WIP libgccjit codegen backend for rust

This is a GCC codegen for rustc, which means it can be loaded by the existing rustc frontend, but benefits from GCC: more architectures are supported and GCC's optimizations are used.

Despite its name, libgccjit can be used for ahead-of-time compilation, as is used here.

Motivation

The primary goal of this project is to be able to compile Rust code on platforms unsupported by LLVM. A secondary goal is to check if using the gcc backend will provide any run-time speed improvement for the programs compiled using rustc.

Building

This requires a patched libgccjit in order to work. The patches in this repository need to be applied. (Those patches should work when applied on master, but in case it doesn't work, they are known to work when applied on 079c23cfe079f203d5df83fea8e92a60c7d7e878.) You can also use my fork of gcc which already includes these patches.

Put the path to your custom build of libgccjit in the file `gcc_path`.

```
$ git clone https://github.com/rust-lang/rustc_codegen_gcc.git
$ cd rustc_codegen_gcc
$ git clone https://github.com/llvm/llvm-project llvm --depth 1 --single-branch
$ export RUST_COMPILER_RT_ROOT="$PWD/llvm/compiler-rt"
$ ./prepare_build.sh # download and patch sysroot src
$ ./build.sh --release
```

To run the tests:

```
$ ./prepare.sh # download and patch sysroot src and install hyperfine for
benchmarking
$ ./test.sh --release
```

Usage

`$cg_gccjit_dir` is the directory you cloned this repo into in the following instructions.

Cargo

```
$ CHANNEL="release" $cg_gccjit_dir/cargo.sh run
```

If you compiled cg_gccjit in debug mode (aka you didn't pass `--release` to `./test.sh`) you should use `CHANNEL="debug"` instead or omit `CHANNEL="release"` completely.

Rustc

> *You should prefer using the Cargo method.*

```
$ rustc +$(cat $cg_gccjit_dir/rust-toolchain) -Cpanic=abort -Zcodegen-
backend=$cg_gccjit_dir/target/release/librustc_codegen_gcc.so --sysroot
$cg_gccjit_dir/build_sysroot/sysroot my_crate.rs
```

Env vars

CG_GCCJIT_INCR_CACHE_DISABLED
: Don't cache object files in the incremental cache. Useful during development of cg_gccjit to make it possible to use incremental mode for all analyses performed by rustc without caching object files when their content should have been changed by a change to cg_gccjit.

CG_GCCJIT_DISPLAY_CG_TIME
: Display the time it took to perform codegen for a crate

Debugging

Sometimes, libgccjit will crash and output an error like this:

```
during RTL pass: expand
libgccjit.so: error: in expmed_mode_index, at expmed.h:249
0x7f0da2e61a35 expmed_mode_index
    ../../../gcc/gcc/expmed.h:249
0x7f0da2e61aa4 expmed_op_cost_ptr
    ../../../gcc/gcc/expmed.h:271
0x7f0da2e620dc sdiv_cost_ptr
    ../../../gcc/gcc/expmed.h:540
0x7f0da2e62129 sdiv_cost
    ../../../gcc/gcc/expmed.h:558
0x7f0da2e73c12 expand_divmod(int, tree_code, machine_mode, rtx_def*, rtx_def*,
rtx_def*, int)
    ../../../gcc/gcc/expmed.c:4335
0x7f0da2ea1423 expand_expr_real_2(separate_ops*, rtx_def*, machine_mode,
expand_modifier)
    ../../../gcc/gcc/expr.c:9240
0x7f0da2cd1a1e expand_gimple_stmt_1
    ../../../gcc/gcc/cfgexpand.c:3796
0x7f0da2cd1c30 expand_gimple_stmt
    ../../../gcc/gcc/cfgexpand.c:3857
0x7f0da2cd90a9 expand_gimple_basic_block
    ../../../gcc/gcc/cfgexpand.c:5898
0x7f0da2cdade8 execute
    ../../../gcc/gcc/cfgexpand.c:6582
```

To see the code which causes this error, call the following function:

```
gcc_jit_context_dump_to_file(ctxt, "/tmp/output.c", 1 /* update_locations */)
```

This will create a C-like file and add the locations into the IR pointing to this C file. Then, rerun the program and it will output the location in the second line:

```
libgccjit.so: /tmp/something.c:61322:0: error: in expmed_mode_index, at expmed.h:249
```

Or add a breakpoint to `add_error` in gdb and print the line number using:

```
p loc->m_line
p loc->m_filename->m_buffer
```

To print a debug representation of a tree:

```
debug_tree(expr);
```

To get the `rustc` command to run in `gdb`, add the `--verbose` flag to `cargo build`.

How to use a custom-build rustc

- Build the stage2 compiler (`rustup toolchain link debug-current build/x86_64-unknown-linux-gnu/stage2`).
- Clean and rebuild the codegen with `debug-current` in the file `rust-toolchain`.

How to build a cross-compiling libgccjit

Building libgccjit

- Follow these instructions: https://preshing.com/20141119/how-to-build-a-gcc-cross-compiler/ with the following changes:
- Configure gcc with `../gcc/configure --enable-host-shared --disable-multilib --enable-languages=c,jit,c++ --disable-bootstrap --enable-checking=release --prefix=/opt/m68k-gcc/ --target=m68k-linux --without-headers`.
- Some shells, like fish, don't define the environment variable `$MACHTYPE`.
- Add `CFLAGS="-Wno-error=attributes -g -O2"` at the end of the configure command for building glibc (`CFLAGS="-Wno-error=attributes -Wno-error=array-parameter -Wno-error=stringop-overflow -Wno-error=array-bounds -g -O2"` for glibc 2.31, which is useful for Debian).

Configuring rustc_codegen_gcc

- Set `TARGET_TRIPLE="m68k-unknown-linux-gnu"` in config.sh.
- Since rustc doesn't support this architecture yet, set it back to `TARGET_TRIPLE="mips-unknown-linux-gnu"` (or another target having the same attributes). Alternatively, create a target specification file (note that the `arch` specified in this file must be supported by the rust compiler).
- Set `linker='-Clinker=m68k-linux-gcc'`.
- Set the path to the cross-compiling libgccjit in `gcc_path`.
- Disable the 128-bit integer types if the target doesn't support them by using `let i128_type = context.new_type::<i64>();` in `context.rs` (same for u128_type).
- Comment the line: `context.add_command_line_option("-masm=intel");` in src/base.rs.
- (might not be necessary) Disable the compilation of libstd.so (and possibly libcore.so?).