1 Contents 3
This is the new BRG wiki that is compatible both with GitHub and Read The Docs.
1.1 Users

This directory contains user-specific documentation. This usually comes in the form of weekly journals and personal notes. Each user should define a README.md page inside their directories and a list of contents (see this for an example).

1.1.1 Personal Directory

1.1.2 Personal Directory

1.1.3 Personal Directory

1.1.4 Berkin’s Personal Directory

Notes (incomplete)

Download and install ruby:

```
% cd ruby
% wget https://cache.ruby-lang.org/pub/ruby/2.3/ruby-2.3.1.tar.gz
% tar xzf ruby-2.3.1.tar.gz
% mkdir build
% cd build
% ../configure --prefix=$BITS/stowdir/ruby-2.3.1
% make -j15
% make install
% cd ~/install/stow-pkgs/x86_64-centos6/pkgs
% ln -s $BITS/stowdir/ruby-2.3.1
% stow ruby-2.3.1
```

Follow the Jekyll instructions: https://help.github.com/articles/setting-up-your-github-pages-site-locally-with-jekyll/:

```
% gem install bundler jekyll
```

After this, need to re-stow because the new files aren’t tracked.

```
% cd ~/install/stow-pkgs/x86_64-centos6/pkgs % stow ruby-2.3.1
```

Create a new directory:
Create a file called Gemfile and put the following:

```
source 'https://rubygems.org'
gem 'github-pages', group: :jekyll_plugins
```

Do installation:

```
% bundle install
```

Generate local files:

```
% bundle exec jekyll new . --force
```

**instructions for realms:**

Create a new virtualenv, then `pip install realms-wiki`. This complained because `lber.h` couldn’t be found. Check which package to install using `yum whatprovides "*/lber.h"`, and we need `openldap-devel-XXX`. Yum-installed this (and its dependency `cyrus-sasl`) on `brg-05`.

**ARM notes**

packages installed:

```
protobuf-compiler.x86_64 (not very necessary...)
libtool.x86_64 (required for ArchC (and qemu?))
glibc-static.x86_64 (required for building certain cross-compile toolchains)
```

**Installing qemu**  Download and untar:

```
% wget http://wiki.qemu-project.org/download/qemu-2.1.0.tar.bz2
% tar xjf qemu-2.1.0.tar.bz2
```

Configure:

```
% cd qemu-2.1.0
% mkdir build
% cd build/
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/qemu-2.1.0
```

If getting an error message `error: possibly undefined macro: AC_PROG_LIBTOOL, then need to install libtool. Need to do this through yum though, not like below.

The rest is straightforward. Beware though, compiling takes a while, so `-j 16` it:

```
% make -j 16
% make install
% cd $STOW_PKGS_PREFIX/pkgs
% stow qemu-2.1.0
```

Running was very straightforward. This managed to run non-OS as well OS binaries. Note though, we need to change the `uname` string, similar to what I’ve done in gem5:
Python, Release

% qemu-arm hello-arm-baremetal
% qemu-arm -r "3.12" hello-arm-linux

**Installing libtool (deprecated)**  Note: this section is not needed.

Pretty straightforward:

```bash
% wget http://ftp.wayne.edu/gnu/libtool/libtool-2.4.2.tar.gz
% tar xzf libtool-2.4.2
% mkdir build
% cd build/
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/libtool-2.4.2
% make
% make install
```

... 

**SPEC benchmarks**  Cluster has the spec benchmarks, in `/ufs/cluster/benchmarks`. Copy these over:

```bash
% scp berkin@cluster.csl.cornell.edu:/ufs/cluster/benchmarks/SPEC2006/install_archives/cpu2006.tar.xz .
% mkdir cpu2006
% cd cpu2006
% tar xf ../cpu2006.tar.xz
```

After this we need to “install” the benchmarks. Installation is simple, and it just installs it in the same directory as the default untarred directory:

```bash
% ./install.sh
```

To be able to use the benchmarks, you need to source `shrc`. The documentation here is pretty good.

I used the `linux64-amd64-gcc43+` configuration. It should be possible to add a new configuration for the arm platform relatively easily:

```bash
% cd config/
% cp Example-linux64-amd64-gcc43+.cfg brg-linux.cfg
% cd ..
```

After this building and running a benchmark was pretty easy:

```bash
% runspec --config=brg-linux.cfg --action=build --tune=base bzip2
% runspec --config=brg-linux.cfg --size=test --noreportable --tune=base bzip2
```

And to run the benchmarks officially for the entire suite:

```bash
% runspec --config=brg-linux.cfg --tune=base int
```

To compile spec benchmarks with arm, we need to copy a configuration and set the compiler accordingly:

```bash
% cd configs/
% cp Example-linux64-amd64-gcc43+.cfg brg-arm.cfg
```

In this, we want to change the compilers to respective compilers from `arm-unknown-linux-gnueabi` toolchain. We also want to change the extension, so that we can identify the arm binaries. Also, we need to pass `-static` to the linker:
# berkin:
ext = arm-gcc43
#ext = gcc43-64bit

# berkin:
CC = /work/bits0/bi45/misc/arm-toolchain/arm-unknown-linux-gnueabi/x-tools/arm-unknown-linux-gnueabi/bin/arm-unknown-linux-gnueabi-gcc
CXX = /work/bits0/bi45/misc/arm-toolchain/arm-unknown-linux-gnueabi/x-tools/arm-unknown-linux-gnueabi/bin/arm-unknown-linux-gnueabi-g++
FC = /work/bits0/bi45/misc/arm-toolchain/arm-unknown-linux-gnueabi/x-tools/arm-unknown-linux-gnueabi/bin/arm-unknown-linux-gnueabi-gfortran
#CC = /usr/bin/gcc
#CXX = /usr/bin/g++
#FC = /usr/bin/gfortran

# berkin: we need to pass -static to the linker for arm
LDCLFLAGS = -static
LDCLXXFLAGS = -static
LDFFLAGS = -static

We want to build and run the entire suite natively first so that we can check the command line args etc. (this uses test size, but larger sizes should be similar):

% runspec --loose --size test --tune base --config brg-linux --iterations=1 int

470.lbm is apparently fairly small so we can try porting it:

% runspec --loose --size test --tune base --config brg-arm --iterations=1 lbm

This creates benchspec/CPU2006/470.lbm. The executables are in the exe/ directory with the appropriate suffix. To see the running args, go to the appropriate run:

% cd benchspec/CPU2006/470.lbm
% specinvoke -n
# specinvoke r6392
# Invoked as: specinvoke -n
# timer ticks over every 1000 ns
# Use another -n on the command line to see chdir commands and env dump
# Starting run for copy #0
../run_base_test_arm-gcc43.0000/lbm_base.arm-gcc43 20 reference.dat 0 1 100_100_130_cf_a.of > lbm.out 2>> lbm.err

specinvoke -n just does a dry run, so it doesn’t actually run it. It might be a good idea to run this for real on the host to see the expected outputs. Now, we can try running this on gem5:

% cd <gem5 dir>
% mkdir eval
% cd eval
% mkdir spec
% cd spec
% ln -s /work/bits0/bi45/misc/spec/cpu2006/benchspec/CPU2006/470.lbm/run/run_base_test_arm-gcc43.00000
% ./build/ARM/gem5.opt configs/example/se.py -c eval/spec/470.lbm/lbm_base.arm-gcc43 -o "-h"
% ./build/ARM/gem5.opt configs/example/se.py -c eval/spec/470.lbm/lbm_base.arm-gcc43 -o "20 reference.dat 0 1 eval/spec/470.lbm/100_100_130_cf_a.of"

To recompile, you can simply use --rebuild flag in specinvoke:

% runspec --loose --size test --tune base --config brg-arm-newlib --iterations --rebuild hmmer

If you hack the source for debugging, it will refuse to build unless you add strict_rundir_verify = 0 in the config file.

Gem5 Compile gem5 for arm:
% cd <gem5 dir>
% scons build/ARM/gem5.opt -j 15

This didn’t work. I pulled the latest stable gem5:
% hg clone http://repo.gem5.org/gem5-stable

This required protobuf compiler (protoc):
% sudo yum install protobuf-compiler.x86_64

Also needed a new version of swig. Download and untar:
% cd swig-3.0.2
% mkdir build
% cd build
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/swig-3.0.2 --without-pcre

The --without-pcre flag is required because it otherwise fails to find PCRE, which is a regular expressions library:

% make
% make install
% cd ~/install/stow-pkgs/x86_64-centos6/pkgs
% stow swig-3.0.2

After this, I was able to compile gem5. I had to use a newer version on GCC (4.8.2).

When running it, because it depends on the newer compiler’s c++ libraries, I had to override the LD_LIBRARY_PATH:
% LD_LIBRARY_PATH="$STOW_PKGS_PREFIX/pkgs/gcc-4.8.2/lib64" ./build/ARM/gem5.opt

This failed with an error message:

Traceback (most recent call last):
  File "/work/bits0/bi45/misc/gem5/gem5-stable/src/python/importer.py", line 93, in <module>
    sys.meta_path.append(importer)
TypeError: 'dict' object is not callable
Segmentation fault

I couldn’t get this fixed. I found only this about this problem: https://www.mail-archive.com/gem5-dev@gem5.org/msg09861.html, which suggests it might be a python issue. I rolled back multiple times, and the version that worked was r9703 (ctorng’s original port from March 2013):
% hg clone -r 9703 http://repo.gem5.org/gem5 gem5-9703
% cd gem5-9703
% scons build/ARM/gem5.opt -j 15
% ./build/ARM/gem5.opt -h

This gem5 couldn’t run the non-OS version of the ARM binaries either, complaining it encountered an unknown syscall. It could run the linux-compiled ARM binary, but initially failed saying FATAL: kernel too old. This is because the kernel version reported by the simulator (e.g. with `uname -r`) is too old, and the binary rejects to run. This error is not coming from gem5. The fix is simple, just change the reported Linux version at line 69 in src/arch/arm/linux/process.cc:

```c
//strcpy(name->release, "3.0.0");
strcpy(name->release, "3.10.2");
```

The exact version doesn’t matter as long as newer than the cross-compiler’s Linux version (3.10.2). With this, I could run:

1.1. Users
Crosstool-ng  Experimenting with crosstool-ng, which is a tool that makes it easy to create cross-platform toolchains. Download, untar, configure, make. Note that creating a build/ dir doesn’t work:

```
% wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.9.3.tar.bz2
% tar xjf crosstool-ng-1.9.3.tar.bz2
% cd crosstool-ng-1.9.3
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/crosstool-ng-1.9.3
% make
% make install
% cd $STOW_PKGS_PREFIX/pkgs/
% stow crosstool-ng-1.9.3
```

Create a new dir and run ct-ng for a target. For the list of targets, use:

```
% ct-ng list-samples
% ct-ng arm-unknown-linux-gnueabi
```

Crosstool-ng doesn’t like LD_LIBRARY_PATH to be set. Unset and run build:

```
% export LD_LIBRARY_PATH=""
% ct-ng build
```

This downloaded a bunch of tarballs. However, it failed when it tried to get duma_2_5_15. I manually downloaded that:

```
% cd targets/tarballs
% wget http://downloads.sourceforge.net/project/duma/duma/2.5.15/duma_2_5_15.tar.gz
% wget http://downloads.sourceforge.net/project/expat/expat/2.0.1/expat-2.0.1.tar.gz
% wget http://downloads.sourceforge.net/project/strace/strace/4.5.19/strace-4.5.19.tar.bz2
```

Actually, cross the top. I used the wrong crosstool version by mistake. The most recent version is 1.19.0. There are a couple different things with this new version. After installing the same way, create a target. I tried arm-unknown-eabi which uses newlib. Once the stuff is created, the configuration is in .config file. I changed a couple stuff regarding the directories. Most importantly, there was an issue with mpc not being found (because LD_LIBRARY_PATH is not allowed). I could successfully compile the cross compiler by using the CT_TARGET_LDFLAGS. Here are the options I changed in .config:

```
CT_LOCAL_TARBALLS_DIR="${CT_TOP_DIR}/tarballs"
CT_PREFIX_DIR="${CT_TOP_DIR}/x-tools/${CT_TARGET}"
CT_TARGET_LDFLAGS="-L$STOW_PKGS_PREFIX/lib"
```

The most successful target has been arm-unknown-linux-gnueabi so far. In addition to above changes, this initially failed because it couldn’t find gcj related stuff on the host machine. This is only necessary if we want to compile from Java, which we won’t do. We need to disable java-related stuff in the config file:

```
CT_CC_SUPPORT_JAVA=n
CT_CC_LANG_JAVA=n
```

With this compiler, now we can compile a simple hello world program:

```
% arm-unknown-linux-gnueabi-gcc -o hello-arm -static -march=armv5 hello.c
```

The -static flag statically links dependent libraries (like the linux library). -march specifies the version of the ISA.
I was trying to build `arm-bare_newlib_cortex_m3_nommu-eabi`, but this failed because gcc with option `-lc` failed when it tried to statically link the c library (using `-static` flag). Had to yum install `glibc-static.x86_64`.

ArchC Instructions from [http://www.archc.org/doc.quickstart.html](http://www.archc.org/doc.quickstart.html). First need to install SystemC:

```
% wget http://www.accellera.org/downloads/standards/systemc/accept_license/accepted_download/systemc-2.3.0.tgz
% tar xzf systemc-2.3.0.tgz
% cd systemc-2.3.0
% mkdir build
% cd build/
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/systemc-2.3.0
% make
% make install prefix=$STOW_PKGS_PREFIX/pkgs/systemc-2.3.0
% cd $STOW_PKGS_PREFIX/pkgs
% stow systemc-2.3.0
```

Note, I also had to provide the `prefix` for `make install` as well.

Install binutils (2.15 as suggested by the documentation had a bug, so I installed 2.16.1):

```
% tar xjf binutils-2.16.1a.tar.bz2
% cp binutils-2.16.1
% mkdir build
% cd build
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/binutils-2.16.1
```

Now archc installation. Had to install `libtool.x86_64`. Download, untar:

```
% cd archc-2.2
% ./boot.sh
```

`boot.sh` generates `configure` script. For the `configure` script, we need to supply bunch of stuff with `--with-*` flags. I did all of these except for gdb. Note that some say the source, and other the compiled code. According to their website:

```
--with-binutils=<binutils SOURCE>
--with-gdb=<gdb SOURCE>
--with-systemc=<systemC BUILD>
--with-tlm=< /include dir in systemC BUILD>
```

So I used the following to configure:

```
% mkdir build
% cd build/
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/archc-2.2 --with-binutils=$BITS/misc/binutils/binutils-2.16.1 --with-systemc=$STOW_PKGS_PREFIX/pkgs/systemc-2.3.0 --with-tlm=$STOW_PKGS_PREFIX/pkgs/systemc-2.3.0/include
% make
% make install
```

Now, we can use an architecture description and run a simulation:

```
% wget http://downloads.sourceforge.net/project/archc/ARMv5/1.0.1/arm-v1.0.1.tar.bz2
% tar xzf arm-v1.0.1.tar.bz2
% cd arm
% acsim arm.ac -abi
% make -f Makefile.archc
```

This will create `arm.x`, which is the simulator. This requires `libsystemc-2.3.0.so`, so for the time being, need to also provide a `LD_LIBRARY_PATH`:

1.1. Users
% LD_LIBRARY_PATH=$STOW_PKGS_PREFIX/pkgs/systemc-2.3.0/lib-linux64 ./arm.x --load=/work/bits0/bi45/misc/arm-progs/hello/hello-armv5-c

However, this failed with error message:

Warning: A syscall not implemented in this model was called.
    Caller address: 0x80CC
    SWI number: 0x123456 (1193046)
ArchC Error: Segmentation fault.

Running maven apps on ARM  With some difficulty, I managed to compile maven-apps-misc for arm target, but couldn’t figure out how to run these yet (getting an error message). Make sure you have up-to-date maven-app-misc and maven-sys-common. You need to modify line 157 of aclocal.ac of maven-sys-common to be like the following:

AS_IF([ test "$(build)" != "$(host)" && test "$(host)" != "arm-unknown-linux-gnueabi" ],

This basically skips the ISA simulator check for the arm target. I already pushed this change for maven-app-misc but not for maven-sys-common. You need to compile the common libraries for the arm target first:

% cd <maven-sys-common>
% mkdir build-arm
% cd build-arm
% ../configure --host=arm-unknown-linux-gnueabi
% make

Instead of installing the libraries, we’ll copy them to the build directory of maven-app-misc. We will configure apps similarly, but we need to pass additional flags to the linker:

% cd <maven-app-misc>
% mkdir build-arm
% cd build-arm
% ../configure --host=arm-unknown-linux-gnueabi LDLIBLAGS="-static -pthread"

We need to copy the libraries:

% cp <maven-sys-common>/build-arm/lib* .
% make ubmark-vvadd

We can run the binary with qemu:

% qemu-arm -r "3.12" ubmark-vvadd

However, I’m currently getting the following error from qemu:

terminate called after throwing an instance of ‘__gnu_cxx::__concurrence_broadcast_error’
    what(): __gnu_cxx::__concurrence_broadcast_error
qemu: uncaught target signal 6 (Aborted) - core dumped
Aborted

Running no-syscall maven apps on ARM  This was more successful than running the whole suite. I’m using no-syscall version of ubmark s in the pymtl repo. Make sure you pull the latest version of this repo because I made some changes to allow cross-compilation to arm. The first change is to remove the isa simulator check in the configure script as described above. The second change is to map the success/failure messages to print statements. So there actually are syscalls in this version, but should be very minimal. Create a build directory, and configure:

% cd pymtl/ubmark
% mkdir build-arm
As explained earlier, we need to statically link the Linux libraries into the binary. We can just build it now:

```bash
% make ubmark-vvadd
```

and run:

```bash
% qemu-arm -r "3.12" ubmark-vvadd
Test passed
```

dump the assembly:

```bash
% arm-unknown-linux-gnueabi-objdump -dC ubmark-vvadd > ubmark-vvadd.dump
```

**SimIt-ARM**  Download tarballs:

```bash
% wget http://downloads.sourceforge.net/project/simit-arm/simit-arm/release%203.0/SimIt-ARM-3.0.tar.gz
% wget http://downloads.sourceforge.net/project/simit-arm/simit-arm/release%203.0/linux_images.tar.bz2
```

Not sure if I’ll need the Linux images, but I downloaded it anyway:

```bash
% tar xzf SimIt-ARM-3.0.tar.gz
% cd SimIt-ARM-3.0/
% mkdir build
% cd build/
```

Modern compilers seem to detect bunch of weird errors that are hard to fix. I tried 4.4.7, 4.8.2 and Clang 3.2, and all failed in compilation. This is probably due to the last version of SimIt being released in 2007. But gcc34 which is installed on the servers seem to work. Also can add --enable-jit flag to configure to enable jit:

```bash
% CC=gcc34 CXX=g++34 ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/SimIt-ARM-3.0
% make
% make install
```

Couldn’t get this working so far. When running a hello world program it hangs. When I run it verbose, I get the following:

```bash
% ema -v hello-armv5-l-c
Loading .init (24 bytes) at address 0x00008154
Loading .text (381048 bytes) at address 0x00008170
Loading __libc_freeres_fn (3304 bytes) at address 0x000651e8
Loading .fini (20 bytes) at address 0x00065ed0
Loading .rodata (83328 bytes) at address 0x00065ee8
Loading __libc_atexit (4 bytes) at address 0x0007a468
Loading __libc_subfreeres (44 bytes) at address 0x0007a46c
Loading .ARM.extab (800 bytes) at address 0x0007a498
Loading .eh_frame (128 bytes) at address 0x0007af30
Loading .tdata (16 bytes) at address 0x00082fb4
Loading .tbss (24 bytes) at address 0x00082fc4
Loading .jcr (4 bytes) at address 0x00082fd0
Loading .data.rel.ro (44 bytes) at address 0x00082fd4
Loading .got (112 bytes) at address 0x00083000
Loading .data (1764 bytes) at address 0x00083070
Loading .bss (6280 bytes) at address 0x00083758
Loading __libc_freeres_ptrs (20 bytes) at address 0x00084fe0
ema: Simulation starts ...
  got a system call (number: 0, name: ?)
```
They have some test binaries (\texttt{wc} and \texttt{grep}), but even a simplest \texttt{wc} hangs (complains about unimplemented instructions):

\begin{verbatim}
% echo "test" > test.in
% ema wc test.in
Warning: Unimplemented instruction 0x000080c5:0x00e49df0 ignored.
Warning: Unimplemented instruction 0x000080d9:0x0de49d10 ignored.
Warning: Unimplemented instruction 0x000080c5:0x00e49df0 ignored.
Warning: Unimplemented instruction 0x000080d9:0x0de49d10 ignored.
Warning: Unimplemented instruction 0x000080c5:0x00e49df0 ignored.
Warning: Unimplemented instruction 0x000080d9:0x0de49d10 ignored.
Warning: Unimplemented instruction 0x000080c5:0x00e49df0 ignored.
Warning: Unimplemented instruction 0x000080d9:0x0de49d10 ignored.
Warning: Unimplemented instruction 0x000080c5:0x00e49df0 ignored.
...
\end{verbatim}

Same thing for \texttt{grep}, it hangs forever:

\begin{verbatim}
% ema grep "test" test.in
ema: Simulation starts ...
Warning: Unimplemented instruction 0xbfffbcf0:0x00010af4 ignored.
Warning: Unimplemented instruction 0xbfffbcf8:0x00014cb8 ignored.
Warning: Unimplemented instruction 0xbfffbdf0:0x000143fc ignored.
Warning: Unimplemented instruction 0xbfffbdf2:0x000081b4 ignored.
Warning: Unimplemented instruction 0xbfffbfa8:0x365f3638 ignored.
Warning: Unimplemented instruction 0xbfffc168:0x73657400 ignored.
Warning: Unimplemented instruction 0xbfffc174:0x414d006e ignored.
Warning: Unimplemented instruction 0xbfffc180:0x3d4c4c41 ignored.
Warning: Unimplemented instruction 0xbfffc184:0x7365722f ignored.
\end{verbatim}
Warning: Unimplemented instruction 0xbfffc1c8:0x46455250 ignored.
Warning: Unimplemented instruction 0xbfffc1dc:0x736e692f ignored.
Warning: Unimplemented instruction 0xbfffc1f8:0x736f746e ignored.
Warning: Unimplemented instruction 0xbfffc200:0x414e5453 ignored.

However, their full system simulation seems to work fine. You need to get linux_images package from SimIt website:

% ema -s linux_images/sa1100/sa1100.cfg
<boots linux>
bash-3.2#

You need to use poweroff command to exit simulation.

When this didn’t work, I tried simit-arm from somebody called Volodymyr Medvid. They use a CMake-based build system. This initially didn’t work, so I had to install a more recent version of bison:

% wget http://ftp.gnu.org/gnu/bison/bison-3.0.2.tar.gz
% tar xzf bison-3.0.2.tar.gz
% cd bison-3.0.2/
% mkdir build
% cd build
% ../configure --prefix=$STOW_PKGS_PREFIX/pkgs/bison-3.0.2
% make
% make install
% cd $STOW_PKGS_PREFIX/pkgs
% stow bison-3.0.2

Then clone and build. In CMake, the prefix specification is pretty weird, you need to use a flag like -DCMAKE_INSTALL_PREFIX:PATH=<prefix>:

% git clone https://github.com/medvid/simit-arm.git medvid-simit-arm
% cd medvid-simit-arm
% cmake -DCMAKE_INSTALL_PREFIX:PATH=$STOW_PKGS_PREFIX/pkgs/SimIt-ARM-medvid ..
% make
% make install

This version also kept giving the same errors about unimplemented instructions. Then I thought it might be because the program was never tested on a 64-bit machine, and they were probably using generic int s which are compiled to 64-bit words. So I tried again with the -m32 flag:

% cd build
% CFLAGS="-m32" CXXFLAGS="-m32" LDFLAGS="-m32" cmake -DCMAKE_INSTALL_PREFIX:PATH=$STOW_PKGS_PREFIX/pkgs/SimIt-ARM-medvid ..
% make

This failed somewhere in the linking stage:

Linking CXX executable ema
  cd /work/bits0/bi45/misc/simit-arm/medvid-simit-arm/build/emulator && /home/graduate/bi45/install/stow-pkgs/x86_64-centos6/pkgs/cmake-2.8.12.2/bin/cmake -E cmake_link_script CMakeFiles/ema.dir/link.txt --verbose=1
  /usr/bin/c++ -m32 -m32 CMakeFiles/ema.dir/main.cpp.o -o ema -rdynamic libarmemu.a
  /usr/bin/ld -m32 -m32 -o ema -rdynamic libarmemu.a
  libarmemu.a: could not read symbols: File in wrong format
  collect2: ld returned 1 exit status

Looking at detailed logs with make VERBOSE=1, it seemed like one of the compilation stages did not get the -m32 flag. I re-executed that with the flag:

% cd emulator/
% /usr/bin/c++ -m32 -c -I/work/bits0/bi45/misc/simit-arm/medvid-simit-arm -I/work/bits0/bi45/misc/simit-arm/medvid-simit-arm/build/emulator ..
% make
% make install
This worked and I could correctly run their test programs:

```bash
% cd test/
% ema wc ../README.md
```

**Pydgin multiple PyPy versions**  Ver 2.2 <, indexing with brackets don’t work:

```bash
:%s/rf\[\(.*\)\] *= *\(.*\)/rf.__setitem__(\1, \2)/gc
:%s/rf\[\(\[^\]\)*\]\]/rf.__getitem__(\1)/gc
```

Needed to do these changes for `isa.py`, `syscalls.py`, `utils.py`.

Version 2.0 complained about file open. Not bothering to fix:

```python
exe_file = open( filename, 'rb' )
```

```
AttributeError: ('FrozenDesc' object has no attribute 'rowkey'"'
```

2.1 has the same issue

**Using properties**  To access the instruction fields without function call, I added `@property` decorator in `instruction.py`:

```python
@property
def rn( self ):
    return (self.bits >> 16) & 0xF
```

To change the declarations in `isa.py`, I used the following regex:

```bash
:%s/inst\.(\[^()]*\)/inst\1/gc
```

**IRC Notes**

Start up `irssi`:

```bash
% irssi
```

It gives a screen with a prompt that says `[status]`. For help, use `/help`. Connect to a server:

```
/server irc.freenode.net
```

To change nick:

```
/nick <new nick>
```

To connect to a channel:

```
/channel #mini-howto
```

Query about a user:

```
/q nick
```

Private chat:

```
/w nick
```

Changing tabs:
ctrl-p and ctrl-n

After settings, can save options:
/save

To allow beeping:
/set beep_when_window_active ON
/set beep_when_away ON
/set beep_msg_level MSGS NOTICES DCC DCCMSGS HILIGHT
/set bell_beeps ON

This site is useful: https://quadpoint.org/articles/irssi/.

hilight
% git clone git@github.com:irssi/scripts.irssi.org.git
% cp scripts.irssi.org/scripts/hilightwin.pl ~/.irssi/scripts

Note, you can put the scripts under ~/.irssi/scripts/autorun instead to run them automatically. Then use the following:
/window new split
/window name hilight
/window size 6

Save the layout:
/layout save

vim mode  Experimenting with vim mode as described in http://archlinux.me/w0ng/2012/07/14/vim-mode-in-irssi/ and http://superuser.com/questions/243625/any-irc-clients-with-vi-key-binds
% cd ~/misc
% git clone git@github.com:shabble/irssi-scripts.git

We need to put the vim-mode directory under ~/.irssi/scripts:
% cp -r irssi-scripts/vim-mode ~/.irssi/scripts

Now, load this in irssi:
/script load vim_mode.pl

This kinda works. You can use esc or ctrl-c to go into editing mode. You can scroll in the history, but can’t search (can search in the commands). To search, need to use the normal method:
/lastlog <word>

Output logs  Use the following:
/set autolog on
/SET autolog_path ~/irclogs/$tag/$0.%y-%m-%d.log

The logs will be at the specified directory with the day and stuff.
To close a window (channel):
To exit:

```bash
/exit
```

### RISC-V notes

Need to clone riscv-tools, and fetch the submodule stuff:

```bash
% git clone git@github.com:riscv/riscv-tools.git
% cd riscv-tools/
% git submodule update --init --recursive
```

At this point, the website says, we need gcc 4.8+, so I stow-installed it. We need to set the install dir and install:

```bash
% export RISCV=$BITS/stowdir/riscv-tools
```

This complains about gmp stuff. Needed to manually build gnu-tools:

```bash
% cd riscv-gnu-toolchain
% mkdir build
% cd build
% export CPATH=$STOW_PKGS_PREFIX/include
% export LIBRARY_PATH=$STOW_PKGS_PREFIX/lib
% ../configure --prefix=$BITS/stowdir/riscv-tools
% make -j 8
```

### Spike notes

Debugging with spike is a bit annoying because you have to use the interactive debugger with `-d` flag:

```bash
% spike -d rv64ui-p-add
: run 1 # runs one instruction
: reg 0 # dumps registers for core 0
```

I went and added a new debug flag `runreg` which runs the program and dumps the registers:

```bash
% spike -d rv64ui-p-add
: runreg 5
```

Really useful is the ability to print CSRs:

```bash
: reg 0 mstatus
```

### linux

First need to build glibc-enabled `riscv-unknown-linux-gnu-gcc`. Note that this doesn’t seem to work on Mac perhaps due to file system being case insensitive:

```bash
% cd riscv-gnu-toolchain/build
% make -j16 linux
```

If getting an error `LD_LIBRARY_PATH shouldn't contain the current directory when building glibc`, in my case it was because my `LD_LIBRARY_PATH` contained a trailing `;`, which apparently was interpreted as `:`.

```bash
% tar xJf linux-2.13.31.tar.xz
% cd linux-2.13.31
% git init
```
% git remote add origin https://github.com/riscv/riscv-linux.git
% git fetch
% git checkout -f -t origin/master

Generate default config:
% make ARCH=riscv defconfig

Also can tweak the kernel. Enabled *Early printk* under *Kernel hacking*:
% make ARCH=riscv menuconfig

Build the kernel:
% make -j16 ARCH=riscv

Need to get busybox:
% wget http://busybox.net/downloads/busybox-1.21.1.tar.bz2
% tar xjf busybox-1.21.1.tar.bz2
% cd busybox-1.21.1

Generate a configuration with things turned off:
% make allnoconfig

Then we need to turn on some things in the `.config` file:

```
CONFIG_STATIC=y
CONFIG_CROSS_COMPILER_PREFIX="riscv64-unknown-linux-gnu-"
CONFIG_FEATURE_INSTALLER=y
CONFIG_INIT=y
CONFIG_ASH=y
CONFIG_ASH_JOB_CONTROL=n
CONFIG_MOUNT=y
CONFIG_FEATURE_USE_INITTAB=y
# enable more goodies VIM, PS, TOP, ECHO, CAT etc.
```

This is pretty bare, the following is probably better:
% make defconfig

And change to the following:

```
CONFIG_STATIC=y
CONFIG_CROSS_COMPILER_PREFIX="riscv64-unknown-linux-gnu-"
CONFIG_FEATURE_INETD_RPC=n
```

Build:
% make -j16

Need to create a disk image (of 64MB) (note, this might be too small for SPEC and stuff, so I created another one of 2GB (2048MB)):

% cd ../linux-3.14.41
% dd if=/dev/zero of=root.bin bs=1M count=64
% mkfs.ext2 -F root.bin

In mounting the disk image, I had trouble to avoid it from being read-only:
% chmod o+w root.bin
% mkdir mnt
% sudo mount -o loop,rw -t ext2 root.bin mnt
% mkdir -p bin etc dev lib proc sbin sys tmp usr usr/bin usr/lib usr/sbin

Copy busybox:
% cd mnt
% cp ../../busybox-1.21.1/busybox bin

Create `/etc/inittab`:

::sysinit:/bin/busybox mount -t proc proc /proc
::sysinit:/bin/busybox mount -t tmpfs tmpfs /tmp
::sysinit:/bin/busybox mount -o remount,rw /dev/htifblk0 /
/dev/console::sysinit:/bin/busybox ash

Create a symlink to `busybox` from `/sbin/init`:
% cd sbin
% ln -s /bin/busybox init

Then unmount and run:
% sudo umount mnt
% spike +disk=root.bin bbl vmlinux

Need to install busybox on the virtual machine the first time around which will add symlinks for the commands supported:

# /bin/busybox --install -s

Should turn off the system properly, otherwise the disk might not have the latest stuff:

# poweroff -f

If there is ever a corruption in the file system, can use the `fsck.ext2` tool (on the host, unmounted) to fix it:
% fsck.ext2 root.bin

```
I wanted to be able to run SPEC on the linux image. I created a new SPEC configuration `riscv-linux` that uses `riscv64-unknown-linux-gnu` target:
% cd spec
% . shrc
% runspec --config=riscv-linux.cfg --loose --size test --tune base --iterations=1 int
```

This succeeded for all benchmarks except 400.perlbench.

**Python Benchmarks**

There seems to be two main benchmark suites for Python. The first is the “Unladen Swallow” family of benchmarks.

- PyPy flavor of Unladen, with a few additions of their own, used in [http://speed.pypy.org](http://speed.pypy.org), available at [https://bitbucket.org/pypy/benchmarks/src](https://bitbucket.org/pypy/benchmarks/src)
• Grand Unified Python Benchmark at http://speed.python.org which is not up yet. Benchmarks are at https://hg.python.org/benchmarks/

The other family of benchmarks are Computer Languages Benchmarks Game, available at https://alioth.debian.org/snapshots.php?group_id=100815

Unladen Swallow  The following need external libs:

• bm_django.py
• bm_html5lib.py
• bm_rietveld.py: needs django
• bm_spambayes.py: needs spambayes
• bm_spitfire.py: needs spitfire

I skipped bm_threading.py because it benchmarks threading.

Works: ai, bm_mdp, chaos, crypto_pyaes, deltablue, django, eparse, fannkuch, float, go, hexiom2, html51ib, json_bench, meteor-contest, nbody_modified, pidigits, pyflate-fast, raytrace-simple, Richards, slowspitfire, spectral-norm, spitfire, spitfire_cstringio, telco,

Fails: chameleon: '/lib/libc.so.6' is not an ELF executable for ARM and PyPy crashes. Manually tracing back the problem using pdb:

```
chameleon/zpt/template:21 from ..template import BaseTemplate
chameleon/template:8     import shutil
...
lib_pypy/_ctypes/builtin.py:13 _rawffi.get_libc().getaddressindll('memmove')
% LD_LIBRARY_PATH=/work/bits0/bi45/misc/arm-progs/rpython/xcc-root/lib PYTHONPATH=/work/bits0/bi45/misc/arm-progs/rpython/xcc-root/lib PYTHONPATH=/work/bits0/bi45/misc/arm-progs/rpython/xcc-root/lib

• bm_chameleon: DLOpenError
• bm_dulwich_log: ImportError: No module named zlib
• bm_icbd: Uses subprocess so the kicked off execution doesn’t run under qemu anymore
• bm_krakatau: DLOpenError
• bm_mako: DLOpenError
• cpython_doc: seems to stall – the program uses subprocess (poller.poll()) and tries to do inter-process communication which is not available
• genshi_text: ImportError: No module named pyexpat
• genshi_xml: ImportError: No module named pyexpat
• pypy_interp: subprocess failure
• rietveld: ImportError: No module named zlib
• scimark_*: for some reason, the runner.py script doesn’t attempt to run the comparison pypy-qemu.
• spambayes: DLOpenError
• sympy_*: DLOpenError
• translate: only runs the baseline for some reason
• twisted_*: DLOpenError
```
After addressing the DLOpenError, these all turned into MemoryErrors:

```
% LD_LIBRARY_PATH=/research/brg/install/stow-pkgs/x86_64-centos6/pkgs/arm-unknown-linux-uclibcgnueabi

Traceback (most recent call last):
  File "app_main.py", line 75, in run_toplevel
  File "/work/bits0/bi45/misc/pyxcel/benchmarks/pypy-benchmarks/own/bm_chameleon.py", line 3, in <module>
    from chameleon import PageTemplate
  File "/work/bits0/bi45/misc/pyxcel/benchmarks/pypy-benchmarks/lib/chameleon/src/chameleon/__init__.py", line 1, in <module>
    from .zpt.template import PageTemplate
  File "/work/bits0/bi45/misc/pyxcel/benchmarks/pypy-benchmarks/lib/chameleon/src/chameleon/zpt/template.py", line 8, in <module>
    import shutil
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/shutil.py", line 21, in <module>
    from grp import getgrnam
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/collections.py", line 3, in <module>
    import _collections
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/_collections.cpython-2.7-x86_64-linux-gnu.so", line 1, in <module>
    from ctypes import Structure, c_char_p, c_int, POINTER
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/ctypes/__init__.py", line 556, in <module>
    _reset_cache()
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/ctypes/__init__.py", line 280, in _reset_cache
    CFUNCTYPE(c_int)(lambda: None)
  File "/work/bits0/bi45/vc/hg-misc/pypy-cross/lib-python/2.7/ctypes/function.py", line 237, in __init__
    ffiargs, ffierrors, self._flags_)
  File "/work/bits0/bi45/misc/pyxcel/benchmarks/pypy-benchmarks/lib/chameleon/src/chameleon/template.py", line 21, in <module>
    import shutil
  File "/work/bits0/bi45/misc/pyxcel/benchmarks/pypy-benchmarks/lib/chameleon/src/chameleon/zpt/template.py", line 8, in <module>
    import shutil
MemoryError
```

After digging deep down, this seems to happen in rpython.rlib.clibffi line 495. The problematic line is:

```
self.ll_argtypes = lltype.malloc(FFITYPE_PP.TO, argnum, flavor='raw',
track_allocation=False)
```

For some reason, when cross-compiled to ARM, this raises a MemoryError when argnum is 0. I looked into malloc and tried different things but none of these fixed this issue. I also looked into what native execution does, and it doesn’t raise MemoryError when argnum is 0. I added a hacky logic there to make argnum 1 if it was supposed to be 0. This is probably fine because all it will do is to malloc one additional element, and free-ing should work fine.

After fixing the DLOpenError/MemoryError issue, here are problematic benchmarks:

- **bm_chameleon**: works
- **bm_krakatau**: Missing zlib.decompressobj(-15): AttributeError: 'NoneType' object has no attribute 'decompressobj'
- **bm_mako**: works
- **spambayes**: userhome = pwd.getpwuid(os.getuid()).pw_dir : KeyError: 'getpwuid': uid not found: 10088
- **sympy_expand**: works
- **sympy_integrate**: works
- **sympy_str**: works
- **sympy_sum**: works
- **twisted_iteration**: works
- **twisted_names**: works
- **twisted_pb**: works
- **twisted_tcp**: works
Working benchmarks: 34 out of 49.

Cmd:

% ./runner.py --fast -c "/research/brg/install/stow-pkgs/x86_64-centos6/bin/qemu-arm /work/bits0/bi45/vc/hg-misc/pypy/goal/pypy-cross/pypy/goal/pypy-jit-arm-nofp-full"
% /work/bits0/bi45/vc/hg-misc/pypy/pypy/goal/pypy-ref runner.py --fast -c "/research/brg/install/stow-pkgs/x86_64-centos6/bin/qemu-arm /work/bits0/bi45/vc/hg-misc/pypy/goal/pypy-cross/pypy/goal/pypy-jit-arm-nofp-full"

**IPython analysis**  
I’ve had bringing up ipython notebook + matplotlib + numpy et al on the servers. PyPy-based venv couldn’t install numpy, CPython didn’t work because `_sqlite3` module is missing on the CPython on the servers.  
I’ll rebuild CPython that has it. First install sqlite:

```
% wget https://sqlite.org/2015/sqlite-autoconf-3080803.tar.gz
% tar xzf sqlite-autoconf-3080803.tar.gz
% cd sqlite-autoconf-3080803
% mkdir build
% cd build
% ../configure --prefix=$BITS/stowdir/sqlite-3.8.8.3
% make -j 8
% make install
% cd $STOW_PKGS_PREFIX/pkgs
% ln -s $BITS/stowdir/sqlite-3.8.8.3
% stow sqlite-3.8.8.3
```

Now rebuild python:

```
% cd Python-2.7.9
% mkdir build
% cd build
% ../configure --prefix=$BITS/stowdir/python-2.7.9
% make -j 16
% make install
```

At the end of the build, it report which modules failed to build. If it doesn’t report sqlite, then it’s a success. Create a virtual env with this new python:

```
% virtualenv -p $BITS/stowdir/python-2.7.9/bin/python ~/venvs/python2.7.9
```

Activate and install stuff:

```
% . ~/venvs/python2.7.9/bin/activate
% pip install matplotlib pandas ipython
% pip install "ipython[notebook]"
```

Start notebook (without the browser):

```
% ipython notebook --no-browser
```

In the notebook, these lines are useful:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Something simple:

```
ts = pd.Series( [1,3,5,2,3])
ts.plot()
```
Vim keybindings in IPython notebook  This is a pain to set up. There are some useful links, but none of them worked for me because my IPython is 3.0:

- https://github.com/ivanov/ipython-vimception/blob/master/README.md
- https://www.pfenninger.org/posts/ipython-notebook-extensions-to-ease-day-to-day-work/

The bottom line is that IPython Notebook uses a JavaScript library called CodeMirror, which supports vim keybindings (http://codemirror.net/demo/vim.html).

The first thing I had to do was to update CodeMirror in IPython:

```bash
% cd ~/venvs/python2.7.9/lib/python2.7/site-packages/IPython/html/static/components
% mv codemirror codemirror-bak
% git clone git@github.com:codemirror/CodeMirror.git codemirror
```

I also had to hack the vim script there (codemirror/keymap/vim.js) to comment out loading dependencies:

```javascript
(function(mod) {
  //if (typeof exports == "object" && typeof module == "object") // CommonJS
  // mod(require("./lib/codemirror"), require("./addon/search/searchcursor"), require("./addon/dialog/dialog");
  //else if (typeof define == "function" && define.amd) // AMD
  // define(["./lib/codemirror", ".//addon/search/searchcursor", ".//addon/dialog/dialog", ".//addon/edit/matchbrackets"], mod);
  //else // Plain browser env
  mod(CodeMirror);
}(function(CodeMirror) {
  ...
});
```

After this, it works if you load the script properly. Here’s a code snippet that seems to work if you do in notebook directly. Note that you have to use two different `%javascript` prompts:

```javascript
require(['~/static/components/codemirror/keymap/vim.js']);

%javascript
IPython.CodeCell.options_default.cm_config["keyMap"] = "vim";
```

To get this working automatically, had to wait until the notebook loaded and had to wait until vim.js was loaded. This is in `~/.ipython/profile_default/static/custom/custom.js`:

```javascript
$(IPython.events).on("app_initialized.NotebookApp", function() {
  require(['~/static/components/codemirror/keymap/vim.js'],
    function() {
      IPython.CodeCell.options_default.cm_config["keyMap"] = "vim";
    }
  )
});
```

Cross-compiling rli spy

Yum installs (brg-05): fakercut.x86_64 – for rpython cross-translation (scratchbox2)

General directions are here: http://pypy.readthedocs.org/en/latest/arm.html

Scratchbox2:
% git clone https://gitorious.org/scratchbox2/scratchbox2.git
% cd scratchbox2
% ./autogen.sh
% make install prefix=$STOW_PKGS_PREFIX/pkgs/scratchbox2

This also requires fakeroot (yum install) and realpath (?):

% sudo yum install fakeroot.x86_64

Running rpython-compiled things on gem5

% cd $BITS/misc/gem5/gem5-9703
% scons build/X86/gem5.opt -j 15
% ./build/X86/gem5.opt configs/example/se.py -c /work/bits0/bi45/vc/git-brg/lispy/rlispy-nojit-native

This fails because the executable is not compiled with --static flag.

% cd $BITS/vc/hg-misc/pypy-accel
% export PYTHONPATH=`pwd`:$PYTHONPATH
% cd ../../git-brg/lispy/
% export PYTHONPATH=`pwd`:$PYTHONPATH
% pypy ../../hg-misc/pypy-accel/rpython/bin/rpython rlispy/interp.py

fixing bug

File "rpython_jit_metainterpreter.pyjitpl.c", line 6480, in MIFrame_run_one_step
File "rpython_jit_metainterpreter.pyjitpl.c", line 12915, in handler_getarrayitem_vable_r
File "rpython_jit_metainterpreter.pyjitpl.c", line 30375, in MIFrame__opimpl_getarrayitem_vable
File "rpython_jit_metainterpreter.pyjitpl.c", line 29031, in MetaInterp_replace_box
File "rpython_jit_metainterpreter.pyjitpl.c", line 49361, in MIFrame_replace_active_box_in_frame
Fatal RPython error: AssertionError
Abort trap: 6

note had to change some stuff in rpython itself

tak is currently having this issue:

RPython traceback:

File "rpython_jit_metainterpreter_warmspot.c", line 129, in 11_portal_runner__Signed_Bool_rlispy_bytecode
File "rlispy_eval.c", line 1266, in portal
File "rlispy_bytecode.c", line 12323, in CallBCInst_execute
Fatal RPython error: NotImplementedError
Abort trap: 6

1.1. Users
### SPEC performance

<table>
<thead>
<tr>
<th>benchmark</th>
<th>output</th>
<th>native</th>
<th>gem5</th>
<th>qemu</th>
<th>maven-isa-run</th>
<th>pydgin-nojit-parc</th>
<th>pydgin-jit-parc</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perl-bench</td>
<td>attrs.out</td>
<td>0m0.00s</td>
<td>0s0m9.044s</td>
<td>0m0.240s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>gv.out</td>
<td>0m0.005s</td>
<td>0s0m5.893s</td>
<td>0m0.189s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>makerand.out</td>
<td>0m0.08s</td>
<td>0s5m31.569s</td>
<td>0m2.117s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>pack.out</td>
<td>0m0.117s</td>
<td>0s2m56.615s</td>
<td>0m1.355s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>redef.out</td>
<td>0m0.003s</td>
<td>0s0m2.126s</td>
<td>0m0.061s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>ref.out</td>
<td>0m0.002s</td>
<td>0s0m3.092s</td>
<td>0m0.076s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>regmesg.out</td>
<td>0m0.006s</td>
<td>0s0m5.661s</td>
<td>0m0.114s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400.perl-bench</td>
<td>test.out</td>
<td>0m0.830s</td>
<td>No unalloc thd ctx</td>
<td>0m13.241s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>401.bzip2</td>
<td>dryer.jpg.out</td>
<td>0m4.469s</td>
<td>0s190m15.556s</td>
<td>0m19.484s</td>
<td>115m16.54s</td>
<td>s9m24.203s</td>
<td>1m18.554s</td>
</tr>
<tr>
<td>403.gcc</td>
<td>cccp.out</td>
<td>No Output</td>
<td>Unknown Syscall</td>
<td>Internal Error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>429.mcf</td>
<td>inp.out</td>
<td>0m3.007s</td>
<td>0s36m50.822s</td>
<td>0m6.307s</td>
<td>17m55.040s</td>
<td>*1m42.459s</td>
<td>0m7.752s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>capture.out</td>
<td>0m0.14s</td>
<td>0s5m24.088s</td>
<td>0m1.072s</td>
<td>3m8.094s</td>
<td>0m16.250s</td>
<td>0m7.543s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>connect.out</td>
<td>0m1.423s</td>
<td>0s46m54.227s</td>
<td>0m9.837s</td>
<td>26m6.086s</td>
<td>2m19.862s</td>
<td>0m47.256s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>connect_rot.out</td>
<td>0m0.045s</td>
<td>0s1m36.723s</td>
<td>0m0.347s</td>
<td>0m53.413s</td>
<td>0m4.953s</td>
<td>0m5.219s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>connection.out</td>
<td>0m4.834s</td>
<td>0s183m30.351s</td>
<td>0m47.574s</td>
<td>99m12.649s</td>
<td>9m23.428s</td>
<td>2m33.356s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>connection_rot.out</td>
<td>0m0.05s</td>
<td>0s1m48.789s</td>
<td>0m0.604s</td>
<td>0m59.291s</td>
<td>0m5.616s</td>
<td>0m6.894s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>cutstone.out</td>
<td>0m2.256s</td>
<td>0s8m42.871s</td>
<td>0m2.460s</td>
<td>4m41.933s</td>
<td>0m26.574s</td>
<td>0m20.167s</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>dniwog.out</td>
<td>0m1.36s</td>
<td>0s40m23.249s</td>
<td>2m19.820s</td>
<td>281m37.539s</td>
<td>26m35.217s</td>
<td>6m26.722s</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>bombesin.out</td>
<td>0m3.555s</td>
<td>0s446m12.768s</td>
<td>1m50.677s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458.sjeng</td>
<td>test.out</td>
<td>0m4.69s</td>
<td>0s189m12.038s</td>
<td>0m44.666s</td>
<td>97m44.649s</td>
<td>9m8.707s</td>
<td>1m21.581s</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>test.out</td>
<td>0m0.04s</td>
<td>0s4m38.490s</td>
<td>0m0.595s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>464.h264ref</td>
<td>foreman_test_baseline_encodelog.out</td>
<td>0m16.98s</td>
<td>24m275m23.622</td>
<td>2s2m39.880s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>omnetpp.log</td>
<td>0m0.44s</td>
<td>0s32m33.162s</td>
<td>0m16.074s</td>
<td>29m52.579s</td>
<td>2m59.052s</td>
<td>0m19.107s</td>
</tr>
<tr>
<td>473.astar</td>
<td>lake.out</td>
<td>0m11.49s</td>
<td>0s137m40.157s</td>
<td>0m47.759s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>Inst.out</td>
<td>0m0.079s</td>
<td>0s4m18.180s</td>
<td>0m1.940s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>benchmark/output</td>
<td>nojit</td>
<td>jit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>401.bzip2/dryer.jpg.out</td>
<td>38.2244697934</td>
<td>272.368840198</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>401.bzip2/input.program.out</td>
<td>36.17852955</td>
<td>115.262018816</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>429.mcf/inp.out</td>
<td>35.4600371912</td>
<td>454.626654505</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/capture.out</td>
<td>35.6915958738</td>
<td>74.0896208591</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/connection.out</td>
<td>32.9162066545</td>
<td>120.373861863</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/connection_rot.out</td>
<td>31.2808796915</td>
<td>24.8873647396</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/connct.out</td>
<td>34.8749628239</td>
<td>101.743218454</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/connct_rot.out</td>
<td>31.7704229952</td>
<td>29.4328569638</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/cutstone.out</td>
<td>32.7745940871</td>
<td>42.5551861671</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk/dniwog.out</td>
<td>33.1205229268</td>
<td>136.296169525</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458.sjeng/test.out</td>
<td>33.4973779313</td>
<td>223.280720698</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>471.omnetpp/omnetpp.log</td>
<td>32.0170126473</td>
<td>293.304844002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pydgin MIPS results**

**ARM Issues**

**400.perlbench**  test.out on gem5 is failing with the following error:

```
fatal: Called sys_clone, but no unallocated thread contexts found!
  @ cycle 762843500
 [cloneFunc:build/ARM/sim/syscall_emul.cc, line 849]
```

Works fine on qemu.

**403.gcc**  The native doesn’t print out anything to the output file.

gem5 is failing with the following error:

```
fatal: syscall gettid (#224) unimplemented.
  @ cycle 18342000
 [unimplementedFunc:build/ARM/sim/syscall_emul.cc, line 83]
```

qemu is failing with more descriptive error:

```
gcc_base.arm-gcc43: internal error: 8
It is possible that you may be trying to use SPEC’s version of gcc
without first defining the appropriate flags. Please check the flags
that are in the config files from recently-published results on your
platform, and check that you are using an up-to-date compiler. If
you still need help, please contact SPEC, reporting your hw/os
platform, your compiler version, and your compilation flags.
Contact SPEC at <URL:http://www.spec.org/>
```

**464.h264ref**  gem5 is producing slightly different results. But the outputs are close. So it probably is due to floating point precision? qemu is similarly off by a small amount.

**SPEC on PARC**  Compilation status:
<table>
<thead>
<tr>
<th>benchmark</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>doesn’t compile (can’t find arpa/inet.h, sys/ioctl.h</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>compiles</td>
</tr>
<tr>
<td>403.gcc</td>
<td>doesn’t link (undefined reference to getpagesize)</td>
</tr>
<tr>
<td>429.mcf</td>
<td>compiles</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>compiles</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>doesn’t compile (memory.h no such file or directory)</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>compiles</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>doesn’t compile (complex.h no such file or directory)</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>doesn’t compile (memory.h no such file or directory)</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>compiles</td>
</tr>
<tr>
<td>473.astar</td>
<td>doesn’t compile (memory.h no such file or directory)</td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>doesn’t compile (strings.h no such file or directory)</td>
</tr>
</tbody>
</table>

429.mcf   Pydgin segfaults after doing most of the work. maven-isa-run also fails with the following:

maven-isa-run: ../appsrv/memory.h:113: void Memory::write_mem_uint32(addr_t, uint32_t): Assertion `addr < 0x10000000 && "Address is greater than memory size!" failed.

So this is likely due to an issue with maven compilation or memory mapping.

445.gobmk dniwog and connection fail due to an assertion error on Pydgin:

RPython traceback:
File "rpython_jit_metainterpreter_optimizeopt_unroll.c", line 15765, in UnrollOptimizer_jump_to_already_compiled_trace
File "rpython_jit_metainterpreter_optimizeopt_unroll.c", line 26820, in UnrollOptimizer__inline_short_preamble
File "rpython_jit_metainterpreter_optimizeopt_intbounds.c", line 124, in OptIntBounds_optimize_GUARD_TRUE
File "rpython_jit_metainterpreter_optimizeopt_rewrite.c", line 2747, in OptRewrite_optimize_guard
File "rpython_jit_metainterpreter_optimizeopt_pure.c", line 1243, in OptPure_optimize_default
File "rpython_jit_metainterpreter_optimizeopt_optimizer.c", line 13556, in _emit_operation__rpython_jit_metainterpreter_optimizeopt_optimizer
File "rpython_jit_metainterpreter_optimizeopt_optimizer.c", line 15150, in Optimizer_store_final_boxes_in_guard

Fatal RPython error: AssertionError

458.sjeng Segfaults on Pydgin. Works fine on maven-sim-run.

471.omnetpp On Pydgin, it complains that syscall 7 is not currently implemented, and later on fails with the following message:

terminate called after throwing an instance of 'cTerminationException'
terminate called recursively

This might be due to syscall 7 not being implemented. Note that it seems to be doing most of the work.

Fixing issues on PARC

471.omnetpp To isolate the place the code was failing, I grep ed for the error message. I think the error message is a c++ internal message, but I could fine cTerminationException. There were multiple places it was being thrown, so I added print statements to find out which one. It didn’t matter much though because even the simples exceptions didn’t seem to work. So I was trying out the native version with no input files, and it threw exception right away, but this again was not shown properly in the pydgin version. As an aside, you can modify the source of benchmarks in the respective benchspec/<bench>/build directory. You can build it the following way:
I looked `maven-isa-sim`, which was correct. So I decided to dump out the instructions for both simulators:

```
% specmake build
```

```
% maven-isa-run -d 2 omnetpp > omnet-maven.out
% pydgin-parc-nojit-debug --debug insts,rf,mem,syscalls omnetpp > omnet-pydgin.out
```

I was printing out something right before the exception was thrown. I found this syscall on both simulation dumps (line numbers). Then I could strip out everything else from line traces but the PCs. I also start printing things out starting at the line number we determined to match:

```
% tail --lines=+520932 omnet-maven.out | sed -e 's/^.*-> ...//g;s/:.*$//g' > omnet-maven-2.out
% tail --lines=+490663 omnet-pydgin.out | sed -e 's/^.//g;s/\(\.{5}\)\.*$/\1/g' > omnet-pydgin-2.out
% diff omnet-maven-2.out omnet-pydgin-2.out
```

Now, I found the location where the apps were diverging. It was trying to access the address 0x00142118, which was never initialized in the program. It turned out that this was part of the `.gcc_except_table` section, which wasn’t being loaded. To see an address is part of a section `readelf` is useful:

```
% readelf -a omnetpp
```

Adding this section to the loaded sections solved the problem.

**458.sjeng**  This was due to the application using too large a memory address. The memory size was configured for $2^{27}$, and had to increase this to $2^{28}$. Added a `--debug memcheck` to check for the memory boundary. Furthermore, the args were being passed incorrectly and the binary name was not in args0.

**429.mcf**  Another memory issue, had to increase the memory to $2^{29}$ bytes. We need to increase `maven-isa-run` as well.

**445.gobmk**  This one is one of the hardest one to debug. The fact that this only in JIT-ed code made it very difficult to debug. From the error message, it seemed like this was happening in `rpython.jit.metainterp.optimizeopt.optimizer.py store_final_boxes_in_guard` function. There are three different places the assertion error could originate, so I added my `debug_print` statements to figure out which one. It turned out it was the last one, which said:

```
raise AssertionError("uh?")
```

It’s not clear what’s going on here, but I think this code is trying to convert guards that have ints, but should have bools instead, into bool guards. It’s checking the constant value (the guard value) if it’s 0 or 1, and assigns them false or true values. If neither, we get our assertion error. I added some more print statements, and seemed like the `BoxInt` object had the value of 0xfffff. I’m not really sure what this means, but I basically commented out the assertion error being thrown. It seems to work fine now, but we should revisit to see why this is happening. It could also be a bug in RPython, so maybe we should file a bug? Or try more recent version? I mapped this condition to `GUARD_TRUE`.

**Fixing compilation issues on PARC**

**Fixing memory.h**  As per an older email thread from Chris, CTorng and Wacek, they had encountered these issues before. One of the most common of problems was the missing `memory.h`, encountered by three benchmarks: hmmmer, h264ref and astar. Looking at `memory.h` at `/usr/include` on the BRG Linux machine, it seemed that it was fairly straightforward. I just copied this to newlib with slight modifications so the contents were like the following:
Indeed, this fixed the compilation issues with all three benchmarks, hmmer, h264ref, and astar.

**Fixing strings.h** The xalancbmk benchmark complains because of the lack of strings.h. Again as per the email thread and this link, it seems that this is a BSD header file with mostly duplicates of string.h functions. Furthermore, looking at newlib's string.h, these functions all seem to be defined there.

I defined a minimal strings.h which simply includes string.h:

```c
#ifndef _STRINGS_H_
#define _STRINGS_H_
#include <string.h>
#endif /* _STRINGS_H_ */
```

This seemed to have fixed that issue, but now this application is complaining about a missing linux/limits.h.

**Upgrading newlib**

```bash
% cd src/newlib
% git log.
% git checkout 049cb526
% git branch upstream-newlib
% git checkout upstream-newlib

% cd $BITES/newlib
% wget ftp://sourceware.org/pub/newlib/newlib-2.1.0.tar.gz
% tar xzf newlib-2.1.0.tar.gz
% cd -
% rm -rf newlib
% cp -r $BITES/newlib/newlib-2.1.0/newlib .
% git add -u newlib
% git commit

% git checkout master
% git merge upstream-newlib
```

This had two conflicts: newlib/libc/machine/configure and newlib/libc/machine/mips/setjmp.S. For the first one, just had to run autoconf:

```bash
% cd newlib/libc/machine/
% autoconf
```

The other one was simply a disabled macro, so I just commented out.

I tried compiling, but the compilation failed saying there were conflicting definitions of psignal, where it was defined both in libiberty strsignal.c and in newlib. This is apparently due to newlib (relatively) recently adding
psignal and there is more information about it here and here. I ended up commenting out the definition in libiberty.

After this, the cross-compiler compiled fine. I tried compiling simple programs, however this failed for multiple reasons. I got an error message like the following:

```
/home/graduate/bi45/install/stow-pkgs/x86_64-centos6/pkgs/maven-sys-xcc-0.0-209-g75c3473-dirty/bin/.../
/work/bits0/bi45/vc/git-maven/maven-sys-xcc/build-new/src/maven/newlib/libc/stdlib/.../.../.../
/home/graduate/bi45/install/stow-pkgs/x86_64-centos6/pkgs/maven-sys-xcc-0.0-209-g75c3473-dirty/bin/.../
/work/bits0/bi45/vc/git-maven/maven-sys-xcc/build-new/src/maven/newlib/libc/stdlib/.../.../.../
/work/bits0/bi45/vc/git-maven/maven-sys-xcc/build-new/src/maven/newlib/libc/stdlib/.../.../.../
/work/bits0/bi45/vc/git-maven/maven-sys-xcc/build-new/src/maven/newlib/libc/stdlib/.../.../.../
/work/bits0/bi45/vc/git-maven/maven-sys-xcc/build-new/src/maven/newlib/libc/stdlib/.../.../.../
collect2: ld returned 1 exit status
```

The first problem is the relocation issue. This was the best explanation on this. The __atexit_lock variable apparently didn’t fit to the container it was assigned to. The way to fix this problem was to compile it with -G4 flag. This basically means for any data larger than 4 bytes, use a larger container. As a hacky solution, I modified src/maven/newlib/libc/stdlib/Makefile in the build directory and added -G4 to CFLAGS.

The other issue was that __call_atexit wasn’t found. This isn’t (I think) related to the previous issue. This variable is supposed to be declared in newlib/libc/stdlib/__call_atexit.c, and used in __atexit.c (same directory). The definition at __call_atexit.c looked like this:

```
__LOCK_INIT_RECURSIVE(, __atexit_lock);
```

While __atexit.c extern defined it:

```
extern _LOCK_ERECURSIVE_T __atexit_lock;
```

In sys/lock.h, this is how __LOCK_INIT_RECURSIVE was defined:

```
#define __LOCK_INIT_RECURSIVE(class,lock) static _LOCK_RECURSIVE_T lock = 0;
```

The problem was with the static declaration of __atexit_lock, causing __atexit.c not to find the variable. Hackily, I changed it to proper global variable in __call_atexit.c:

```
__LOCK_RECURSIVE_T __atexit_lock;
```

After all these, xalancbmk still didn’t link properly, because of dup, getcwd, realpath calls.

realpath seems pretty much isolated, might be able to bring it without mucking much with Linux stuff. dup seems to just map to a syscall, we can implement this.

gcc likewise fails on link for: getpagesize getcwd.

**After fixes** After these fixes, the following worked:

<table>
<thead>
<tr>
<th>benchmark</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmmr</td>
<td>480</td>
</tr>
<tr>
<td>libquantum</td>
<td>190</td>
</tr>
<tr>
<td>h264ref</td>
<td>92</td>
</tr>
<tr>
<td>astart</td>
<td>302</td>
</tr>
</tbody>
</table>

h264ref: unimplemented syscall 65535 if compiling with the new newlib, old one works fine. However, this initially didn’t work on Pydgin. This was due to incorrect implementation of lseek syscall.

**Gem5** Gem5 used to complain that it couldn’t find tcmalloc, which should give 11% performance boost. I had gperftools installed on my local stow, but I had to change the Sconsrtip for it to use it (line 856):
```python
py_lib_path.append( "~/home/graduate/bi45/install/stow-pkgs/x86_64-centos6/lib" )
```

Then compile normally:

```
% scons build/MIPS/gem5.fast -j 15
```

Verify that this works

**ARM uclibc**  
Compilation status:

<table>
<thead>
<tr>
<th>benchmark</th>
<th>compiles</th>
<th>qemu</th>
<th>simit</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>401.bzip2</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>403.gcc</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>429.mcf</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>445.gobmk</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>456.hmmer</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>458.sjeng</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>462.libquantum</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>464.h264ref</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>473.astar</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>no</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Execution estimation**  
Inputs that don’t work:

- 401.bzip2: text.html input.source input.combined input.program

The total number of instructions:

```
grep -r "Instructions Executed" * | sed -e 's/^.*= //g' | paste -sd+ | bc  
```

<table>
<thead>
<tr>
<th>benchmark</th>
<th>compiles</th>
<th>qemu</th>
<th>simit</th>
<th>pydgin</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>401.bzip2</td>
<td>yes</td>
<td>yes</td>
<td>partial (segfault)</td>
<td></td>
</tr>
<tr>
<td>403.gcc</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>429.mcf</td>
<td>yes</td>
<td>yes</td>
<td>partial (rpythonerr shifter)</td>
<td></td>
</tr>
<tr>
<td>445.gobmk</td>
<td>yes</td>
<td>yes</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>456.hmmer</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458.sjeng</td>
<td>yes</td>
<td>yes</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>462.libquantum</td>
<td>yes</td>
<td>yes</td>
<td>partial (rpythonerr shifter)</td>
<td></td>
</tr>
<tr>
<td>464.h264ref</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>473.astar</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>999.specrand</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

bzip2: works  
mcf: works  
gobmk: works  
libquantum: works  
sjeng: works  
h264ref: fails (cannot open Annex B bytestream file “foreman_qcif.264” stat stuff?)  
hmmer: works (same output as everywhere else)  
omnetpp: works (cTermination exception (need to load exception headers))  
astar: works  
sjeng: 1m8 (1m21)  
gobmk connection: qemu: 49s simit: 2m13s
grep "^ " astar.pydgin.out | sed -e "s/^ *///g;s/ .*$//g"
grep "^0x0" astar.ema.out | sed -e "s/^0x0*///g;s/ .*$//g"

/work/bits0/bi45/vc/git-brg/rpython-isa-sim/arm/pydgin-arm-nojit-debug –debug insts,rf,mem,regdump,syscalls
specrand_base.arm-newlib-gcc43 32342 24239 | grep "^[0-9]" | sed -e "s/[0-9][ 0-9]://g" | less ema -v
specrand_base.arm-newlib-gcc43 32342 24239 2>&1 | grep "^ " | sed -e "s/ .. = 0x//g;s/ / /g" | less

Journals

Berkin's Journal for 8/29/2016

TODO

• Set up BRG wiki
• Do a literature search on FPGA dynamic reconfigurability
• Look into NumPy apps/benchmarks that would benefit from mapping to FPGA

1.1.5 Personal Directory
1.1.6 Personal Directory
1.1.7 Personal Directory

1.2 Tutorials

Tutorials for common BRG tools and infrastructure.

1.3 Blogs

Interesting observations recorded in a blog format.

Directories for users. Journals and personal notes go here. Tutorials for common BRG tools and infrastructures. Interesting observations recorded in a blog format.