence.  */
+  tree source_ref;
+
+  /* The reference sink of the dependence.  */
+  tree sink_ref;
+
+  /* Type of the dependence.  */
+  enum rdg_dep_type type;
+
+  /* Level of the dependence: the depth of the loop that
+    carries the dependence.  */
+  int level;
+
+  /* true if the dependence is between two scalars.  Usually,
+    it is known of a dependence between two memory elements
+    of dimension 0.  */
+  bool scalar_p;
+};
+
+#define RDGE_COLOR(E)       (E)->color
+#define RDGE_SOURCE(E)      (E)->source
+#define RDGE_SINK(E)        (E)->sink
+#define RDGE_SOURCE_REF(E)  (E)->source_ref
+#define RDGE_SINK_REF(E)    (E)->sink_ref
+#define RDGE_TYPE(E)        (E)->type
+#define RDGE_LEVEL(E)       (E)->level
+#define RDGE_SCALAR_P(E)    (E)->scalar_p
+
+/* This graph represents a partition: each vertex is a group of
+   existing RDG vertices, each edge is a dependence between two
+   partitions.  */
+struct prdg
+{
+  /* The RDG used for partitionning.  */
+  rdg_p rdg;
+
+  /* The vertices of the graph.  */
+  VEC (prdg_vertex_p, heap) *vertices;
+
+  /* The edges of the graph.  */
+  VEC (prdg_edge_p, heap) *edges;
+};
+
+#define PRDG_RDG(G)       (G)->rdg
+#define PRDG_NBV(G)       VEC_length (prdg_vertex_p,(G)->vertices)
+#define PRDG_V(G)         (G)->vertices
+#define PRDG_VERTEX(G,i)  VEC_index (prdg_vertex_p,(G)->vertices,i)
+#define PRDG_NBE(G)       VEC_length (prdg_edge_p,(G)->edges)
+#define PRDG_E(G)         (G)->edges
+#define PRDG_EDGE(G,i)    VEC_index (prdg_edge_p,(G)->edges,i)
+
+/* A vertex representing a group of RDG vertices.  */
+struct prdg_vertex
+{
+  /* The partition number.  */
+  int num;
+
+  /* Used for graph algorithms.  */
+  int color;
+
+  /* Discovery time.  Used by DFS.  */
+  int d;
+
+  /* Finishing time.  Used by DFS and SCC.  */
+  int f;
+
+  /* SCC number.  */
+  int scc;
+
+  /* Predecessor after DFS computation.  */
+  prdg_vertex_p pred;
+
+  /* Vertices of the RDG that are in this partition.  */
+  VEC (rdg_vertex_p, heap) *pvertices;
+};
```

```
+
+#define PRDGV_N(V)            (V)->num
+#define PRDGV_COLOR(V)        (V)->color
+#define PRDGV_D(V)            (V)->d
+#define PRDGV_F(V)            (V)->f
+#define PRDGV_SCC(V)          (V)->scc
+#define PRDGV_PRED(V)         (V)->pred
+#define PRDGV_NPV(V)          VEC_length (rdg_vertex_p,(V)->pvertices)
+#define PRDGV_PV(V)           (V)->pvertices
+#define PRDGV_PVERTEX(V,i)   VEC_index (rdg_vertex_p,(V)->pvertices,i)
+
+/* Dependence egde of the partition graph.  */
+struct prdg_edge
+{
+  /* Vertex source of the dependence.  */
+  prdg_vertex_p source;
+
+  /* Vertex sink of the dependence.  */
+  prdg_vertex_p sink;
+
+  /* Original edge of the RDG.  */
+  rdg_edge_p rdg_edge;
+};
+
+#define PRDGE_SOURCE(V)    (V)->source
+#define PRDGE_SINK(V)      (V)->sink
+#define PRDGE_RDG_EDGE(V)  (V)->rdg_edge
+
+
+/* Array to check if a loop has already been distributed.  */
+bool *treated_loops;
+
+/* Loop location.  */
+static LOC dist_loop_location;
+
+/* Function prototype from tree-vectorizer.  */
+LOC find_loop_location (struct loop*);
+
+/* Helper function for Depth First Search.  */
+
+static void
+dfs_rdgp_1 (prdg_p g, prdg_vertex_p v, unsigned int *t, unsigned int scc)
+{
+  unsigned int i;
+  prdg_edge_p e;
+  rdg_vertex_p rdg_v;
+
+  PRDGV_COLOR (v) = VERTEX_GRAY;
+  (*t)++;
+  PRDGV_D (v) = *t;
+  PRDGV_SCC (v) = scc;
+
+  /* If scc!=0, add this SCC to each vertex of the partition. */
+  if (scc)
+    for (i = 0; VEC_iterate (rdg_vertex_p, PRDGV_PV (v), i, rdg_v); i++)
+      VEC_safe_push (int, heap, RDGV_SCCS (rdg_v), scc);
+
+  for (i = 0; VEC_iterate (prdg_edge_p, PRDG_E (g), i, e); i++)
+    if (PRDGE_SOURCE (e) == v)
+      {
+ prdg_vertex_p u = PRDGE_SINK (e);
+
+ if (PRDGV_COLOR (u) == VERTEX_WHITE)
+   {
+     PRDGV_PRED (u) = v;
+     dfs_rdgp_1 (g, u, t, scc);
+   }
+      }
+
+  PRDGV_COLOR (v) = VERTEX_BLACK;
+  (*t)++;
+  PRDGV_F (v) = *t;
+}
+
+/* Depth First Search.  This is an adaptation of the depth first search
+   described in Cormen et al., "Introduction to Algorithms", MIT Press.
+   Returns the max of "finishing times" for the partition graph G.  */
+
+static int
+dfs_rdgp (prdg_p g)
+{
+  unsigned int i;
+  /* t represents the max of finishing times.  */
+  unsigned int t = 0;
+  prdg_vertex_p v;
+
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    {
+      PRDGV_COLOR (v) = VERTEX_WHITE;
+      PRDGV_PRED (v) = NULL;
+    }
+
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    if (PRDGV_COLOR (v) == VERTEX_WHITE)
+      dfs_rdgp_1 (g, v, &t, 0);
+
+  return t;
+}
+
+/* Comparison function to compare "finishing times" of
+   two vertices.  */
+
+static bool
+rdgp_vertex_less_than_p (const prdg_vertex_p a,
+                         const prdg_vertex_p b)
+{
+  return (PRDGV_F (a) < PRDGV_F (b));
+}
+
+/* Helper function for the computation of strongly connected components.  */
+
+static unsigned int
+scc_rdgp_1 (prdg_p g, int max_f)
+{
+  unsigned int i;
+  unsigned int t = 0;
+  unsigned int scc = 0;
+  prdg_vertex_p v;
+  VEC (prdg_vertex_p, heap) *sorted_vertices;
+
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    {
+      PRDGV_COLOR (v) = VERTEX_WHITE;
+      PRDGV_PRED (v) = NULL;
+    }
+
+  /* Here we build a vector containing the vertices sorted by increasing
+     finishing times F (computed by DFS).   This is a contradiction with
+     the complexity of the SCC algorithm that is in linear time
+     O(V+E).   We could have used an array containing pointers to vertices,
+     the index of this array representing F for the corresponding vertex.
+     This array has a size equal to 'max_f' with holes.  */
+
+  sorted_vertices = VEC_alloc (prdg_vertex_p, heap, max_f);
+
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    {
+      unsigned int idx = VEC_lower_bound (prdg_vertex_p, sorted_vertices,
+   v, rdgp_vertex_less_than_p);
+
+      VEC_safe_insert (prdg_vertex_p, heap, sorted_vertices, idx, v);
+    }
+
+  gcc_assert (VEC_length (prdg_vertex_p, sorted_vertices));
+
+  while (VEC_length (prdg_vertex_p, sorted_vertices))
+    {
+      v = VEC_pop (prdg_vertex_p, sorted_vertices);
+
+      if (PRDGV_COLOR (v) == VERTEX_WHITE)
+ dfs_rdgp_1 (g, v, &t, ++scc);
+    }
+
+  VEC_free (prdg_vertex_p, heap, sorted_vertices);
+
+  return scc;
+}
+
+/* Change the directions of all edges.  */
+
+static void
+transpose_rdgp (prdg_p g)
+{
+  unsigned int i;
+  prdg_edge_p e;
+
+  for (i = 0; VEC_iterate (prdg_edge_p, PRDG_E (g), i, e); i++)
+    {
+      prdg_vertex_p tmp = PRDGE_SINK (e);
+
+      PRDGE_SINK (e) = PRDGE_SOURCE (e);
+      PRDGE_SOURCE (e) = tmp;
+    }
+}
+
+/* Computes the strongly connected components of G.  */
+
+static unsigned int
+scc_rdgp (prdg_p g)
+{
```

```
+  unsigned int nb_sccs;
+  int max_f;
+
+  max_f = dfs_rdgp (g);
+  transpose_rdgp (g);
+  nb_sccs = scc_rdgp_1 (g, max_f);
+  transpose_rdgp (g);
+
+  return nb_sccs;
+}
+
+/* Returns true when vertex V is in partition P.  */
+
+static bool
+vertex_in_partition_p (rdg_vertex_p v, int p)
+{
+  int i;
+  int vp;
+
+  for (i = 0; VEC_iterate (int, RDGV_PARTITIONS (v), i, vp); i++)
+    if (vp == p)
+      return true;
+
+  return false;
+}
+
+/* Returns true when vertex V is in SCC S.  */
+
+static bool
+vertex_in_scc_p (rdg_vertex_p v, int s)
+{
+  int i;
+  int vs;
+
+  for (i = 0; VEC_iterate (int, RDGV_SCCS (v), i, vs); i++)
+    if (vs == s)
+      return true;
+
+  return false;
+}
+
+/* Allocates a new partition vertex.  */
+
+static prdg_vertex_p
+new_prdg_vertex (unsigned int p)
```

```
+{
+  prdg_vertex_p v;
+
+  v = XNEW (struct prdg_vertex);
+  PRDGV_N (v) = p;
+  PRDGV_COLOR (v) = 0;
+  PRDGV_D (v) = 0;
+  PRDGV_F (v) = 0;
+  PRDGV_PRED (v) = NULL;
+  PRDGV_SCC (v) = 0;
+  PRDGV_PV (v) = VEC_alloc (rdg_vertex_p, heap, RDG_VS);
+
+  return v;
+}
+
+/* Free a partition vertex.  */
+
+static void
+free_prdg_vertex (prdg_vertex_p v)
+{
+  VEC_free (rdg_vertex_p, heap, PRDGV_PV (v));
+  free (v);
+}
+
+/* Allocates a new partition edge.  */
+static prdg_edge_p
+new_prdg_edge (rdg_edge_p re,
+        prdg_vertex_p sink,
+                prdg_vertex_p source)
+{
+  prdg_edge_p e;
+
+  e = XNEW (struct prdg_edge);
+  PRDGE_RDG_EDGE (e) = re;
+  PRDGE_SINK (e) = sink;
+  PRDGE_SOURCE (e) = source;
+
+  return e;
+}
+
+/* Free a partition edge.  */
+
+static void
+free_prdg_edge (prdg_edge_p e)
+{
```

```
+  free (e);
+}
+
+/* Allocates a new partition graph.  */
+
+static prdg_p
+new_prdg (rdg_p rdg)
+{
+  prdg_p rdgp = XNEW (struct prdg);
+
+  PRDG_RDG (rdgp) = rdg;
+  PRDG_V (rdgp) = VEC_alloc (prdg_vertex_p, heap, RDG_VS);
+  PRDG_E (rdgp) = VEC_alloc (prdg_edge_p, heap, RDG_VS);
+
+  return rdgp;
+}
+
+/* Free a partition graph.  */
+
+static void
+free_prdg (prdg_p g)
+{
+  unsigned int i;
+  prdg_vertex_p v;
+  prdg_edge_p e;
+
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    free_prdg_vertex (v);
+
+  for (i = 0; VEC_iterate (prdg_edge_p, PRDG_E (g), i, e); i++)
+    free_prdg_edge (e);
+
+  VEC_free (prdg_vertex_p, heap, PRDG_V (g));
+  VEC_free (prdg_edge_p, heap, PRDG_E (g));
+}
+
+/* Builds a strongly connected components partition graph of G.  */
+
+static prdg_p
+build_scc_graph (prdg_p g)
+{
+  prdg_p sccg;
+  unsigned int nb_sccs;
+  unsigned int i, j;
+
+  /* Computes the SCC of g.  */
+  nb_sccs = scc_rdgp (g);
+
+  /* Builds a new partition graph of the SCC of g.  */
+  sccg = new_prdg (PRDG_RDG (g));
+
+  /* Create SCC vertices.  */
+  for (i = 0; i < nb_sccs; i++)
+    {
+      unsigned int current_scc = i + 1;
+      unsigned int nbv = RDG_NBV (PRDG_RDG (sccg));
+      prdg_vertex_p v = new_prdg_vertex (current_scc);
+
+      for (j = 0; j < nbv; j++)
+ {
+   rdg_vertex_p rdg_v = RDG_VERTEX (PRDG_RDG (sccg), j);
+
+   if (vertex_in_scc_p (rdg_v, current_scc))
+     VEC_safe_push (rdg_vertex_p, heap, PRDGV_PV (v), rdg_v);
+ }
+
+      PRDGV_SCC (v) = current_scc;
+      VEC_safe_push (prdg_vertex_p, heap, PRDG_V (sccg), v);
+    }
+
+  /* Create SCC edges.  */
+  for (i = 0; i < RDG_NBE (PRDG_RDG (g)); i++)
+    {
+      rdg_edge_p e = RDG_EDGE (PRDG_RDG (g), i);
+
+      /* Here we take only into account data dependences.  */
+      if (!RDGE_SCALAR_P (e))
+ {
+   prdg_edge_p pe;
+   int source_idx = VEC_index (int, RDGV_SCCS (RDGE_SOURCE (e)), 0);
+   int sink_idx = VEC_index (int, RDGV_SCCS (RDGE_SINK (e)), 0);
+
+   gcc_assert (source_idx && sink_idx);
+
+   pe = new_prdg_edge (e, PRDG_VERTEX (sccg, source_idx - 1),
+       PRDG_VERTEX (sccg, sink_idx - 1));
+
+   VEC_safe_push (prdg_edge_p, heap, sccg->edges, pe);
+ }
+    }
```

```
+
+  return sccg;
+}
+
+/* Returns true if the vertex can be recomputed, meaning
+   that the vertex and all the nodes on the path that goes up are only
+   scalars.  */
+
+static bool
+can_recompute_vertex_p (rdg_vertex_p v)
+{
+  rdg_edge_p in_edge;
+  unsigned int i;
+
+  if (RDGV_DD_P (v))
+    return false;
+
+  for (i = 0; VEC_iterate (rdg_edge_p, RDGV_IN (v), i, in_edge); i++)
+    if (RDGE_SCALAR_P (in_edge))
+      if (!can_recompute_vertex_p (RDGE_SOURCE (in_edge)))
+        return false;
+
+  return true;
+}
+
+/* Create one partition in RDG starting from vertex V with a number p.  */
+
+static void
+one_prdg (rdg_p rdg, rdg_vertex_p v, int p)
+{
+  rdg_edge_p o_edge, i_edge;
+  unsigned int i;
+
+  if (vertex_in_partition_p (v, p))
+    return;
+
+  VEC_safe_push (int, heap, RDGV_PARTITIONS (v), p);
+
+  for (i = 0; VEC_iterate (rdg_edge_p, RDGV_IN (v), i, i_edge); i++)
+    if (RDGE_SCALAR_P (i_edge))
+      one_prdg (rdg, RDGE_SOURCE (i_edge), p);
+
+  if (!can_recompute_vertex_p (v))
+    for (i = 0; VEC_iterate (rdg_edge_p, RDGV_OUT (v), i, o_edge); i++)
+      if (RDGE_SCALAR_P (o_edge))
+        one_prdg (rdg, RDGE_SINK (o_edge), p);
+}
+
+/* Returns true if partitions are correct.  */
+
+static bool
+correct_partitions_p (rdg_p rdg, int p)
+{
+  unsigned int i;
+
+  if (!p)
+    return false;
+
+  /* All vertices must have color != 0.  */
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    {
+      if (RDGV_DD_P (RDG_VERTEX (rdg, i))
+   && !VEC_length (int, RDGV_PARTITIONS (RDG_VERTEX (rdg, i))) == 1)
+ return false;
+
+      if (!VEC_length (int, RDGV_PARTITIONS (RDG_VERTEX (rdg, i))))
+        return false;
+    }
+
+  return true;
+}
+
+/* Marks each vertex that contains an ARRAY_REF with the number of the
+   partition it belongs. Returns the number of partitions.
+   This number is at least 1.  */
+
+static unsigned int
+mark_partitions (rdg_p rdg)
+{
+  rdg_vertex_p rdg_v;
+  unsigned int i;
+  int k, p = 0;
+
+  /* Clear all existing partitions.  */
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    VEC_truncate (int, RDGV_PARTITIONS (RDG_VERTEX (rdg,i)), 0);
+
+  /* If there are no dd_vertices, put all in one single partition.  */
+  if (VEC_length (rdg_vertex_p, RDG_DDV (rdg)) == 0)
+    {
```

```
+      /* Mark all vertices with p=1.  */
+      for (i = 0; i < RDG_NBV (rdg); i++)
+        VEC_safe_push (int, heap, RDGV_PARTITIONS (RDG_VERTEX (rdg, i)), 1);
+
+      return 1;
+    }
+
+  /* Mark each vertex with its own color and propagate.  */
+  for (i = 0; VEC_iterate (rdg_vertex_p, RDG_DDV (rdg), i, rdg_v); i++)
+    if (VEC_length (int, RDGV_PARTITIONS (rdg_v)) == 0)
+      one_prdg (rdg, rdg_v, ++p);
+
+  /* Add the vertices that are not in a partition in all partitions.
+     Those vertices does not contain any ARRAY_REF (otherwise, they would
+     have been added by the previous loop on dd_vertices).  */
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    if (VEC_length (int, RDGV_PARTITIONS (RDG_VERTEX (rdg, i))) == 0)
+      for (k = 1; k <= p; k++)
+        VEC_safe_push (int, heap, RDGV_PARTITIONS (RDG_VERTEX (rdg, i)), k);
+
+  gcc_assert (correct_partitions_p (rdg, p));
+
+  return p;
+}
+
+/* Builds a partition graph of an RDG.  This partition represents the
+   maximal distribution of the loops if we break all dependences of level l
+   greater than 0 that are of dimension l.  Note that this graph can have
+   new cycles that were not visible in the RDG.
+
+   The principle of this partition is twofold:
+    - we allow the recomputation of scalar values;
+    - we do not allow the recomputation of array references, because this
+      is what we try to distribute to parallelize the iterations of the loop.
+
+   Vertices that contain an array reference (has_dd_p == true) are in one
+   and only one partition.  Vertices that do not contain any array reference
+   are in one or more partitions.
+
+   This function returns NULL if there are no ARRAY_REF statement
+   in the rdg.  */
+
+static prdg_p
+build_prdg (rdg_p rdg)
+{
```

```
+  unsigned int i, j;
+  rdg_vertex_p rdg_v;
+  prdg_p rdgp = new_prdg (rdg);
+  unsigned int nbp = mark_partitions (rdg);
+
+  /* Create partition vertices.  */
+  for (i = 0; i < nbp; i++)
+    {
+      unsigned int current_partition = i+1;
+      prdg_vertex_p v = new_prdg_vertex (current_partition);
+
+      for (j = 0; j < rdg->nb_vertices; j++)
+ {
+   rdg_v = RDG_VERTEX (rdg, j);
+
+   if (vertex_in_partition_p (rdg_v, current_partition))
+     VEC_safe_push (rdg_vertex_p, heap, PRDGV_PV (v), rdg_v);
+ }
+
+      VEC_safe_push (prdg_vertex_p, heap, PRDG_V (rdgp), v);
+    }
+
+  /* Create partition edges.  */
+  for (i = 0; i < rdg->nb_edges; i++)
+    {
+      rdg_edge_p e = RDG_EDGE (rdg, i);
+
+      /* Here we take only into account data dependences.  */
+      if (!RDGE_SCALAR_P (e))
+ {
+   int so_idx = VEC_index (int, RDGV_PARTITIONS (RDGE_SOURCE (e)), 0);
+          int si_idx = VEC_index (int, RDGV_PARTITIONS (RDGE_SINK (e)), 0);
+   prdg_edge_p pe = new_prdg_edge (e, PRDG_VERTEX (rdgp, so_idx-1),
+   PRDG_VERTEX (rdgp, si_idx-1));
+
+   VEC_safe_push (prdg_edge_p, heap, PRDG_E (rdgp), pe);
+ }
+    }
+
+  return rdgp;
+}
+
+/* Print out a partition graph in DOT format and other informations.  */
+
+static void
```

```
+dump_prdg (FILE *outf, prdg_p rdgp)
+{
+  unsigned int p, i;
+  prdg_vertex_p pv;
+  prdg_edge_p pe;
+  rdg_vertex_p v;
+
+  fprintf (outf, "<graphviz><![CDATA[\n");
+  fprintf (outf, "digraph ");
+  print_generic_expr (outf, RDG_IDX (PRDG_RDG (rdgp)), 0);
+  fprintf (outf, " {\n");
+
+  /* Print out vertices. Each vertex represents a partition, then it
+    can contain several statements.  */
+  for (p = 0; VEC_iterate (prdg_vertex_p, PRDG_V (rdgp), p, pv); p++)
+    {
+      fprintf (outf, " P%d [ shape=rect,label = \" P%d(%d): ",
+        PRDGV_N (pv), PRDGV_N (pv), PRDGV_SCC (pv));
+
+      for (i = 0; VEC_iterate (rdg_vertex_p, PRDGV_PV (pv), i, v); i++)
+ fprintf (outf, "S%d;", RDGV_N (v));
+
+      fprintf (outf, "\"];\n");
+
+      fprintf (outf, " v%d [ label = \" P%d(%d)",  PRDGV_N (pv),
+               PRDGV_N (pv), PRDGV_SCC (pv));
+
+      fprintf (outf, "\"];\n");
+
+     fprintf (outf, "{rank=same; P%d; v%d; }\n",  PRDGV_N (pv), PRDGV_N (pv));
+    }
+
+  for (i = 0; VEC_iterate (prdg_edge_p, PRDG_E (rdgp), i, pe); i++)
+    fprintf (outf, "v%d -> v%d [label=\"%c:%d\" style=dotted];\n",
+      PRDGV_N (PRDGE_SOURCE (pe)),
+      PRDGV_N (PRDGE_SINK (pe)),
+      RDGE_TYPE (PRDGE_RDG_EDGE (pe)),
+      RDGE_LEVEL (PRDGE_RDG_EDGE (pe)));
+
+  fprintf (outf, "}\n");
+  fprintf (outf, "]]></graphviz>\n");
+}
+
+/* Print out loop informations.  */
+
+static void
+dump_loop_infos (FILE *outf, struct loop *loop_nest)
+{
+  fprintf (outf, " <location>\n");
+
+  if (dist_loop_location == UNKNOWN_LOC)
+    fprintf (outf, "  <filename>%s</filename>\n  <line>%d</line>\n",
+             DECL_SOURCE_FILE (current_function_decl),
+             DECL_SOURCE_LINE (current_function_decl));
+  else
+    fprintf (outf, "  <filename>%s</filename>\n  <line>%d</line>\n",
+      LOC_FILE (dist_loop_location), LOC_LINE (dist_loop_location));
+
+  fprintf (outf, " </location>\n");
+  fprintf (outf, " <depth>%d</depth>\n", loop_nest->depth);
+  fprintf (outf, " <level>%d</level>\n", loop_nest->level);
+  fprintf (outf, " <nodes>%d</nodes>\n", loop_nest->num_nodes);
+  fprintf (outf, " <parallel>%d</parallel>\n", loop_nest->parallel_p);
+}
+
```

```
+/* Dump a RDG in DOT format plus other informations.  */
+
+static void
+dump_rdg (FILE *outf, rdg_p rdg)
+{
+  unsigned int i;
+  rdg_vertex_p vertex;
+
+  fprintf (outf, "<graphviz><![CDATA[\n");
+  fprintf (outf, "digraph ");
+  print_generic_expr (outf, RDG_IDX (rdg), 0);
+  fprintf (outf, " {\n");
+
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    {
+      rdg_vertex_p v = RDG_VERTEX (rdg, i);
+
+      fprintf (outf, " v%d [ label = \"", RDGV_N (v));
+      fprintf (outf, "S%d : ", RDGV_N (v));
+      print_generic_expr (outf, RDGV_STMT (v), 0);
+      fprintf (outf, "\"");
+
+      if (RDGV_DD_P (v))
+ fprintf (outf, " shape=rect style=filled color=\".7 .3 1.0\"]");
+      else
+ fprintf (outf, " shape=rect]");
+
+      fprintf (outf, ";\n");
+    }
+
+  for (i = 0; i < RDG_NBE (rdg); i++)
+    {
+      struct rdg_edge *e = RDG_EDGE (rdg, i);
+      struct rdg_vertex *sink = RDGE_SINK (e);
+      struct rdg_vertex *source = RDGE_SOURCE (e);
+
+      fprintf (outf, " v%d -> v%d", RDGV_N (source), RDGV_N (sink));
+      fprintf (outf, " [ label=\"%c:%d", RDGE_TYPE (e), RDGE_LEVEL (e));
+
+      if (RDGE_SCALAR_P (e))
+        fprintf (outf, " d=0");
+      else
+        fprintf (outf, " d=x");
+
+      fprintf(outf, "\" ");
+
+      /* TODO: Here, it is not the level that matters...
+         In fact, it is the dimension of the dependence, a dependence
+         of level=0 with a dimension=1 can be stored and
+         then can be broken.  */
+      if (RDGE_LEVEL (e) > 0)
+        fprintf (outf, " style=dotted");
+
+      fprintf(outf, "]\n");
+    }
+
+  fprintf (outf, "}\n");
+  fprintf (outf, "]]></graphviz>\n");
+  fprintf (outf, "<dd_vertices>\n");
+
+  for (i = 0; VEC_iterate (rdg_vertex_p, RDG_DDV (rdg), i, vertex); i++)
+    {
+      fprintf (outf, "<dd_vertex s=\"s%d\">", RDGV_N (vertex));
+      print_generic_expr (outf, RDGV_STMT (vertex), 0);
+      fprintf (outf, "</dd_vertex>\n");
+    }
+
+  fprintf (outf, "</dd_vertices>\n");
```

```
+}
+
+/* Find the vertex containing a given statement in a RDG or return NULL
+   if the statement is not in any vertex.  */
+
+static rdg_vertex_p
+find_vertex_with_stmt (rdg_p rdg, tree stmt)
+{
+  rdg_vertex_p vertex = NULL;
+  unsigned int i;
+
+  for (i = 0; i < RDG_NBV (rdg) && vertex == NULL; i++)
+    if (RDGV_STMT (RDG_VERTEX (rdg,i)) == stmt)
+      vertex = RDG_VERTEX (rdg, i);
+
+  return vertex;
+}
+
+/* Returns true if the statement is a control statement of the loop.  */
+
+static bool
+loop_nest_control_p (rdg_p rdg, tree stmt)
+{
+  if (TREE_CODE (stmt) == LABEL_EXPR)
+    return true;
+
+  if (stmt == RDG_EXIT_COND (rdg))
+    return true;
+
+  return false;
+}
+
+/* Computes the number of vertices of a given loop nest.  */
+
+static int
+number_of_vertices (rdg_p rdg)
+{
+  basic_block bb;
+  unsigned int i;
+  unsigned int nb_stmts = 0;
+  block_stmt_iterator bsi;
+  struct loop *loop_nest = RDG_LOOP (rdg);
+  basic_block *bbs = get_loop_body (loop_nest);
+
+  for (i = 0; i < loop_nest->num_nodes; i++)
+    {
+      bb = bbs[i];
+
+      /* Test whether the basic block is a direct son of the loop,
+         the bbs array contains all basic blocks in DFS order.  */
+      if (bb->loop_father == loop_nest)
+        /* Iterate of each statement of the basic block.  */
+        for (bsi = bsi_start (bb); !bsi_end_p (bsi); bsi_next (&bsi))
+          if (!loop_nest_control_p (rdg, bsi_stmt (bsi)))
+            nb_stmts++;
+    }
+
+  free (bbs);
+
+  return nb_stmts;
+}
+
+/* Returns true if a statement has a data reference in the given
+   data reference vector.  */
+
+static bool
+contains_dr_p (tree stmt, VEC (data_reference_p, heap) *datarefs)
+{
+  data_reference_p dr;
+  unsigned int i;
+
+  for (i = 0; VEC_iterate (data_reference_p, datarefs, i, dr); i++)
+    if (DR_STMT (dr) == stmt)
+      return true;
+
+  return false;
+}
+
+/* Create vertices of a RDG.  */
+
+static void
+create_vertices (rdg_p rdg)
+{
+  basic_block *bbs;
+  basic_block bb;
+  unsigned int i;
+  unsigned int vertex_index;
+  block_stmt_iterator bsi;
+  struct loop *loop_nest = RDG_LOOP (rdg);
+
+  RDG_NBV (rdg) = number_of_vertices (rdg);
+  RDG_V (rdg) = XCNEWVEC (struct rdg_vertex, RDG_NBV (rdg));
+
+  vertex_index = 0;
+  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 (!loop_nest_control_p (rdg, stmt))
+            {
+              rdg_vertex_p v = RDG_VERTEX (rdg, vertex_index);
+              RDGV_STMT (v) = stmt;
+              RDGV_N (v) = vertex_index;
+              RDGV_BB (v) = i;
+              RDGV_COLOR (v) = 0;
+              RDGV_DD_P (v) = contains_dr_p (stmt, RDG_DR (rdg));
+              RDGV_IN (v) = VEC_alloc (rdg_edge_p, heap, RDG_VS) ;
+              RDGV_OUT (v) = VEC_alloc (rdg_edge_p, heap, RDG_VS) ;
+              RDGV_PARTITIONS (v) = VEC_alloc (int, heap, RDG_VS) ;
+              RDGV_SCCS (v) = VEC_alloc (int, heap, RDG_VS) ;
+              vertex_index++;
+            }
+        }
+    }
+  free (bbs);
+}
+
+/* Checks whether the modify expression correspond to something we
+   can deal with.  */
+
+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;
+    }
+}
+
+/* Checks the statements of a loop body.  */
+
+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
+       && !correct_modify_expr_p (stmt))
+     return false;
+        }
+    }
+
+  free (bbs);
+  return true;
+}
+
+/* Computes the number of uses of a lvalue.  */
+
+static int
```

```
+number_of_lvalue_uses (rdg_p rdg, tree stmt)
+{
+  tree lhs;
+
+  gcc_assert (TREE_CODE (stmt) == MODIFY_EXPR);
+
+  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;
+}
+
+/* Computes the number of scalar dependences to add to the RDG.  */
+
+static int
+number_of_scalar_dependences (rdg_p rdg)
+{
+  unsigned int i;
+  unsigned int nb_deps = 0;
+
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    {
+      tree stmt = RDGV_STMT (RDG_VERTEX (rdg, i));
+
+      if (TREE_CODE (stmt) == MODIFY_EXPR)
+        nb_deps += number_of_lvalue_uses (rdg, stmt);
+    }
+
+  return nb_deps;
+}
+
+/* Computes the number of data dependences to add to the RDG.  */
+
+static int
+number_of_data_dependences (rdg_p rdg)
+{
+  unsigned int nb_deps = 0;
+  ddr_p ddr;
+  unsigned int i;
+
+  for (i = 0; VEC_iterate (ddr_p, RDG_DDR (rdg), i, ddr); i++)
+    if (DDR_ARE_DEPENDENT (ddr) == NULL_TREE)
+      nb_deps += DDR_NUM_DIST_VECTS (ddr);
+
+  return nb_deps;
+}
+
+/* Gets the dependence level with a distance vector.  */
+
+static unsigned int
+get_dependence_level (lambda_vector dist_vect, unsigned int length)
+{
+  unsigned int level;
+  unsigned int i;
+
+  level = 0; /* 0 means a lexicographic dependence */
+
+  for (i = 0; i < length && level == 0; i++)
+    if (dist_vect[i] > 0)
+      level = i + 1;
+
+  return level;
+}
+
+/* Creates an edge with a data dependence vector.  */
+
+static void
+update_edge_with_ddv (ddr_p ddr, unsigned int index_of_vector, rdg_p rdg,
+                      unsigned int index_of_edge)
+{
+  data_reference_p dra;
+  data_reference_p drb;
+  rdg_edge_p edge = RDG_EDGE (rdg, index_of_edge);
+  rdg_vertex_p va;
+  rdg_vertex_p vb;
+
+  /* Invert data references according to the direction of the
+     dependence.  */
+  if (DDR_REVERSE_P (ddr))
+    {
+      dra = DDR_B (ddr);
+      drb = DDR_A (ddr);
+    }
+  else
+    {
+      dra = DDR_A (ddr);
+      drb = DDR_B (ddr);
+    }
+
+  /* Locate the vertices containing the statements that contain
+     the data references.  */
+  va = find_vertex_with_stmt (rdg, DR_STMT (dra));
+  vb = find_vertex_with_stmt (rdg, DR_STMT (drb));
+  gcc_assert (va && vb);
+
+  /* Update source and sink of the dependence.  */
+  RDGE_SOURCE (edge) = va;
+  RDGE_SINK (edge) = vb;
+  RDGE_SOURCE_REF (edge) = DR_REF (dra);
+  RDGE_SINK_REF (edge) = DR_REF (drb);
+
+  /* Determines the type of the data dependence.  */
+  if (DR_IS_READ (dra) && DR_IS_READ (drb))
+    RDGE_TYPE (edge) = input_dd;
+  else if (!DR_IS_READ (dra) && !DR_IS_READ (drb))
+    RDGE_TYPE (edge) = output_dd;
+  else if (!DR_IS_READ (dra) && DR_IS_READ (drb))
+    RDGE_TYPE (edge) = flow_dd;
+  else if (DR_IS_READ (dra) && !DR_IS_READ (drb))
+    RDGE_TYPE (edge) = anti_dd;
+
+  RDGE_LEVEL (edge) = get_dependence_level (DDR_DIST_VECT (ddr,
+                                                             index_of_vector),
+     DDR_NB_LOOPS (ddr));
+  RDGE_COLOR (edge) = 0;
+  RDGE_SCALAR_P (edge) = false;
+
+  VEC_safe_push (rdg_edge_p, heap, RDGV_OUT (va), edge);
+  VEC_safe_push (rdg_edge_p, heap, RDGV_IN (vb), edge);
+}
+
+/* Creates all the edges of a RDG.  */
+
+static void
+create_edges (rdg_p rdg)
+{
+  unsigned int i;
+  unsigned int j;
+  unsigned int edge_index;
+  unsigned int data_edges;
+  unsigned int scalar_edges;
+  struct data_dependence_relation *ddr;
+
+  scalar_edges = number_of_scalar_dependences (rdg);
+  data_edges = number_of_data_dependences (rdg);
+
+  if (scalar_edges == 0 && data_edges == 0)
+    {
+      RDG_NBE (rdg) = 0;
+      RDG_E (rdg) = NULL;
+      return;
+    }
+
+  /* Allocate an array for scalar edges and data edges.  */
+  RDG_NBE (rdg) = scalar_edges + data_edges;
+  RDG_E (rdg) = XCNEWVEC (struct rdg_edge, RDG_NBE (rdg));
+
+  /* Create data edges.  */
+  edge_index = 0;
+
+  for (i = 0; VEC_iterate (ddr_p, RDG_DDR (rdg), i, ddr); i++)
+    if (DDR_ARE_DEPENDENT (ddr) == NULL_TREE)
+      for (j = 0; j < DDR_NUM_DIST_VECTS (ddr); j++)
+        update_edge_with_ddv (ddr, j, rdg, edge_index++);
+
+  /* Create scalar edges. The principle is as follows: for each vertex,
+     if the vertex represents a MODIFY_EXPR (an assignment), we create one
+     edge for each use of the SSA_NAME on the LHS. This edge
+     represents a flow scalar dependence of level 0.  */
+
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    {
+      rdg_vertex_p def_vertex = RDG_VERTEX (rdg, i);
+      tree stmt = RDGV_STMT (def_vertex);
+
+      if (TREE_CODE (stmt) == MODIFY_EXPR)
+        {
+          tree lhs = TREE_OPERAND (stmt, 0);
+
+          if (TREE_CODE (lhs) == SSA_NAME)
+            {
+              use_operand_p imm_use_p;
+              imm_use_iterator iterator;
+
+              FOR_EACH_IMM_USE_FAST (imm_use_p, iterator, lhs)
+                {
+                  rdg_vertex_p use_vertex;
+
+   use_vertex = find_vertex_with_stmt (rdg,
+       USE_STMT (imm_use_p));
+
+   /* If use_vertex != NULL, it means that there is a vertex
+      in the RDG that uses the value defined in
+      def_vertex.  */
+                  if (use_vertex)
+    {
+       rdg_edge_p edge = RDG_EDGE (rdg, edge_index);
+
+       RDGE_LEVEL (edge) = 0;
+       RDGE_SINK (edge) = use_vertex;
+                      RDGE_SOURCE (edge) = def_vertex;
+                      RDGE_SINK_REF (edge) = *(imm_use_p->use);
+                      RDGE_SOURCE_REF (edge) = lhs;
+                      RDGE_COLOR (edge) = 0;
+                      RDGE_TYPE (edge) = flow_dd;
+       RDGE_SCALAR_P (edge) = true;
+       VEC_safe_push (rdg_edge_p, heap,
+                                   RDGV_IN (use_vertex), edge);
+       VEC_safe_push (rdg_edge_p, heap,
+                                   RDGV_OUT (def_vertex), edge);
+                      edge_index++;
+                    }
+                }
+            }
+        }
+    }
+
+  gcc_assert (edge_index == RDG_NBE (rdg));
+}
+
+/* Get the loop index of a loop nest.  */
+
+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;
+}
+
+/* Returns true if the dependences are all computable.  */
+
+static bool
+known_dependences_p (VEC (ddr_p, heap) *dependence_relations)
+{
+  ddr_p ddr;
+  unsigned int i;
+
+  for (i = 0; VEC_iterate (ddr_p, dependence_relations, i, ddr); i++)
+    if (DDR_ARE_DEPENDENT (ddr) == chrec_dont_know)
+      return false;
+
+  return true;
+}
+
+/* Returns the number of phi-nodes of a basic block.  */
+
+static int
+number_of_phi_nodes (basic_block bb)
+{
+  tree phi = phi_nodes (bb);
+  int n;
+
+  for (n = 0; phi; phi = PHI_CHAIN (phi))
+    if (is_gimple_reg (PHI_RESULT (phi)))
+      n++;
+
+  return n;
+}
+
+/* Returns the phi-node containing the loop index.
+   Right know it just returns the first valid phi-node it finds.  */
+
+static tree
+get_index_phi_node (struct loop *loop_nest)
+{
+  tree phi = phi_nodes (loop_nest->header);
+
+  for (; phi; phi = PHI_CHAIN (phi))
+    if (is_gimple_reg (PHI_RESULT (phi)))
+      return phi;
+
+  return NULL_TREE;
+}
+
+static void
+dump_check_info (const char *msg)
+{
+  if (dump_file)
+    fprintf (dump_file, "<loop_check>%s</loop_check>\n", msg);
+}
+
+/* Checks if the loop fit the constraints we impose on a loop.  */
+
+static bool
+loop_is_good_p (struct loop *loop_nest)
+{
+  if (loop_nest->depth != 1)
+    {
+      /* Right now, we only deal with loop nests that
+        are at depth = 1.  */
+      dump_check_info ("depth != 1");
+      return false;
+    }
+  else if (loop_nest->inner)
+    {
+      /* Right now, only consider single loop nests */
+      dump_check_info ("Loop has inner loop(s)");
+      return false;
+    }
+  else if (!loop_nest->single_exit)
+    {
+      /* Only consider loops with a single exit */
+      dump_check_info ("More than one exit");
+      return false;
+    }
+  else if (!get_loop_exit_condition (loop_nest))
+    {
+      /* the exit condition is too difficult to analyze */
+      dump_check_info ("Cannot determine loop exit condition");
+      return false;
+    }
+  else if (loop_nest->num_nodes != 2)
+    {
+      /* If we have two basic blocks, it means that we have the loop body
+         plus the basic block containing the exit label.  */
+      /* If we have more that two basic blocks, it means that
+         we have some complicated control flow.  */
+      dump_check_info ("Complicated control flow");
+      return false;
+    }
+  else if (EDGE_COUNT (loop_nest->header->preds) != 2)
+    {
+      /* Too many incoming edges.  */
+      dump_check_info ("Too many incoming edges");
+      return false;
+    }
+  else if (!empty_block_p (loop_nest->latch))
+    {
+      /* The loop exit condition must be at the end of the loop, the loop
+         header has to contain all the executable statements and the
+         latch has to be empty.  */
+      dump_check_info ("Bad loop form");
+      return false;
+    }
+  else if (empty_block_p (loop_nest->header))
+    {
+      /* The loop must contain some statements.  */
+      dump_check_info ("Empty loop body");
+      return false;
+    }
+  else if (number_of_phi_nodes (loop_nest->header) > 1)
+    {
+      /* We consider that we should have no more than 1 PHI node
+         representing the loop index.  */
+      dump_check_info ("More than one PHI node");
+      return false;
+    }
+  else if (!check_statements (loop_nest))
+    {
+      /* Some lvalues are not correct and then cannot be handled
+         right now.  */
+      dump_check_info ("Bad statement(s) in loop body");
+      return false;
+    }
+  else if (!get_loop_index (loop_nest))
+    {
+      /* We are not able to find out the loop index.  */
+      dump_check_info ("Cannot find loop index");
+      return false;
+    }
+  else if (!get_index_phi_node (loop_nest))
+    {
+      /* Cannot find the PHI node of the loop index.  */
+      dump_check_info ("Cannot find loop index PHI node");
+      return false;
+    }
+
+  /* Note that we do not have to check whether the statements inside
+     the loop body have side effects or not because this check is
+     is going to be done by the data dependence analyzer. */
+  dump_check_info ("OK");
+
+  return true;
+}
+
+/* Build a Reduced Dependence Graph with one vertex per statement of the
+   loop nest and one edge per data dependence or scalar dependence.  */
+
+static rdg_p
```

```
+build_rdg (struct loop *loop_nest)
+{
+  rdg_p rdg;
+  VEC (ddr_p, heap) *dependence_relations;
+  VEC (data_reference_p, heap) *datarefs;
+  unsigned int i;
+  rdg_vertex_p vertex;
+
+  /* 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))
+    {
+      dump_check_info ("Dependences: not computable");
+      free_dependence_relations (dependence_relations);
+      free_data_refs (datarefs);
+      return NULL;
+    }
+  else
+    dump_check_info ("Dependences: OK");
+
+  /* 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 (rdg) = loop_nest;
+  RDG_EXIT_COND (rdg) = get_loop_exit_condition (loop_nest);
+  RDG_IDX (rdg) = get_loop_index (loop_nest);
+  RDG_IDX_UPDATE (rdg) = SSA_NAME_DEF_STMT (RDG_IDX (rdg));
+  RDG_IDX_PHI (rdg) = get_index_phi_node (loop_nest);
+
+  RDG_DDR (rdg) = dependence_relations;
+  RDG_DR (rdg) = datarefs;
+
+  create_vertices (rdg);
+  create_edges (rdg);
+
+  RDG_DDV (rdg) = VEC_alloc (rdg_vertex_p, heap, RDG_VS);
+
+  for (i = 0; i < RDG_NBV (rdg); i++)
+    {
+      vertex = RDG_VERTEX (rdg, i);
+
+      if (RDGV_DD_P (vertex))
+ VEC_safe_push (rdg_vertex_p, heap, RDG_DDV (rdg), vertex);
+    }
+
+  return rdg;
+}
+
+/* Free the RDG.  */
+static void
+free_rdg (rdg_p rdg)
+{
+  unsigned int i;
+  free_dependence_relations (RDG_DDR (rdg));
+  free_data_refs (RDG_DR (rdg));
+
+  if (RDG_NBV (rdg))
+    {
+      for (i = 0; i < RDG_NBV (rdg); i++)
+        {
+          rdg_vertex_p v = RDG_VERTEX (rdg, i);
+
+          VEC_free (rdg_edge_p, heap, RDGV_IN (v));
+          VEC_free (rdg_edge_p, heap, RDGV_OUT (v));
+          VEC_free (int, heap, RDGV_PARTITIONS (v));
+          VEC_free (int, heap, RDGV_SCCS (v));
+        }
+      free (RDG_V (rdg));
+    }
+
+  if (RDG_NBE (rdg))
+    free (RDG_E (rdg));
+
+  VEC_free (rdg_vertex_p, heap, RDG_DDV (rdg));
+  free (rdg);
+}
+
+/* Sort topologically the PRDG vertices.  */
+
+static VEC (prdg_vertex_p, heap)*
+topological_sort (prdg_p g)
+{
```

```
+  unsigned int max_f, i;
+  prdg_vertex_p *vertices;
+  prdg_vertex_p v;
+  VEC (prdg_vertex_p, heap) *sorted_vertices;
+
+  /* Depth First Search.  */
+  max_f = dfs_rdgp (g);
+
+  /* Allocate array of vertices.  */
+  vertices = XCNEWVEC (prdg_vertex_p, max_f+1);
+
+  /* Allocate a vector for sorted vertices.  */
+  sorted_vertices = VEC_alloc (prdg_vertex_p, heap, RDG_VS);
+
+  /* All vertices are set to NULL.  */
+  for (i = 0; i <= max_f; i++)
+    vertices[i] = NULL;
+
+  /* Iterate on each vertex of the PRDG and put each vertex at
+     the right place.  */
+  for (i = 0; VEC_iterate (prdg_vertex_p, PRDG_V (g), i, v); i++)
+    vertices[PRDGV_F (v)] = v;
+
+  /* Push all non-NULL vertices to vector of vertices.  */
+  for (i = max_f; i > 0; i–)
+    if (vertices[i])
+      VEC_safe_push (prdg_vertex_p, heap, sorted_vertices, vertices[i]);
+
+  free (vertices);
+
+  return sorted_vertices;
+}
+
+static void
+open_loop_dump (struct loop *loop_nest)
+{
+  if (dump_file)
+    {
+      fprintf (dump_file , "<LOOP num=\"%d\">\n", loop_nest->num);
+      dump_loop_infos (dump_file, loop_nest);
+    }
+}
+
+static void
+close_loop_dump (void)
```

```
+{
+  if (dump_file)
+    fprintf (dump_file, "</LOOP>\n");
+}
+
+/* Do the actual loop distribution. */
+
+static void
+do_distribution (struct loop *loop_nest)
+{
+  rdg_p rdg; /* Reduced dependence graph.  */
+  prdg_p rdgp; /* Graph of RDG partitions.  */
+  prdg_p sccg; /* Graph of Strongly Connected Components.  */
+  VEC (prdg_vertex_p, heap) *dloops; /* Distributed loops.  */
+
+  open_loop_dump (loop_nest);
+
+  /* Check whether a RDG can be build for this loop nest or not */
+  if (!loop_is_good_p (loop_nest))
+    {
+      close_loop_dump ();
+      return;
+    }
+
+  rdg = build_rdg (loop_nest);
+
+  if (!rdg)
+    {
+      close_loop_dump ();
+      return;
+    }
+
+  if (dump_file)
+    {
+      fprintf (dump_file, "<rdg>\n");
+      dump_rdg (dump_file, rdg);
+      fprintf (dump_file, "</rdg>\n");
+    }
+
+  rdgp = build_prdg (rdg);
+
+  if (dump_file)
+    {
+      fprintf (dump_file, "<prdg>\n");
+      dump_prdg (dump_file, rdgp);
+      fprintf (dump_file, "</prdg>\n");
+    }
+
+  sccg = build_scc_graph (rdgp);
+
+  if (dump_file)
+    {
+      fprintf (dump_file, "<sccp>\n");
+      dump_prdg (dump_file, sccg);
+      fprintf (dump_file, "</sccp>\n");
+    }
+
+  dloops = topological_sort (sccg);
+
+  if (dump_file)
+    {
+      prdg_vertex_p v;
+      int i;
+
+      fprintf (dump_file, "<topological_sort>\n");
+
+      for (i = 0; VEC_iterate (prdg_vertex_p, dloops, i, v); i++)
+        fprintf (dump_file, "  <dloop n=\"%d\">P%d</dloop>\n",
+                 i, PRDGV_N (v));
+
+      fprintf (dump_file, "</topological_sort>\n");
+    }
+
+  free_rdg (rdg);
+  free_prdg (rdgp);
+  free_prdg (sccg);
+  VEC_free (prdg_vertex_p, heap, dloops);
+
+  close_loop_dump ();
+}
+
+/* Iterates on the loops to distribute.  */
+
+void
+distribute_loops (struct loops *loops)
+{
+  unsigned int i;
+  unsigned int dist_loops; /* Number of loops before distribution.  */
+
+  dist_loops = loops->num;
+  treated_loops = XCNEWVEC (bool, dist_loops);
+
+  for (i = 1; i < dist_loops; i++)
+    treated_loops[i] = false;
+
+  for (i = 1; i < dist_loops; i++)
+    {
+      struct loop *loop_nest = loops->parray[i];
+
+      treated_loops[i] = true;
+
+      if (!loop_nest)
+        continue;
+
+      dist_loop_location = find_loop_location (loop_nest);
+
+      do_distribution (loop_nest);
+    }
+
+  free (treated_loops);
+}
+
+/* Function executed by the pass for each function.  */
+
+static unsigned int
+tree_loop_distribution (void)
+{
+  if (!current_loops)
+    return 0;
+
+  if (dump_file)
+    {
+      fprintf (dump_file, "<distribute_loops>\n");
+
+      if (current_function_decl)
+        {
+          fprintf (dump_file, "<function name=\"%s\"><![CDATA[\n",
+                   lang_hooks.decl_printable_name (current_function_decl,
+                                                   2));
+          dump_function_to_file (current_function_decl,
+                                 dump_file, dump_flags);
+          fprintf (dump_file, "]]></function>\n");
+        }
+    }
+
+  distribute_loops (current_loops);
+
+  if (dump_file)
+    {
+      fprintf (dump_file, "</distribute_loops>\n");
+    }
+
+  return 0;
+}
+
+static bool
+gate_tree_loop_distribution (void)
+{
+  return flag_tree_loop_distribution != 0;
+}
+
+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_verify_loops,            /* todo_flags_finish */
+  0                             /* letter */
+};
```

```
Index: tree-pass.h
===================================================================
– tree-pass.h (revision 113325)
+++ tree-pass.h (working copy)
@@ -250,6 +250,7 @@ extern struct tree_opt_pass pass_scev_cp
 extern struct tree_opt_pass pass_empty_loop;
 extern struct tree_opt_pass pass_record_bounds;
 extern struct tree_opt_pass pass_if_conversion;
+extern struct tree_opt_pass pass_loop_distribution;
 extern struct tree_opt_pass pass_vectorize;
 extern struct tree_opt_pass pass_complete_unroll;
 extern struct tree_opt_pass pass_loop_prefetch;
Index: timevar.def
===================================================================
— timevar.def (revision 113325)
+++ timevar.def (working copy)
@@ -108,6 +108,7 @@ DEFTIMEVAR (TV_TREE_LOOP_UNSWITCH     , "
 DEFTIMEVAR (TV_COMPLETE_UNROLL       , "complete unrolling")
 DEFTIMEVAR (TV_TREE_VECTORIZATION    , "tree vectorization")
 DEFTIMEVAR (TV_TREE_LINEAR_TRANSFORM , "tree loop linear")
+DEFTIMEVAR (TV_TREE_LOOP_DISTRIBUTION, "tree loop distribution")
 DEFTIMEVAR (TV_TREE_PREFETCH       , "tree prefetching")
 DEFTIMEVAR (TV_TREE_LOOP_IVOPTS       , "tree iv optimization")
 DEFTIMEVAR (TV_TREE_LOOP_INIT       , "tree loop init")
Index: tree-data-ref.c
===================================================================
— tree-data-ref.c (revision 113325)
+++ tree-data-ref.c (working copy)
@@ -2134,7 +2134,8 @@ initialize_data_dependence_relation (str
   DDR_LOOP_NEST (res) = loop_nest;
   DDR_DIR_VECTS (res) = NULL;
   DDR_DIST_VECTS (res) = NULL;
-
+  DDR_REVERSE_P (res) = false;
+
   for (i = 0; i < DR_NUM_DIMENSIONS (a); i++)
     {
       struct subscript *subscript;
@@ -3675,7 +3676,8 @@ build_classic_dist_vector (struct data_d
    build_classic_dist_vector_1 (ddr, DDR_B (ddr), DDR_A (ddr),
         save_v, &init_b, &index_carry);
    save_dist_v (ddr, save_v);
-
+   DDR_REVERSE_P (ddr) = true;
+
    /* In this case there is a dependence forward for all the
       outer loops:

Index: tree-data-ref.h
===================================================================
— tree-data-ref.h (revision 113325)
+++ tree-data-ref.h (working copy)
@@ -237,6 +237,9 @@ struct data_dependence_relation

   /* The classic distance vector.  */
   VEC (lambda_vector, heap) *dist_vects;
+
+  /* Is the dependence reversed with respect to the lexicographic order?  */
+  bool reverse_p;
 };

 typedef struct data_dependence_relation *ddr_p;
@@ -266,7 +269,7 @@ DEF_VEC_ALLOC_P(ddr_p,heap);
   VEC_index (lambda_vector, DDR_DIR_VECTS (DDR), I)
 #define DDR_DIST_VECT(DDR, I) \
   VEC_index (lambda_vector, DDR_DIST_VECTS (DDR), I)
-
+#define DDR_REVERSE_P(DDR) DDR->reverse_p


 extern tree find_data_references_in_loop (struct loop *,
Index: common.opt
===================================================================
— common.opt (revision 113325)
+++ common.opt (working copy)
@@ -953,6 +953,10 @@ 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
Index: tree-flow.h
===================================================================
— tree-flow.h (revision 113325)
+++ tree-flow.h (working copy)
@@ -928,6 +928,9 @@ bool sra_type_can_be_decomposed_p (tree)
 /* In tree-loop-linear.c  */
 extern void linear_transform_loops (struct loops *);

+/* In tree-loop-distribution.c  */
+extern void distribute_loops (struct loops *);
+
 /* In tree-ssa-loop-ivopts.c  */
 bool expr_invariant_in_loop_p (struct loop *, tree);
 bool multiplier_allowed_in_address_p (HOST_WIDE_INT);
```

```
Index: Makefile.in
===================================================================
— Makefile.in (revision 113325)
+++ Makefile.in (working copy)
@@ -975,6 +975,7 @@ OBJS-common = \
  tree-ssa-loop-ivcanon.o tree-ssa-propagate.o tree-ssa-address.o     \
  tree-ssa-math-opts.o     \
  tree-ssa-loop-ivopts.o tree-if-conv.o tree-ssa-loop-unswitch.o     \
+ tree-loop-distribution.o \
  alias.o bb-reorder.o bitmap.o builtins.o caller-save.o calls.o       \
  cfg.o cfganal.o cfgbuild.o cfgcleanup.o cfglayout.o cfgloop.o     \
  cfgloopanal.o cfgloopmanip.o loop-init.o loop-unswitch.o loop-unroll.o     \
@@ -2086,6 +2087,11 @@ tree-loop-linear.o: tree-loop-linear.c $
     $(DIAGNOSTIC_H) $(TREE_FLOW_H) $(TREE_DUMP_H) $(TIMEVAR_H) $(CF-
GLOOP_H) \
    tree-pass.h $(TREE_DATA_REF_H) $(SCEV_H) $(EXPR_H) $(LAMBDA_H) \
    $(TARGET_H) tree-chrec.h
+tree-loop-distribution.o:    tree-loop-distribution.c    $  (CONFIG_H)    $  (SYS-
TEM_H) coretypes.h \
+   $(TM_H) $(GGC_H) $(OPTABS_H) $(TREE_H) $(RTL_H) $(BASIC_BLOCK_H) \
+   $(DIAGNOSTIC_H) $(TREE_FLOW_H) $(TREE_DUMP_H) $(TIMEVAR_H) $(CF-
GLOOP_H) \
+   tree-pass.h $(TREE_DATA_REF_H) $(SCEV_H) $(EXPR_H) \
+   $(TARGET_H) tree-chrec.h
 tree-stdarg.o: tree-stdarg.c $(CONFIG_H) $(SYSTEM_H) coretypes.h $(TM_H) \
    $(TREE_H) $(FUNCTION_H) $(DIAGNOSTIC_H) $(TREE_FLOW_H) tree-pass.h \
    tree-stdarg.h $(TARGET_H) langhooks.h
Index: passes.c
===================================================================
— passes.c (revision 113325)
+++ passes.c (working copy)
@@ -599,6 +599,7 @@ init_optimization_passes (void)
   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);
```

参考文献

[1] The GCC Team. GCC Internals manual. http://gcc.gnu.org/onlinedocs/gccint/

[2] The GCC Team. GCC 4.3.3 manuals. http://gcc.gnu.org/onlinedocs/gcc-4.3.3/gcc/

[3] Georges-André SILBER. Loop distribution in GCC. 2006

[4] Abhijat Vichare. Introduction to GCC Compilation. 2007

[5] Diego Novillo. GCC Internals. 2007