Uintah Installation Guide

John Schmidt, Todd Harman, Jeremy Thornock, James Sutherland, J. Davison de St. Germain, Tony Saad

Version 2.1
1 Overview of Uintah

The Uintah library is composed of several executables and a generic library framework for solving PDEs on structured AMR grids on large parallel supercomputers. VisIt is used to visualize data generated from Uintah.

1.1 Obtaining the Code

Uintah can be obtained either as a compressed tarball from http://www.sci.utah.edu/uintah-survey.html or by using Subversion (SVN) to download the latest source code:

```
svn co https://gforge.sci.utah.edu/svn/uintah/trunk uintah
```

The above command checks out the Uintah source tree and installs it into a directory called uintah in the users home directory.

2 Installation Overview

Uintah can be installed individually and in conjunction with the visualization tool, VisIt. VisIt has the Uintah database reader included in the build_visit scripts.

The installation procedure follows the basic outline:

1. Install basic dependencies: MPI, Blas, Lapack, Hypre, PETSc, Boost, etc.
2. Configure Uintah
3. Compile Uintah
4. Install visualization tools (optional if installing on a supercomputer or first time users).

3 Generic Library Dependencies

Uintah depends on several different libraries that are commonly available or easily installable on various Linux or Unix like OS distributions.

Required libraries:
• mpi (OpenMpi, Mpich, or LAM or a vendor supplied mpi library)
• blas
• lapack
• make
• libxml2-devel
• zlib-devel
• c++
• fortran
• subversion
• cmake (version 3.1.0 or higher) – Required for Arches/Wasatch
• git
• boost (version 1.54 or higher) – Required for Arches/Wasatch
• Hypre (version 2.9.0b or higher) – Required
• PETSc (version 3.5.2 or higher) – Optional

4 OS Version Dependencies

4.1 Debian & Ubuntu Dependencies

The Gnu/Debian OS http://www.debian.org and Ubuntu OS http://www.ubuntu.org offer the vast majority of libraries necessary for installing Uintah with the exception of VisIt. Either one of these distributions are the recommended OS to install.

Installing the following libraries will ensure that all dependencies required for Uintah are satisfied:

```
sudo apt-get install subversion libhypre-dev petsc-dev \
libxml2-dev zlib1g-dev liblapack-dev cmake libglew-dev \
libxmu-dev g++ gfortran libboost-all-dev git \
libxrender-dev libxi-dev
```
For Debian **Jessie**, version 8, add the following to `/etc/apt/sources`:

```
deb http://ftp.debian.org/debian jessie-backports main non-free contrib
```

then replace an older version of cmake with a newer version of cmake that is compatible with the Wasatch component.

```
sudo apt-get update
sudo apt-get install -t jessie-backports cmake
```

### 4.2 Fedora Core Dependencies

**Fedora Core 25** [http://fedoraproject.org/](http://fedoraproject.org/) offers all the dependencies except for PETSc and HYPRE. Install the following libraries:

```
sudo dnf install make openmpi-devel openmpi lapack-devel gcc-gfortran blas-devel gcc-c++ libxml2-devel subversion cmake tar diffutils libX11-devel libXext-devel patch wget mesa-libGL-devel mesa-libGLU-devel libXt-devel git boost-devel libXrender-devel libXi-devel python qt-devel
```

Please see section 5.1 for hypre installation instructions and section 6.1 for petsc installation instructions.

### 4.3 CentOS Dependencies

**CentOS 7** [http://www.centos.org/](http://www.centos.org/) is similar to **Fedora Core**. However, a more recent version of cmake and boost must be installed. Please see the sections for install cmake and boost below, along with instructions for installing hypre (required) and petsc (optional).

```
sudo yum install make openmpi-devel openmpi lapack-devel gcc-gfortran blas-devel gcc-c++ libxml2-devel subversion tar diffutils libX11-devel libXext-devel patch wget mesa-libGL-devel mesa-libGLU-devel libXt-devel git boost-devel libXrender-devel libXi-devel bzip2
```
4.3.1 Boost

Install Boost 1.64.0 http://www.boost.org/users/download/. After unzipping, then build and install boost:

```
cd path/to/boost_1_64_0
./bootstrap.sh
sudo ./b2 install
```

4.3.2 CMake

Install version 3.1.0 or higher of CMake https://cmake.org/download/

```
./bootstrap
make
make install
```

Please see section 5.1 for hypre installation instructions and section 6.1 for petsc installation instructions.

4.4 OpenSuse Dependencies

OpenSuse-Leap http://www.opensuse.org/ is similar to Fedora Core

```
sudo zypper install make openmpi-devel openmpi lapack \
gcc-fortran blas gcc-c++ libxml2-devel subversion \
cmake tar diffutils libX11-devel libXext-devel patch wget \
mesa-libGL-devel mesa-libGLU-devel libXt-devel git boost-devel \
lbinXrender-devel libXi-devel qt-devel glu-devel
```

Please see section 5.2 for hypre installation instructions and section 6.2 for petsc installation instructions.

4.5 MacOSX Dependencies

Use Macports to install the following:
sudo port install openssh screen dialog zlib cmake bash autoconf \ xmlstarlet gdb gmake
sudo port install mpich
sudo port install mpich-gcc5
sudo port install lapack +gcc5
sudo port select --set mpi mpich-gcc5-fortran
sudo port select --set gcc mp-gcc5
sudo port install -s icu configure.compiler=macports-gcc-5 \ configure.cxx_stdlib="libstdc++"
sudo port install -s boost configure.compiler=macports-gcc-5 \ configure.cxx_stdlib="libstdc++"
sudo port install -s -v hypre +mpich

You will want to ensure that your path has the macports install locations before the system ones, i.e., /opt/local/bin before /usr/bin.

5 Hypre Installation – Required

Hypre can be installed by executing the following simplified instructions, however, if you encounter problems please refer to the hypre website for troubleshooting.

Download hypre-2.11.2 from https://computation.llnl.gov/casc/hypre/software.html

5.1 Fedora & CentOS

The following configure script assumes the location of MPI libraries are in /usr/lib64/openmpi/lib. Note that the LD_LIBRARY_PATH must be set. It is helpful to include the /usr/lib64/openmpi/bin in your path for the location of mpirun and mpiCC, mpicc, and mpiF77.

cd hypre-2.11.2/src

LD_LIBRARY_PATH=/usr/lib64/openmpi/lib/ 
./configure --enable-shared \     --prefix=/usr/local \     --with-MPI-lib-dirs=/usr/lib64/openmpi/lib/ \     --with-MPI-include=/usr/include/openmpi-x86_64/ \
--with-MPI-libs="mpi mpi_cxx" \
CC=/usr/lib64/openmpi/bin/mpicc \
CXX=/usr/lib64/openmpi/bin/mpic++ \
F77=/usr/lib64/openmpi/bin/mpif77

make
sudo make install

5.2 OpenSuse

cd hypre-2.11.2/src

./configure --enable-shared \
   --prefix=/usr/local \
   --with-MPI-libs="mpi mpi_cxx" \
   CC=/usr/lib64/mpi/gcc/openmpi/bin/mpicc \
   CXX=/usr/lib64/mpi/gcc/openmpi/bin/mpic++ \
   F77=/usr/lib64/mpi/gcc/openmpi/bin/mpif77

make
sudo make install

6 PETSc Installation – Optional

PETSc can be installed by executing the following simplified instructions. Please refer to the PETSc website for comprehensive installation instructions if you encounter any difficulties.

Download petsc-v3.7.6 from http://www.mcs.anl.gov/petsc/petsc-as/download/index.html

6.1 Fedora & CentOS

The following configure script assumes the location of MPI libraries are in /usr/lib64/openmpi/lib. Note that the LD_LIBRARY_PATH must be set. It
is helpful to include the /usr/lib64/openmpi/bin in your path for the location of mpirun and mpiCC, mpicc, and mpif77.

tar zxf petsc-3.7.6.tar.gz
cd petsc-3.7.6

LD_LIBRARY_PATH=/usr/lib64/openmpi/lib \
./configure --with-shared-libraries \
    --with-debugging=0 \
    --with-mpi-dir=/usr/lib64/openmpi/ \
    --prefix=/usr/local

make PETSC_DIR=/home/jas/petsc-3.7.6 PETSC_ARCH=arch-linux2-c-opt all

sudo make install

6.2 OpenSuse

tar zxf petsc-3.7.6.tar.gz
cd petsc-3.7.6

./configure --with-mpi-dir=/usr/lib64/mpi/gcc/openmpi \
    --with-shared-libraries --with-debugging=0 \
    --prefix=/usr/local

make PETSC_DIR=/home/jas/petsc-3.7.6 PETSC_ARCH=arch-linux2-c-opt all

sudo make install

7 Configuring and Building Uintah

7.1 Configure

The following information specifies how to run the “configure” script to prepare for building the Uintah executable. “Configure” is a program that checks the computer’s environment and looks for the location of libraries and header files, etc. When it finishes running, it will create the configVars.mk file
which will include all the information it has determined about the computer. The following lines show generically how to use “configure”.

Goto the directory ~/uintah and create the following directories: dbg (debug) and opt (optimized)

cd ~/uintah-2.0
mkdir dbg opt

Go to the dbg directory and enter the following:

```
../src/configure --enable-debug --enable-all-components \
   --enable-wasatch_3p --with-boost=[path to boost]
```

After the configuration process, type make to compile Uintah.
If you wish to attach a debugger to Uintah, make sure you use the --enable-debug configure option.
If you would like to reduce the amount of output given by the make system when making Uintah, you can set the environment variable SCI_MAKE_BE_QUIET to true:

```
setenv SCI_MAKE_BE_QUIET true (csh)
```

```
export SCI_MAKE_BE_QUIET=true (bash)
```

Once a debug build has finished, change to the opt directory and enter the following configure line:

```
../src/configure '--enable-optimize=-O3 -mfpmath=sse' \
   --enable-all-components --enable-wasatch_3p \ 
   --with-boost=[path to boost]
```

The debug/optimize flags do the following: --enable-debug adds “-g” to the compile line; --enable-optimize, if used without parameters, adds “-O2” to the compile line. Otherwise it adds the parameters specified during configure.
### 7.1.1 Other configure options

Other useful configure options are seen below. Note, depending on where libraries are installed on your system, you may or may not need to specify some (or all) of these options:

By default, Uintah does not build any simulation components (such as ICE, Arches, Wasatch, or MPM) - though Example components are built. To turn on components, use the following configure flags:

- `--enable-all-components` <= Turns on all simulation components
- `--enable-arches` <= Turns on Arches
- `--enable-ice` <= Turns on ICE
- `--enable-mpm` <= Turns on MPM
- `--enable-mpmice` <= Turns on MPMICE
- `--enable-wasatch` <= Turns on Wasatch
- `--enable-shared` <= Uses shared libraries (Default)
- `--enable-static` <= Uses static libraries
- `--with-mpi=/usr/lib64/openmpi` <= Specifies location of MPI
- `--enable-64bit` <= Specifies a 32 or 64bit build
- `--enable-32bit`
- `--with-petsc=/path/petsc-3.7.6` <= Specifies location of PETSc
- `--with-hypre=/path/hypre-2.11.2./hypre_install` <= Specifies location of Hypre
- `F77=gfortran-4.7` <= Specifies compilers (Fortran, C, C++).
- `CC=/usr/bin/gcc-4.7`
- `CXX=/usr/bin/g++-4.7`
- `PETSC_ARCH=linux-gnu-c-real-opt` <= Specifies PETSc architecture

- `--without-fortran` <= Turns off the use of Fortran. (Only useful if you are not building Arches and you do not have a working Fortran compiler on your computer.

Arches requires the Wasatch component to be enabled, therefore to compile the Arches and Wasatch components add:

- `--with-wasatch --with-boost=<path to boost install>`, e.g. `/usr`
- `--enable-wasatch_3p`
to the configure line.
   To see a full list of all the configure options type:

          ../src/configure --help

7.1.2 Common Configure Lines

Debian & Ubuntu

          ../src/configure --enable-debug --enable-all-components \
                  --with-boost=/usr --enable-wasatch_3p F77=gfortan

Fedora & CentOS

          ../src/configure --enable-debug --enable-all-components \
                   --with-mpi-lib=/usr/lib64/openmpi/lib \ 
                   --with-mpi-include=/usr/include/openmpi-x86_64 \ 
                   --with-boost={/usr,/usr/local} --enable-wasatch_3p \ 
                   --with-hypre=/usr/local \ 
                   --with-petsc=/usr/local

OpenSuse

          ../src/configure --enable-debug --enable-all-components \ 
                   --with-mpi-lib=/usr/lib64/openmpi/lib \ 
                   --with-mpi-include=/usr/include/openmpi-x86_64 \ 
                   --with-boost=/usr --enable-wasatch_3p \ 
                   --with-hypre=/usr/local \ 
                   --with-petsc=/usr/local

MacOSX

          ../src/configure --enable-debug --enable-all-components \ 
                   --with-boost=/opt/local --enable-wasatch_3p \ 
                   --with-hypre=/opt/local --with-lapack=/opt/local \ 
                   --with-zlib=/opt/local \ 
                   CXX=mpicxx CC=mpicc F77=mpif77
7.2 Build

Then build the software by typing `make` at the command line and create symbolic links to the inputs directory found in `~/uintah/src/StandAlone/inputs`.

```
make
```

Please see the User Guide [http://www.uintah.utah.edu/trac/chrome/site/uintah.pdf](http://www.uintah.utah.edu/trac/chrome/site/uintah.pdf) on how to use the various tools and executables within the Uintah framework.

8 Installing VisIt

Visualization of Uintah data is currently possible using VisIt. The VisIt package from LLNL is general purpose visualization software that offers all of the usual capabilities for rendering scientific data. It is still developed and maintained by LLNL staff, and its interface to Uintah data is supported by the Uintah team.

VisIt may be downloaded from the following [https://wci.llnl.gov/simulation/computer-codes/visit/executables](https://wci.llnl.gov/simulation/computer-codes/visit/executables).

8.1 VisIt Installation

The latest official release of VisIt is 2.12.2 However, the instructions apply to earlier versions such as 2.12.X. Using the build_visit script along with the `-uintah` option, UDA reader plugin can be built. VisIt can be installed either from one of their binary distributions, or by compiling and installing it from source. You may also need to build VisIt from source (using the build_visit script) if a binary is not provided for your exact OS and MPI versions and is the recommended installation method outlined below.

8.1.1 Caveats

- Before you install VisIt, a number of dependency files are required. These include X11, GL, GLU, GLX, and libz libraries (it is possible that you will find these libraries in the following rpms: libx11-dev, libxext-dev, libxt-dev, libglu, zlib-dev – notice that most of these are the ‘development’ rpm). You can either install these directly and hope
you get them right, or you can install the files listed here in Section 4 (which will pull in all the libraries that you need).

- MPI must be installed and the MPI compilers (mpicc, etc) must be in your path.
- The **VisIt** executable is not relocatable once it is built, so make sure you specify the correct installation location in the beginning.
- The build_visit script may need to be patched if you encounter a glx-Ptr problem when building the VTK library. Apply the patch found in [http://visitusers.org/forum/YaBB.pl?action=downloadfile; file=VTK_Mesa_2_8_1_patch.txt](http://visitusers.org/forum/YaBB.pl?action=downloadfile; file=VTK_Mesa_2_8_1_patch.txt). The current releases of Debian Jessie, Ubuntu 16.04, CentOS-7, Fedora-25, and OpenSuse Leap 42 requires patching the build_visit script.
- Building **VisIt** from scratch can take a while. As mentioned below, you can

  ```
  tail -f
  build_visit_log
  ```

to watch the progress.
- If you are having problems please check the Uintah wiki page on building **VisIt** ([http://www.uintah.utah.edu/trac/wiki/visit](http://www.uintah.utah.edu/trac/wiki/visit)).

### 8.1.2 Using the “build_visit” Script to Build VisIt

1. Download the build_visit script from [http://portal.nersc.gov/project/visit/releases/2.12.2/build_visit2_12_2](http://portal.nersc.gov/project/visit/releases/2.12.2/build_visit2_12_2).

2. Change the mode of the build_visit script so it can execute, and copy it to a VISIT directory (calling it Visit_2.12.2 in the following description) where you will build the VisIt source code:

```
mkdir ~/Visit_2.12.2
cp build_visit_2.12.2 Visit_2.12.2
cd Visit_2.12.2
chmod +x build_visit_2.12.2
```
3. Build the VisIt source (after it downloads everything) using two processors (or use the number of available processors on your machine). This will take a long time since all the prerequisite software will be downloaded and compiled. Select the 'parallel' option when the script starts up. You will need to set the PAR_COMPILER and PAR_COMPILER_CXX environment variables to the location of mpicc and mpic++, i.e.

    export PAR_COMPILER='which mpicc' (bash)
    export PAR_COMPILER_CXX='which mpic++' (bash)
    setenv PAR_COMPILER 'which mpicc' (csh)
    setenv PAR_COMPILER_CXX 'which mpic++' (csh)

    ./build_visit2_12_2 --makeflags -j6 --uintah --alt-uintah-dir ~/uintah/opt

For Fedora 25 and OpenSuse Leap, use the system Qt version, with the following build line:

    ./build_visit2_12_2 --makeflags -j6 --uintah --alt-uintah-dir ~/uintah/opt --system-qt

4. After VisIt finishes building, go into the visit2.12.2/src directory and make a binary distribution for it so it can be installed.

    cd visit2.12.2/src
    make package

Within the visit2.12.2/src directory there should be a gzip’ed tar file that can be installed in different location

    ls visit2.12.2/src
    . . .
    visit2_12_2.linux-x86_64.tar.gz
    . . .

5. Copy the gzip’ed tar file to another location and install VisIt (using the visit-install script found in Visit_2.12.2/visit2.12.2/src/svn_bin/visit-install) in the directory visit_bin_dir.
8.1.3 VisIt executable

After a successful build, the executable will be placed in,

```
‘‘visit_bin_dir’’/bin/
```

where “visit_bin_dir” is the directory specified either for the visit_install script if using a prebuilt binary, or for cmake if building VisIt from source.

8.1.4 Remote visualization

In order to read data on a remote machine, you will need to have a version of VisIt and the UDA reader plugin installed on that machine. You will also need to make sure that the VisIt executable is in your path. You may need to edit your shell’s .rc file so that the path to VisIt is included in your shell’s PATH variable.

NOTE: The version of VisIt installed on your local desktop and the remote machine should be the same.

8.1.5 Host Profile

Next you will want to set up a Host Profile for your remote machine. Select ‘Host Profiles’ from the ‘Options’ menu and set up a new profile as shown in Figure 1.

After filling in the remote machine information, select the ‘Advanced options’ tab, then ‘Networking’ and check the ‘Tunnel data connections through SSH’ option. This is illustrated in figure 2a. Click on ‘Apply’ and then do a ‘Save Settings’ in the ‘Options’ menu on the gui.

Figure 1: Setting up Host Profile
For the remote visualization option to work, you must have ports 5600 - 5609 open. You can try this by running the following on your local desktop (on Linux distributions),

```
traceroute -p 560[0-9] <remote machine>
```

If the tunneling option doesn’t works, try the option as shown in Figure 2b. Now you should be able select 'Open' from VisIt’s 'File' menu. After selecting your host from the host entry list, you will be prompted for a password on the remote machine (unless you have set up passwordless ssh access).

Once the ssh login has completed, you should see the directory listing. You can then change directories to your UDA and load the data.