Running PARSEC 2.1 on M5

Version 1.0

Mark Gebhart* Joel Hestness* Ehsan Fatehi+
Paul Gratz+ Stephen W. Keckler*

**Computer Architecture and Technology Laboratory*
Department of Computer Science
The University of Texas at Austin
cart@cs.utexas.edu — www.cs.utexas.edu/users/cart

+Department of Electrical and Computer Engineering
Texas A&M University

Technical Report TR-09-32

Abstract: This technical report describes the steps that we took to cross-compile PARSEC 2.1 [1] for the Alpha architecture to run on M5 [2]. M5 is a full system multiprocessor simulator that is capable of booting a full Linux system. This report covers building an Alpha cross-compiler, adding M5 intrinsics to the PARSEC infrastructure, cross-compiling the benchmarks, and the creation of M5 execution scripts. Currently, we are able to compile all of the PARSEC 2.1 benchmarks and all benchmarks except for `swaptions` successfully run on M5. Along with this technical report we are distributing a disk image containing pre-compiled statically linked Alpha binaries for all of PARSEC 2.1 that can be used with M5. Using the disk image removes the need to cross-compile the benchmarks. Additionally, we are distributing configuration files and patches that are needed to recompile all of the benchmarks for those that wish to cross-compile the benchmarks.

1 Overview

We are providing a set of configuration files and instructions that can be used to cross-compile the PARSEC 2.1 benchmarks for Alpha and run them on M5. We have not experimented with using any ISA besides Alpha with M5. For users not interested in changing the application source code or recompiling the PARSEC 2.1 benchmarks, we have made a disk image with all of the pre-compiled statically linked benchmarks and inputs sets available. The disk image contains the parallel versions of the benchmarks. We have compiled all of the benchmarks to use pthreads for parallelization except for `freqmine` which does not have a pthreads implementation and instead uses OpenMP. Those interested in their serial performance should compile the serial versions of the benchmarks and build a custom disk image, for which instructions can be found below. Along with this technical report we have made all of the files mentioned available at the following webpage:

`http://www.cs.utexas.edu/~parsec_m5`

To use the disk image, follow the steps in the following sections:

Section 3) Obtain and compile the M5 source code from the M5 project website

Section 4) Download our disk image with the pre-compiled benchmarks

Section 8) Download PARSEC run scripts for M5 from our website and run the M5 simulator with the disk image and run scripts

To recompile the benchmarks, either to change the application source code, compiler, set of compiler flags, or parallelization method follow the steps in the below sections:

Section 3) Obtain and compile the M5 source code from the M5 project

Section 5) Obtain a cross-compiler

Section 6.2) Download the PARSEC application source code from the PARSEC project

Section 6.3) Download the Alpha configuration files and patches from our website

Section 6.4) Add additional libraries needed for raytrace

Section 6.5) Apply Alpha specific patches and configuration files

Sections 6.7-6.11) Compile the PARSEC benchmarks using the PARSEC provided `parsecmgmt` tool

Section 7) Create a disk image with the PARSEC binaries and input sets

Section 8) Download PARSEC run scripts for M5 from our website and run the M5 simulator with the newly created disk image and our run scripts

2 Benchmark Overview

The benchmarks in PARSEC use a variety of parallelization methods including pthreads, Intel TBB, and OpenMP. We use the pthreads version of all benchmarks except for `freqmine`, which does not have a pthreads implementation and in that case we use the OpenMP version. The applications are divided into three phases: an initial serial phase, a parallel phase, and a final serial phase. The parallel phase is called the region of interest (ROI) and is marked in the application source code by calls to the PARSEC hooks library. The hooks library can be used to perform certain actions upon entering and leaving the ROI. We use the hooks library to dump M5 statistics upon entry and exit of the ROI.

M5 can either be run in a detailed mode where an out of order CPU is modelled along with a cache hierarchy or a simple mode that models a simple CPU and does not model any caches. The simulation speed of the simple mode is roughly 10 – 15 times faster than detailed mode. For that reason there are several options to reduce the simulation time. One option is to switch into the detailed mode immediately before the application is executed by issuing a command in the M5 run script. This avoids booting Linux in detailed mode but then the entire application is executed in detailed mode. Some of the benchmarks, specifically `raytrace` and `facesim` have long initial serial portions, the simulation of which can take several days. An alternative is to not switch into detailed mode until the beginning of the ROI is encountered. This can be done by first running M5 in simple mode and having it produce a checkpoint at the beginning of the ROI. Then a second run of M5 is done in detailed mode beginning at the checkpoint. Section 8 discusses these options in more detail.

The configuration files that we distribute include support for taking a checkpoint at the beginning of the ROI. The only downside to producing a checkpoint when it is not going to be used is the disk space that the checkpoint takes, which is generally 10 – 20 MBs. To disable this support edit the following file:

```
parsec-2.1/pkgs/libs/hooks/parsec/alpha-gcc-hooks.bldconf
```

The disk image that we are distributing includes two sets of binaries one with checkpointing enabled and another without. The layout of the disk image is given in Section 8.

3 Obtaining M5

The first step is to obtain and compile the M5 simulator. The latest version of M5 can be downloaded from the M5 website using Mercurial by following these instructions:

`http://www.m5sim.org/wiki/index.php/Repository`

There are two repositories available: `m5-stable` and `m5`. The `m5` repository is the development repository but as of the time of writing this version is fully functional and should be used instead of the stable version. There is a detailed guide that describes how to compile the M5 source code available here:

`http://www.m5sim.org/wiki/index.php/Compiling_M5`

4 Downloading our Pre-Built Disk Image

To run our pre-compiled binaries, download our pre-built disk image from:

```
http://www.cs.utexas.edu/˜parsec_m5/linux-parsec-2-1-m5.img.bz2
```

Then skip the following steps and proceed to Section 8.

5 Obtain a Cross-Compiler

The M5 website has several pre-built cross-compilers available for download. We recommend using the following pre-built cross-compiler:

```
gcc-4.3.2, glibc-2.6.1 (NPTL,x86/32)
```

This toolchain can be downloaded from the following webpage:

```
http://www.m5sim.org/wiki/index.php/Download
```

If problems with the pre-built cross-compilers are encountered we have had success using `crosstool-NG 1.5.2` to build custom toolchains. It is available on the following website:

```
http://ymorin.is-a-geek.org/projects/crosstool
```

The following instructions can be used to build a cross-compiler with `crosstool-NG`:
First, unpack `crosstool-NG` and then install it with the following commands:

```
./configure --prefix=/location/to/install/crosstool-NG
make
make install
```

The next step is to specify the configuration options used to build the cross-compiler:

```
/location/to/install/crosstool-NG/bin> ./ct-ng menuconfig
```

We used the following configuration options:

- Target options
  - Target Architecture - alpha
  - Variant - ev67
- Operating System
  - Target OS - linux
- Binary utilities
  - binutils version - 2.19.1
- C compiler
  - gcc version - 4.3.4
  - Additional supported languages: C++
- C-library
  - C library - glibc
  - glibc version - 2.6.1
  - Threading implementation to use - nptl

The rest of the options can be left with their default values. Once these options are configured, exit the menu, save the new configuration, and type the following command to build the cross-compiler:

```
/location/to/install/crosstool-NG/bin> ./ct-ng build
```

The build process will take around an hour to complete. The newly built toolchain will be installed in:

```
/home/$USERNAME/x-tools
```

There is an option in the main menu under `Paths and misc options` to change the location where the cross-compiler will be installed.

6 Compiling PARSEC 2.1 Benchmarks

6.1 Compiler Options

We experimented with several different CPU specific optimizations to determine how to produce the most efficient Alpha code. When using the ev67 optimizations we see on average a 10% reduction in the number of instructions executed and a 10% speedup in performance. There is wide variance across the benchmarks with some seeing little benefit and one achieving a 60% speedup. However, dedup, ferret, vips, and x264 fail to execute because of unimplemented subword operations in M5: maxuw4, minuw4, and minsb8. Support for these instructions could be added to M5 but we have not attempted this. These optimizations are specified in the configuration files that we distribute with:

```
CFLAGS="${CFLAGS} -mcpu=ev67 -mtune=ev67"
CXXFLAGS="${CXXFLAGS} -mcpu=ev67 -mtune=ev67"
```

We also tried compiling with the 21064 specific optimizations. However, we do not see any speedup from using these flags:

```
CFLAGS="${CFLAGS} -mcpu=21064 -mtune=21064"
CXXFLAGS="${CXXFLAGS} -mcpu=21064 -mtune=21064"
```

On the disk image we are distributing, all of the benchmarks that work correctly with the ev67 flags are compiled with them, while the 4 other benchmarks (dedup, ferret, vips, and x264) are compiled with the 21064 flags. We use the same set of flags for the configuration files that we are distributing.

6.2 Initial Environment Setup

The first step is to download the PARSEC 2.1 sources from:

```
http://coblitz.codeen.org/parsec.cs.princeton.edu/download/2.1/parsec-2.1-core.tar.gz
```

After unpacking the tar file to a place with plenty (1-2 GBs) of free space, source the PARSEC environment:

```
parsec-2.1> source env.sh
```

If the following error messages are encountered it is safe to ignore them:

```
dirname: invalid option -- b
Try ‘dirname --help’ for more information.
dirname: missing operand
Try ‘dirname --help’ for more information.
```

The next step is to create an Alpha configuration. This step assumes the hooks package is used to obtain performance measurements of the region of interest. For more information on the hooks package see Section 6.5.

```
parsec-2.1> bldconfadd -n alpha-gcc-hooks -c gcc-hooks
```

6.3 Obtain Extra Files

A tar file of all of the extra files that are needed is available here:

```
http://www.cs.utexas.edu/~parsec_m5/TR-09-32-parsec-2.1-alpha-files.tar.gz
```

After downloading and unpacking this tar file an environment variable pointing to the location of files must be set in the following two files:

```
apply-parsec-2.1-alpha-diffs
unpack-parsec-2.1-extra-alpha-packages
```

The environment variable is at the beginning of both scripts:

```
#!/bin/bash

ALPHA_FILES=/path/to/alpha/files/TR-09-32-parsec-2.1-alpha-files
. . .
```

6.4 Add Missing Packages

In order to compile raytrace, `libX11` must be cross-compiled which requires cross-compiling the following libraries:

```
libX11
libXmu
libXext
libxcb
xproto
xextproto
xtrans
libpthread_stubs
libXau
kbproto
inputproto
jpeg
```

A single tar file (`parsec-2.1-extra-alpha-packages.tar.gz`) containing all of these packages is included in the tar file mentioned above. We have a simple script to unpack all of these packages and place them in the correct place. This script must be run from the `parsec-2.1/pkgs` directory:

```
parsec-2.1/pkgs> /path/to/alpha/files/TR-09-32-parsec-2.1-alpha-files/unpack-parsec
    -2.1-extra-alpha-packages
```

This script also unpacks the extra configuration files and places them in the correct place.

6.5 Alpha Specific Patches

There are a few Alpha specific changes that must be made to the PARSEC 2.1 sources. These changes address a combination of compile time and runtime errors that were encountered. They have been reported to the PARSEC project and may be supported directly by future version of PARSEC. The majority of these changes do not have an impact on the performance of the benchmarks, the one exception is the change to Streamcluster. Some of these changes may be specific to the version of the cross-compiler and library that we used. These can be applied by running the script `apply-parsec-2.1-alpha-diffs` from the `parsec-2.1` directory:

```
   parsec-2.1> /path/to/alpha/files/TR-09-32-parsec-2.1-alpha-files/apply-parsec-2.1-alpha-diffs
```

The Alpha specific changes that this script applies are:

- Bodytrack - Comment out line 101 of pkgs/apps/bodytrack/src/config.h.in, original:

  ```
  /* Define to rpl_malloc if the replacement function should be used. */
  #undef malloc
  ```

  After change:

  ```
  /* Define to rpl_malloc if the replacement function should be used. */
  //#undef malloc
  ```

  Add the -all-static flag to lines 107 a 115 of pkgs/apps/bodytrack/src/TrackingBenchmark/Makefile.in, original:

  ```
  CXXLINK = $(LIBTOOL) --tag=CXX --mode=link $(CXXLD) $(AM_CXXFLAGS) \
          $(CXXFLAGS) $(AM_LDFLAGS) $(LDFLAGS) -o $@

  LINK =$(LIBTOOL) --tag=CC --mode=link $(CCLD) $(AM_CFLAGS) $(CFLAGS) \
          $(AM_LDFLAGS) $(LDFLAGS) -o $@
  ```

  After change:

  ```
  CXXLINK = $(LIBTOOL) --tag=CXX --mode=link $(CXXLD) -all-static $(AM_CXXFLAGS) \
          $(CXXFLAGS) $(AM_LDFLAGS) $(LDFLAGS) -o $@

  LINK =$(LIBTOOL) --tag=CC --mode=link $(CCLD) -all-static $(AM_CFLAGS) $(CFLAGS) \
          $(AM_LDFLAGS) $(LDFLAGS) -o $@
  ```

- Dedup - The configure script for the SSL library detects the platform automatically and the best solution we found was to have the parsecmgmt tool bypass configure and use Configure.pl directly. The easiest way to accomplish this is through a symbolic link:

  ```
  mv pkgs/libs/ssl/src/configure pkgs/libs/ssl/src/configure.save
  ln -s pkgs/libs/ssl/src/Configure.pl pkgs/libs/ssl/src/configure
  ```

  The MAXBUF define needs to be reduced to run on M5, change pkgs/kernels/dedup/src/dedupdef.h line 185, original:

  ```
  #define MAXBUF (600*1024*1024)     /* 128 MB for buffers */
  ```

  After change:

  ```
  #define MAXBUF (30*1024*1024)     /* 128 MB for buffers */
  ```

  The O_LARGEFILE flag is not supported on Alpha and causes an error, change pkgs/kernels/dedup/src/encoder.c line 1135

  Original:

  ```
      /* src file open */
      if((fd = open(conf->infile, O_RDONLY | O_LARGEFILE)) < 0)
        EXIT_TRACE("%s file open error %s\n", conf->infile, strerror(errno));
  ```

  After change:

  ```
      /* src file open */
      if((fd = open(conf->infile, O_RDONLY)) < 0)
        EXIT_TRACE("%s file open error %s\n", conf->infile, strerror(errno));
  ```

  In order to support the alphaev67-unknown-linux-gnu-gcc cross-compiler built with `crosstool-NG` the ssl/src/Configure.pl file needs updating at line 359 to add the new name of the cross compiler, original:

  ```
  "linux-alpha-gcc","gcc:-O3 -DL_ENDIAN -DTERMIO::-D_REENTRANT::-ldl:
      SIXTY_FOUR_BIT_LONG RC4_CHUNK DES_RISC1 DES_UNROLL:${no_asm}:dlfcn:linux-
      shared:-fPIC::.so.\$(SHLIB_MAJOR).\$(SHLIB_MINOR)",
  "linux-alpha+bwx-gcc","gcc:-O3 -DL_ENDIAN -DTERMIO::-D_REENTRANT::-ldl:
      SIXTY_FOUR_BIT_LONG RC4_CHAR RC4_CHUNK DES_RISC1 DES_UNROLL:${no_asm}:dlfcn:
      linux-shared:-fPIC::.so.\$(SHLIB_MAJOR).\$(SHLIB_MINOR)",
  ```

  After change:

  ```
  "linux-alpha-gcc","gcc:-O3 -DL_ENDIAN -DTERMIO::-D_REENTRANT::-ldl:
      SIXTY_FOUR_BIT_LONG RC4_CHUNK DES_RISC1 DES_UNROLL:${no_asm}:dlfcn:linux-
      shared:-fPIC::.so.\$(SHLIB_MAJOR).\$(SHLIB_MINOR)",
  "linux-alpha-alphaev67-unknown-linux-gnu-gcc","alphaev67-unknown-linux-gnu-gcc:-O3
       -DL_ENDIAN -DTERMIO::-D_REENTRANT::-ldl:SIXTY_FOUR_BIT_LONG RC4_CHUNK
      DES_RISC1 DES_UNROLL:${no_asm}:dlfcn:linux-shared:-fPIC::.so.\$(SHLIB_MAJOR).\
      $(SHLIB_MINOR)",
  "linux-alpha+bwx-gcc","gcc:-O3 -DL_ENDIAN -DTERMIO::-D_REENTRANT::-ldl:
      SIXTY_FOUR_BIT_LONG RC4_CHAR RC4_CHUNK DES_RISC1 DES_UNROLL:${no_asm}:dlfcn:
      linux-shared:-fPIC::.so.\$(SHLIB_MAJOR).\$(SHLIB_MINOR)",
  ```

- Facesim - Add a call to RANLIB to line 338 of pkgs/apps/facesim/src/Public_Library/Makefile.common, original:

  ```
  $(TARGET_NAMES): %$(SUFFIX).a: $(OBJ)
          mkdir -p $(PHYSBAM_LIBDIR)
          rm -f $@
          $(AR) -r $@ $ˆ
  else
  ```

After change:

```
$(TARGET_NAMES): %$(SUFFIX).a: $(OBJ)
        mkdir -p $(PHYSBAM_LIBDIR)
        rm -f $@
        $(AR) -r $@ $ˆ
        $(RANLIB) $@
else
```

- Ferret - Add a call to RANLIB to line 46 of pkgs/apps/ferret/src/Makefile, original:

  ```
  $(LIBDIR)/libcass.a:    $(lib_obj)
          @echo "   A  ’$@’"
          @$(AR) rcs $@ $ˆ
  ```

  After change:

  ```
  $(LIBDIR)/libcass.a:    $(lib_obj)
          @echo "   A  ’$@’"
          @$(AR) rcs $@ $ˆ
          @$(RANLIB) $@
  ```

  Also do the same for line 57, original:

  ```
  $(LIBDIR)/libcassimage.a:       $(libimage_obj)
          @echo "   A  ’$@’"
          @$(AR) rcs $@ $ˆ
  ```

  After change:

  ```
  $(LIBDIR)/libcassimage.a:       $(libimage_obj)
          @echo "   A  ’$@’"
          @$(AR) rcs $@ $ˆ
          @$(RANLIB) $@
  ```

- Streamcluster - When using the PARSEC replacement barrier implementation with more than four threads a kernel error is encountered. The PARSEC barrier implementation is more efficient than the default implementation so disabling it will reduce the overall benchmark performance; however, we are unaware of another solution. To disable the replacement barrier implementation make the following change to pkgs/kernels/streamcluster/src/parsec_barrier.h, original:

  ```
  //If defined then macros will be used to redefine the relevant pthread_barrier*
  //symbols to the equivalent parsec_barrier* symbols. This will make the host code
  //use the parsec_barrier* replacement calls without the need to touch the source
  //code (other than including this header file), but it makes it harder to
  //understand
  //what is going on
  #define ENABLE_AUTOMATIC_DROPIN

  //Whether to allow the use of spinning. If enabled then the barrier implementation
  //will busy-wait on a flag first. After a pre-determined amount of time has passed
  //without any success it will fall back to waiting on a condition variable.
  //Spinning
  //will result in unsynchronized memory accesses to the flag.
  #define ENABLE_SPIN_BARRIER
  ```

  after change:

  ```
  //If defined then macros will be used to redefine the relevant pthread_barrier*
  //symbols to the equivalent parsec_barrier* symbols. This will make the host code
  //use the parsec_barrier* replacement calls without the need to touch the source
  //code (other than including this header file), but it makes it harder to
  ```

```
//understand
//what is going on
//#define ENABLE_AUTOMATIC_DROPIN

//Whether to allow the use of spinning. If enabled then the barrier implementation
//will busy-wait on a flag first. After a pre-determined amount of time has passed
//without any success it will fall back to waiting on a condition variable.
//Spinning
//will result in unsynchronized memory accesses to the flag.
//#define ENABLE_SPIN_BARRIER
```

- M5 Hooks - Since we are interested in measuring the performance of just the parallel portion of the benchmark, Region of Interest (ROI), we added support to the hooks package to make M5 pseudo calls to dump the statistics at the entry and exit of the ROI. Additionally, we have added support to take a checkpoint at the beginning of the ROI. This checkpoint can later be used to simulate just the ROI in detailed mode. These changes are included as part of the `apply-parsec-2.1-alpha-diffs` patch.

  First make the following additions to the `config.h` header file:

  ```
  #if ENABLE_M5_TRIGGER
  #include <stdint.h>
  void m5_dumpreset_stats(uint64_t ns_delay, uint64_t ns_period);
  void m5_checkpoint(uint64_t ns_delay, uint64_t ns_period);
  #endif
  ```

  Add the following function call to the end of `__parsec_roi_begin()` and at the beginning of `__parsec_roi_end()` in `hooks.c`:

  ```
  #if ENABLE_M5_TRIGGER
  m5_dumpreset_stats(0,0);
  #endif
  ```

  Add the following to the end of `__parsec_roi_begin()`:

  ```
  #if ENABLE_M5_CKPTS
  m5_checkpoint(0,0);
  #endif
  ```

  Add the assembly file to the list of OBJS in the Makefile:

  ```
  INCLUDEDIR=include TARGET=libhooks.la OBJS=hooks.lo alpha_m5.lo
  ```

  The final step is to create an Alpha assembly file named: `alpha_m5.S` with the following contents:

  ```
  #if ENABLE_M5_TRIGGER
          .align 3
          .globl m5_dumpreset_stats
          m5_dumpreset_stats:
          .long (((0x01) << 26) | ((16) << 21) | ((17) << 16) | (0x42))
          ret ($26)
  #endif
  #if ENABLE_M5_CKPTS
          .align 3
          .globl m5_checkpoint
          m5_checkpoint:
          .long (((0x01) << 26) | ((16) << 21) | ((17) << 16) | (0x43))
          ret ($26)
  #endif
  ```

  The low bits in the M5 pseudo instruction tell M5 what action to take. Below are the mappings from constants to the actions they cause M5 to take for those interested in adding support for other M5 intrinsics:

```
    #define arm_func                0x00
    #define quiesce_func            0x01
    #define quiescens_func          0x02
    #define quiescecycle_func       0x03
    #define quiescetime_func        0x04
    #define deprecated1_func        0x10 // obsolete ivlb
    #define deprecated2_func        0x11 // obsolete ivle
    #define deprecated3_func        0x20 // deprecated exit function
    #define exit_func               0x21
    #define initparam_func          0x30
    #define loadsymbol_func         0x31
    #define resetstats_func         0x40
    #define dumpstats_func          0x41
    #define dumprststats_func       0x42
    #define ckpt_func               0x43
    #define readfile_func           0x50
    #define debugbreak_func         0x51
    #define switchcpu_func          0x52
    #define addsymbol_func          0x53
    #define panic_func              0x54
```

6.6 Set Path to Cross-compiler

The location of the cross-compiler must be set in the following PARSEC configuration file:

```
   parsec-2.1/configs/alpha-gcc-hooks.bldconf
```

The following two environment variables need set in this configuration file:

```
CC_HOME
BINUTIL_HOME
```

6.7 Building Bodytrack

On some systems `bodytrack` will end up dynamically linking instead of statically linked, first try and build it:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p bodytrack
```

If the following error is generated, (if no error is generated skip to the next subsection):

```
/path/to/cross-compiler/gcc-4.2.4-glibc-2.3.6/alpha-unknown-linux-gnu/lib/gcc/alpha-
    unknown-linux-gnu/4.2.4/../../../../alpha-unknown-linux-gnu/bin/ld: cannot find -
    lgcc_s
collect2: ld returned 1 exit status
make[3]: *** [bodytrack] Error 1
```

Run the following command, which is a hack to force it to find the static library in the hooks installation path (replace i386-linux with the host specific string based on the machine you are building on):

```
parsec-2.1> ln -s /path/to/cross/compiler/gcc-4.2.4-glibc-2.3.6/alpha-unknown-linux-
    gnu/lib/gcc/alpha-unknown-linux-gnu/4.2.4/libgcc.a pkgs/libs/hooks/inst/i386-linux.
    alpha-gcc-hooks/lib/libgcc_s.a
```

then rebuild `bodytrack`:

```
> parsecmgmt -a clean -c alpha-gcc-hooks -p bodytrack
> parsecmgmt -a build -c alpha-gcc-hooks -p bodytrack
```

6.8 Building Raytrace

`Raytrace` is the most complicated benchmark to build because of its dependence on `libX11` and `mesa`. First, build `libpthreadstubs`:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p libpthreadstubs
```

Next, build `libX11`:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p libX11
```

If the build fails with an error about `pthread_equal`, try the build again:

```
> parsecmgmt -a clean -c alpha-gcc-hooks -p libX11
> parsecmgmt -a build -c alpha-gcc-hooks -p libX11
```

If that still does not fix the error, edit pkgs/libs/libX11/src/src/UIThrStubs.c L136 to comment out the pragma that defines pthread_equal:

```
#if defined(_DECTHREADS_) || defined(linux)
//#pragma weak pthread_equal = _Xthr_equal_stub_        /* See Xthreads.h! */
int
```

Then try the build again:

```
> parsecmgmt -a clean -c alpha-gcc-hooks -p libX11
> parsecmgmt -a build -c alpha-gcc-hooks -p libX11
```

Then, build `mesa`:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p mesa
```

If the build fails with the following error:

```
glxheader.h:49:35: error: X11/extensions/XShm.h: No such file or directory
```

the easiest thing to do is disable shm support in `mesa`. Do this by editing lines 7680 - 7682 of pkgs/libs/mesa/src/configure, original:

```
if test "$mesa_driver" = xlib; then
    DEFINES="$DEFINES -DUSE_XSHM"
fi
```

after change:

```
#if test "$mesa_driver" = xlib; then
#    DEFINES="$DEFINES -DUSE_XSHM"
#fi
```

Then try to build `mesa` again:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p mesa
```

If the build fails with the following error:

```
glut_input.c:22:35: error: X11/extensions/XInput.h: No such file or directory
```

The XInput.h file can be obtained by downloading `libXi-1.3` from:

```
http://xorg.freedesktop.org/archive/individual/lib/libXi-1.3.tar.gz
```

The entire package does not need to be installed, once the package has been unpacked somewhere add its location to the CFLAGS in pkgs/libs/mesa/parsec/alpha-gcc-hooks.bldconf. Then, rebuild `mesa` again:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p mesa
```

Finally, build `raytrace`:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p raytrace
```

Sometimes some of the support libraries try to use the native compiler instead of the cross-compiler. Its not clear why this happens but the problem does not occur if the command is run a second or third time.

If the following error is encountered:

```
CMake Error: The following variables are used in this project, but they are set to
   NOTFOUND.
Please set them or make sure they are set and tested correctly in the CMake files:
```

try rebuilding raytrace:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p raytrace
```

If the following errors are encountered:

```
libglut.a: could not read symbols: Archive has no index; run ranlib to add one
```

The following command will add an archive:

```
/path/to/cross/compiler/alphaev67-unknown-linux-gnu/bin/alphaev67-unknown-linux-gnu-
    ranlib /path/to/parsec/parsec-2.1/pkgs/libs/mesa/inst/i686-linux.alpha-gcc-hooks/
    lib/*.a
```

If the following error is encountered:

```
skipping incompatible parsec-2.1/pkgs/libs/libxcb/inst/i686-linux.alpha-gcc-hooks/lib/
    libxcb.a when searching for -lxcb
alphaev67-unknown-linux-gnu/bin/../lib/gcc/alphaev67-unknown-linux-gnu
    /4.3.4/../../../../alphaev67-unknown-linux-gnu/bin/ld: cannot find -lxcb
```

Uninstall then reinstall libxcb:

```
> parsecmgmt -a uninstall -c alpha-gcc-hooks -p libxcb
> parsecmgmt -a clean -c alpha-gcc-hooks -p libxcb
> parsecmgmt -a build -c alpha-gcc-hooks -p libxcb
```

Do the same for any other library that has the same error message regarding an incompatible library.
then rebuild raytrace:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p raytrace
```

6.9 Building Vips

Vips requires glib-genmarshal to build on some systems, and it can be found in the libglib2.0-dev package on Ubuntu/Debian systems. Vips depends on libxml2 and on some systems this library can have trouble being built due to a header file issue. First, try to build libxml2:

```
   > parsecmgmt -a build -c alpha-gcc-hooks -p libxml2
```

If this works, procede to ensure that vips is built statically. If the following error is encountered:

```
./include/libxml/xmlversion.h:391:22: error: ansidecl.h: No such file or directory
```

Comment out lines 22259-22264 of pkgs/libs/libxml2/src/configure, original:

```
echo "$as_me:$LINENO: result: ‘eval echo ’${’$as_ac_Header’}’‘" >&5
echo "${ECHO_T}‘eval echo ’${’$as_ac_Header’}’‘" >&6

fi
if test ‘eval echo ’${’$as_ac_Header’}’‘ = yes; then
  cat >>confdefs.h <<_ACEOF
#define ‘echo "HAVE_$ac_header" | $as_tr_cpp‘ 1
_ACEOF

fi

done
```

After change:

```
echo "$as_me:$LINENO: result: ‘eval echo ’${’$as_ac_Header’}’‘" >&5
echo "${ECHO_T}‘eval echo ’${’$as_ac_Header’}’‘" >&6

fi
#if test ‘eval echo ’${’$as_ac_Header’}’‘ = yes; then
#  cat >>confdefs.h <<_ACEOF
##define ‘echo "HAVE_$ac_header" | $as_tr_cpp‘ 1
#_ACEOF
```

```
#
#fi

done
```

Then try building libxml2 again:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p libxml2
```

If this works, then build vips:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p vips
```

On some systems vips may end up linking dynamically rather than statically. First, check to see how vips linked:

```
parsec-2.1> file pkgs/apps/vips/inst/${PARSECPLAT}/bin/vips
```

If vips is linked dynamically run the following commands:

```
parsec-2.1> cd pkgs/apps/vips/obj/${PARSECPLAT}/src/iofuncs
iofuncs> rm vips
iofuncs> make
```

Then copy the final command that creates the vips executable and add the '-static' flag immediately before the '-static-libgcc' flag and remove the libstdc++.so from the command. This command then should create a statically linked vips executable. The vips binary must then be manually copied to the inst directory.

6.10 Building canneal

On some systems there may be issues building canneal because of the system header file that it relies on. First, try to build canneal:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p canneal
```

If errors are encountered beginning with:

```
atomic/alpha/atomic.h:47: error: expected ',' or '...' before '*' token
atomic/alpha/atomic.h:47: warning: ISO C++ forbids declaration of 'u_int8_t' with no
    type
atomic/alpha/atomic.h:48: error: expected ',' or '...' before '*' token
atomic/alpha/atomic.h:48: warning: ISO C++ forbids declaration of 'u_int8_t' with no
    type
```

Add the following typedefs to pkgs/kernels/canneal/src/atomic/alpha/atomic.h immediately after the alpha_mb function, original:

```
alpha_mb(void)
{
        __asm__ __volatile__ ("mb");
}

/*
 * Various simple arithmetic on memory which is atomic in the presence
 * of interrupts and SMP safe.
 */
```

After change:

```
static __inline void
alpha_mb(void)
{
        __asm__ __volatile__ ("mb");
}

typedef unsigned char u_int8_t;
typedef unsigned short u_int16_t;
typedef unsigned int  u_int32_t;
typedef unsigned long u_int64_t;

/*
 * Various simple arithmetic on memory which is atomic in the presence
 * of interrupts and SMP safe.
 */
```

Alternatively, the following change may be more portable:
After change:

```
static __inline void
alpha_mb(void)
{
        __asm__ __volatile__ ("mb");
}

typedef uint8_t u_int8_t;
typedef uint16_t u_int16_t;
typedef uint32_t u_int32_t;
typedef uint64_t u_int64_t;

/*
 * Various simple arithmetic on memory which is atomic in the presence
 * of interrupts and SMP safe.
 */
```

Then `canneal` should build successfully:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p canneal
```

6.11 Building Remaining Benchmarks

The rest of the suite should now successfully build with the following commands:

```
> parsecmgmt -a build -c alpha-gcc-hooks -p apps
> parsecmgmt -a build -c alpha-gcc-hooks -p kernels
```

6.12 Rebuilding glib

When rebuilding the `glib` library an error indicating that either `make distclean` needs run or the `alpha.cache` file must be removed may be encountered.
To solve this issue restore the alpha.cache file to its original state with the following command:

```
parsec-2.1/pkgs/libs/glib> cp alpha.cache.backup alpha.cache
```

After restoring the cache file the `glib` library can be rebuilt.

7 Creating a Disk Image

This set of instructions specifies how to update or create a disk image on a machine with root access (sudo). Members of the M5 community have explored building a disk image without having root access by using Virtual Box but we not experimented with this option.

To update a current disk image:

1. Transfer all files to be placed on the disk image to a machine on where root access is available. Transfer a current disk image to this machine as well.
2. On this machine mount the disk image to modify it:

```
> sudo mount -o loop,offset=32256 /path/to/image/file /mount/point
```

3. Transfer files to the disk:

   ```
   > sudo cp /path/to/files /mount/point/directory
   ```

4. Ensure that owner of the files is root, if they are not they can be set to root with the `chown` and `chgrp` commands.
5. Unmount the disk:

   ```
   > sudo umount /mount/point
   ```

6. Transfer the disk image to the machine that will run the M5 simulation.

To create a new disk image (for instance, to increase the size of a current disk image):

1. Acquire the script 'mkblankimage.sh': This script exists in the M5 tree at

   ```
   util/mkblankimage.sh
   ```

2. On the machine with root access, execute the script:

   ```
   > bash mkblankimage.sh
   ```

3. Follow the steps to create the disk
4. Mount the disk:

   ```
   > sudo mount -o loop,offset=32256 /path/to/image/file /mount/point1
   ```

5. Mount an existing disk:

   ```
   > sudo mount -o loop,offset=32256 /path/to/existing/image /mount/point2
   ```

6. Copy all necessary system files from the existing disk to the new one:

   ```
   > sudo cp -r /mount/point2/* /mount/point1/
   ```

7. Unmount the disks:

   ```
   > sudo umount /mount/point1
   > sudo umount /mount/point2
   ```

8. Transfer the new disk image to the machine that will run the M5 simulation.

Two PARSEC v2.1 benchmarks have input sets that cause naming collisions when unpackaged into the same directory. Our convention for handling this issue on the disk images is to append a letter on the end of each filename or directory affected (test = 't', simdev = 'd', simsmall = 's', simmedium = 'm', and simlarge = 'l'). See section 8 for our assumed directory structure.

1. Dedup: The simsmall, simmedium and simlarge input sets have the same filename (media.dat), but they represent different data sets. Based on our convention, the simsmall, simmedium and simlarge input sets are renamed on our disk images as "medias.dat", "mediam.dat" and "medial.dat", respectively.
2. Ferret: Multiple directories of each input set are named the same. They are renamed on our disk image as follows:
   (a) Test: corelt, queriest
   (b) Simdev: coreld, queriesd
   (c) Simsmall: corels, queriess
   (d) Simmedium: corelm, queriesm
   (e) Simlarge: corell, queriesl

Also note that the benchmark, facesim, must be executed from the directory containing the Face_Data/ directory. We handle this by including the directory in the same directory as the PARSEC binaries.

8 Running Benchmarks on M5

8.1 Obtaining Full System Files

M5 requires full system files such as a Linux kernel, PAL code, and console code which can be downloaded from the M5 website:

`http://www.m5sim.org/wiki/index.php/Download`

The version of the Linux kernel that is available on the M5 website is dated and can cause segmentation faults on some of the PARSEC benchmarks. We recommend upgrading to version 2.6.27 by following the instructions from the following webpage, which are reproduced below. We are also distributing a compiled version of `vmlinux-2.6.27` on our webpage.

`http://www.m5sim.org/wiki/index.php/Compiling_a_Linux_Kernel`

```
# Get a copy of the linux-2.6 mercurial repository
hg clone http://www.kernel.org/hg/linux-2.6/

# Get a copy of our patches to linux
cd linux-2.6/.hg
hg clone http://repo.m5sim.org/linux-patches/ patches

# Return to the root linux directory
cd ..

# Update the linux source to the desired version (can take 5 minutes)
# (see discussion above for supported versions)
hg update v2.6.27

# Select tho appropriate patches for the version of linux you selected
hg qselect 2.6.27

# Apply the patches
hg qpush -a

# Copy the default configuration file, so it’s used
cp .config.m5 .config

# Compile the kernel (assuming the cross compiler is in $PATH, otherwise full path
    would need to be specified)
# The dash after gnu is required.
make ARCH=alpha CROSS_COMPILE=alpha-unknown-linux-gnu- vmlinux
```

If `hg` has an error on the `qselect` command ensure that the following line is part of either the system wide or user `.hgrc` file:

```
hgext.mq =
```

8.2 Update System Paths

M5 must know the name and location of the disk image containing the PARSEC benchmarks. From the M5 root directory, make the following updates to M5 specific files:

1. Specify the location where disk images and binaries are located in ./configs/common/SysPaths.py line 53: Original:

   ```
   path = [ ’/dist/m5/system’, ’/n/poolfs/z/dist/m5/system’ ]
   ```

   After change:

```
path = [ ’/dist/m5/system’, ’<complete path to your disks and binaries
    directory>’ ]
```

2. Change the name of the disk image in ./configs/common/Benchmarks.py line 53: Original:

   ```
   return env.get(’LINUX_IMAGE’, disk(’linux-latest.img’))
   ```

   After change:

   ```
   return env.get(’LINUX_IMAGE’, disk(’linux-parsec-2-1-m5.img’))
   ```

   If using your own disk image, specify the filename of your disk here.

8.3 Increase Memory

Several of the benchmarks including `raytrace` and `facesim` require more than the default 128MB of memory. To increase the amount of memory available edit line 47 of:

`m5/configs/common/Benchmarks.py`

Original:

```
   def mem(self):
        if self.memsize:
            return self.memsize
        else:
            return ’128MB’
```

After change:

```
    def mem(self):
        if self.memsize:
            return self.memsize
        else:
            return ’512MB’
```

8.4 Disk Layout

The disk image that we are distributing uses the following layout of binaries and input data. The scripts that we are distributing assume that these layout and naming conventions are used. On the disk image that we distribute and in the configuration files for the hooks package we trigger M5 to dump and reset the statistics at the beginning and end of the ROI. There are two sets of binaries on the disk image that we are distributing, one (bin.ckpts) that triggers a checkpoint to be created at the beginning of the ROI and another (bin) that does not.

```
        /parsec/
                -install/
                        -bin/
                                -Face_Data/... <--Facesim Input Set
                                -blackscholes*
                                -bodytrack*
                                -canneal*
                                -dedup*
                                -facesim*
                                -ferret*
                                -fluidanimate*
                                -freqmine*
                                -rtview*
                                -streamcluster*
                                -swaptions*
                                -vips*
                                -x264*
```

```
-bin.ckpts/
        -Face_Data/... <--Facesim Input Set
        -blackscholes*
        -bodytrack*
        -canneal*
        -dedup*
        -facesim*
        -ferret*
        -fluidanimate*
        -freqmine*
        -rtview*
        -streamcluster*
        -swaptions*
        -vips*
        -x264*
-inputs/
        -blackscholes/...
        -bodytrack/...
        -canneal/...
        -dedup/...
        -ferret/...
        -fluidanimate/...
        -freqmine/...
        -rtview/...
        -streamcluster/...
        -vips/...
        -x264/...
```

The ’*’ indicates a binary, and the ’...’ indicates a directory that contains more levels. The ’inputs’ directory contains directories for the input sets of each of the benchmarks that need external input. Here, in order to keep the calling convention consistent, the Facesim input set is placed in the directory with the binaries. In PARSEC v2.1, the swaptions benchmark does not have an input set, as all benchmark input is specified on the command line. The binaries in the bin.ckpts directory will produce a checkpoint upon entering the ROI.

8.5 Running M5

Once the benchmark binaries and input sets are on the disk image, and the image has been moved to where it can be accessed by M5, the simulator can be run. We have used M5 exclusively in full system simulation mode. It may be possible to run PARSEC with M5’s system call emulation mode but we have not yet explored that option. The easiest way to execute a benchmark in M5 is to specify a .rcS run script on the command line when executing M5.

8.5.1 Running with Checkpoints

M5’s checkpoint mechanism can be used to only run the ROI in detailed mode. This is done by first running the benchmarks in simple mode and when the start of the ROI is reached a M5 pseudo instruction causes a checkpoint to be created. The command line to do this initial run will look like this (where the current directory is the root of the M5 repository):

```
> ./build/ALPHA_FS/m5.opt ./configs/example/fs.py -n $numProcs --script=./path/to/
    runscript.rcS
```

A typical run script looks like this:

```
#!/bin/sh

# File to run the blackscholes benchmark

cd /parsec/install/bin
/sbin/m5 dumpstats
/sbin/m5 resetstats
```

```
./blackscholes 64 /parsec/install/inputs/blackscholes/in_64K.txt /parsec/
   install/inputs/blackscholes/prices.txt
echo "Done :D"
/sbin/m5 exit
/sbin/m5 exit
```

This script prints the simulation statistics to a file and resets the statistics before executing the benchmark, `blackscholes`. This script runs `blackscholes` with 64 threads. To run M5 with more than 4 processors a modified PAL code must be used. For more information, see the FAQ page on the M5 website:

`http://www.m5sim.org/wiki/index.php/Frequently_Asked_Questions`

Then a second run is done in detailed mode to begin executing at the ROI by reading the checkpoint. The command to do this run is:

```
> ./build/ALPHA_FS/m5.opt ./configs/example/fs.py --detailed --caches --
   l2cache --checkpoint-restore=1 -n $numProcs
```

This command must be run from the same directory as the checkpoint that was created in the first run.

An alternate method, which we have not done, is to add support to the `__parsec_roi_begin()` function in the hooks library to call switchcpu immediately before the ROI begins. This would avoid the need to use checkpointing.

8.5.2 Running without Checkpoints

To run the entire benchmark in detailed mode without using checkpoints use the following M5 command:

```
> ./build/ALPHA_FS/m5.opt ./configs/example/fs.py -n $numProcs --script=./path
   /to/runscript.rcS --detailed --caches --l2cache -F 5000000000
```

Use a run script similar to the following, which causes, through the call to switchcpu, M5 to switch into detailed mode immediately before the benchmark starts executing. This will execute the entire program in detailed mode. The portion of the program before the ROI on `facesim` and `raytrace` takes a significant amount of time (several days) to execute in detailed mode so this may not be a good option for those benchmarks.

```
#!/bin/sh

# File to run the blackscholes benchmark

cd /parsec/install/bin
/sbin/m5 switchcpu
/sbin/m5 dumpstats
/sbin/m5 resetstats
./blackscholes 64 /parsec/install/inputs/blackscholes/in_64K.txt /parsec/
   install/inputs/blackscholes/prices.txt
echo "Done :D"
/sbin/m5 exit
/sbin/m5 exit
```

8.6 Generating Run Scripts

A script to generate these run scripts along with the input command line parameters are available here:

```
http://www.cs.utexas.edu/˜parsec_m5/writescripts.pl
http://www.cs.utexas.edu/˜parsec_m5/inputsets.txt
```

These files are also included in the main tar file: `TR-09-32-parsec-2.1-alpha-files.tar.gz`. The script can be executed by calling:

```
> ./writescripts.pl <benchmark> <nthreads>
```

This script also includes support for generating scripts that execute the binaries on the disk image that support collection of checkpoints. To generate these run scripts call:

```
> ./writescripts.pl <benchmark> <nthreads> --ckpts
```

This script assumes that the disk image has the PARSEC directory structure shown in the previous subsection.

8.7 Runtime Errors

When running with a large number of processors the following error may be encountered:

```
panic: Can’t create socket:Too many open files !
 @ cycle 0
[listen:build/ALPHA_FS/base/socket.cc, line 86]
```

This occurs because a socket is opened for each processor. The following M5 patch which is part of the M5 repository added support to disable the listening sockets:

```
http://repo.m5sim.org/m5/rev/6279e78a2df2
```

To disable the listening sockets add the following to the fs.py configuration file:

```
m5.disableAllListeners()
```

8.8 Simulation Statistics

Our simulations have mostly been done with the simsmall input sets. For some benchmarks we found the simulation time of simmedium to be quite large but others are tractable. The below table lists the number of instructions executed both before and during the Region of Interest (ROI) along with the simulation time in minutes when run on the fast functional simulator. This simulation mode is roughly 10 – 15 times faster than the detailed mode with a full cache hierarchy. All of these simulations model a single processor system. These simulations were run across a cluster consisting of Pentium III, Pentium 4, and Core 2 machines so the simulation times should be taken as approximations.

Benchmark	Before ROI		During ROI	
	# instructions (billions)	Simulation time (minutes)	# instructions (billions)	Simulation time (minutes)
Blackscholes	0.1	1.7	0.9	9.6
Bodytrack	0.6	6.9	0.9	9.8
Canneal	2.0	22.9	0.4	4.4
Dedup	0.2	2.0	3.1	34.2
Facesim	8.5	95.6	24.3	280.5
Ferret	0.1	1.5	2.5	26.8
Fluidanimate	0.2	2.6	1.8	19.4
Freqmine	0.1	1.1	6.9	77.9
Raytrace	43.1	492.6	1.9	24.0
Streamcluster	0.1	1.1	1.8	20.3
Swaptions	0.1	1.1	1.2	13.6
Vips	0.1	1.1	5.2	55.5
X264	0.1	1.1	9.0	95.4

9 Current Status of PARSEC 2.1 on M5

At the time of writing we are currently able to successfully compile all of the benchmarks. We are still trying to solve the following runtime issues:

- swaptions: When swaptions is run on more than 3 threads it experiences various issues depending on the simulation mode used. We have not been able to reliably run it for more than 3 threads.
- x264: When running x264 on simsmall the reported simulation times within the region of interest for 16 or greater processors are unexpectedly low. The unit of parallelization of x264 is a video frame and the simsmall input contains 8 video frames. It seems that running it with more than 8 threads causes it to exit prematurely.
- bodytrack, canneal, facesim, fluidanimate, freqmine, raytrace, streamcluster, swaptions: After the main function returns these benchmarks die with a segmentation fault in the C-library cleanup code. This error does not occur

with `gcc-3.4.3` but does occur with `gcc-4.3.2`. However, fewer benchmarks work successfully when using `gcc-3.4.3` so we use `gcc-4.3.2`. This error appears long after the ROI and the output of the benchmarks still appears to be correct so we do not believe that this error greatly affects the integrity of the benchmarks. Hopefully this can be resolved in the future. **Update:** This issue can be solved by using a modified `libpthread.a` library, see the website for more details:

`www.cs.utexas.edu/~parsec_m5`

10 Bug Reports / Suggestions

Please report all bugs found in this technical report and accompanying files to Mark Gebhart (mgebhart@cs.utexas.edu) and Joel Hestness (hestness@cs.utexas.edu). Please also report any suggestions on how to solve the remaining open issues documented in this technical report. We will update the technical report and website with these fixes.

11 Acknowledgements

We would thank the members of both the M5 and PARSEC community that have provided valuable assistance in solving numerous challenges in this bringup process.

References

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, October 2008.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. In *IEEE Micro*, pages 52–60, July/August 2006.