

1. Tell gcc when to execute the new pass

This is based on the HiPEAC gcc tutorial at <http://www.hipeac.net/gcc-tutorial> or <http://gcc.gnu.org/wiki/OptimizationCourse>. The two important files are: the slides introducing the middle end, and the patch implementing loop distribution on the tree intermediate language (although there're some issues with this patch, like unmatched code with gcc-4.1.0, and compile errors).

In `init_optimization_passes()` of `passes.c`, add `NEXT_PASS (pass_loop_distribution);` where needed.

`pass_loop_distribution` is a data structure of type `tree_opt_pass`, which describes the pass' gate function and execute function. It's documented in `tree-pass.h`.

The gate and `common.opt` together control what command option enables the pass. The pass and all sub-passes are executed only if the gate function returns true. So the gate function is normally one line:

```
return flag_tree_loop_distribution != 0;
```

The flag `flag_tree_loop_distribution` has already been set by the file `common.opt`:

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

When invoking gcc, this pass is enabled by `-O -ftree-loop-distribution`. `-O` is required most likely because this pass is registered under `pass_tree_loop.sub`. For debugging purpose, we can dump the tree using `-fdump-tree-ldist`, with `ldist` being the terse name.

2. Studying the bounds-checking code

This is based on MIRO (<http://wwwhomes.doc.ic.ac.uk/~aw103/projects/miro/>), which in turn based on mudflap.

The instrumentation code is in `gcc/tree-bounds.c`, and the runtime library is in `libbounds/`. It's controlled by `-fbounds-checking` (no `-O` needed). The `pass_bounds_early` structure registers `tree_bounds_early()`, and the `pass_bounds_late` structure registers `tree_bounds_late()` (the flow mimics the classic mudflap project). The early pass transforms declarations, and the late pass transforms references.

3. Debugging gcc

It's suggested that we configure and build gcc in a directory separate from the gcc's source directory. Suppose we build in `build/`, then the main compilers will be `build/gcc/cc1` and `build/gcc/cc1plus` (you need to supply pre-processed files?). Run `make CFLAGS='-g3'` when you build them, or add `-O0` if you want.

Then it should be pretty easy to load the compiler in gdb, see <http://gcc.gnu.org/wiki/DebuggingGCC>.