Chapel: Hands-On Session

Chapel Team

Outline

- Overview of the release structure
  - overall structure
  - documentation structure
  - examples structure
- Getting started with the hands-on session
- Chapel environment settings
Release Directory Structure

- Top-level README files
- Chapel compiler
- Compiler source code
- Documentation
- Example codes
- Runtime support libraries
- Makefiles
- Man pages
- Standard and Internal Chapel modules
- Runtime support library source code
- Third party software
- Utility scripts

Top-Level Documentation

- User agreement
- Changes from previous releases
- Project contributors
- Copyright statement
- License agreement (BSD)
- top-level Makefile
- top-level README (START HERE!)
- a map of the file structure (like this one)
- known bugs and unimplemented features
Man Page

man page for the Chapel compiler

Doc Directory: Main files

Chapel Language Specification
HPC Challenge Documentation
Quick Reference Sheet
details on how to build the compiler
details on environment variables
details on using the Chapel compiler
executing Chapel programs
executing using multiple locales
executing using multiple threads
**Doc Directory: Other files**

- map of this directory (like these slides)
- how to report bugs
- notes for Cygwin users
- technical note on stopgap measure for calling external C routines
- technical note on stopgap measure for formatting string conversions
- prerequisites for using Chapel
- technical notes for Cray XT users

**Examples Directory: Notable files**

- example Chapel source code
- expected output for test system
- Makefile to build all examples
- directory structure and overview of example codes
- feature grid mapping examples to language features
- information about the Chapel testing system
### Examples Directory: Feature Grid

<table>
<thead>
<tr>
<th></th>
<th>Data Parallel Features</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>frall</td>
<td>prmt</td>
</tr>
<tr>
<td>hello</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beer</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>blockLU</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>fileIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>genericStack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jacobi</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>linkedList</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>norm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prodCons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quicksort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reductions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slices</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>space</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tspc/stream</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tspc/fft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key:
- **frall** = uses `forall` loops
- **prmt** = uses promotion and/or whole-array operations/assignment
- **slice** = uses array slices
- **reduc** = uses reductions
- **scan** = uses scans

### Outline

- **Overview of the release structure**
- **Getting started with the hands-on session**
  - platform notes
  - getting started
  - then what?
- **Chapel environment settings**
Supported Platforms for hands-on session

- **Option 1**: Use the provided installation

- **Option 2**: Use your own machine
  - **Linux, Mac, UNIX users**: should have no problems
  - **Windows users**: have three options:
    - use Cygwin (UNIX emulation environment)
      - works fairly well in practice, particularly for experienced users
      - can be a bit sluggish, particularly on Vista
      - read README.cygwin on the web before getting started
    - ssh/telnet into a UNIX platform and work there
    - find someone to partner with

- **No computer?** find someone to partner with

Steps to getting started (from the README)

1. Make sure you’re in the `chapel/` directory

2. Build the compiler and runtime libraries using `gmake`
   - or `make` if your copy is GNU-make-compatible (as on Cygwin)

3. Set up your shell’s environment to use Chapel
   - if you use… then type…
     - `...csh, tcsh`  `source util/setchplenv.csh`
     - `...bash`  `source util/setchplenv.bash`
     - `...sh`  `. util/setchplenv.sh`
     - `...something else?`  Come talk to us

4. Compile an example program using:
   `chpl -o hello examples/hello.chpl`

5. Execute the resulting program:
   `./hello`
Then what?

- Whatever you want to do:
  - Look at, compile, execute our example programs
  - Explore the release -- see the bottom of the README for pointers
  - Try executing Chapel on multiple locales -- see README.multilocal
  - Try coding up an algorithm of interest to you
  - Work through some of the exercises we've prepared

- Please ask us questions if you have any difficulties
  - (or simply questions)

- Reminders:
  - break at 17:00
  - please fill out and return a survey form before you leave today

Outline

- Overview of the release structure
- Getting started with the hands-on session
- Chapel environment settings
  - main settings
  - cross-compilation settings
  - other settings
Main Chapel Environment Settings

**CHPL_HOME:** points to location of chapel/ directory
- **default:** none
- **typical values:** ~/chapel, /cygdrive/c/chapel, or any path

**CHPL_HOST_PLATFORM:** architecture on which compiler is built, run
- **default:** a best guess is made using `uname -a`
- **typical values:** cygwin, darwin, linux, linux64, sunos, xt-cle

**PATH:** the Chapel compiler’s path should be added to yours
- **default:** none
- **typical value:** $CHPL_HOME/bin/$CHPL_HOST_PLATFORM

**MANPATH:** Chapel’s man page path should be added to yours
- **default:** none
- **typical value:** $CHPL_HOME/man

(See $CHPL_HOME/doc/README.chplenv for more detail)

Cross-Compilation Environment Variables

**CHPL_TARGET_PLATFORM:** architecture for which Chapel is compiled
- **default:** $CHPL_HOST_PLATFORM
- **typical values:** mta, x1, x2, xmt, xt-cle

**CHPL_HOST_COMPILER:** compiler to use for the host platform

**CHPL_TARGET_COMPILER:** compiler to use for the target platform
- **default:** a best guess is made using the corresponding PLATFORM variable
- **typical values:** gnu, intel, pathscale, pgi, cray-hta, cray-vec, cray-xt-gnu, cray-xt-pathscale, cray-xt-pgi, ibm

**CHPL_MAKE:** the GNU-compatible make utility to use for the target
- **default:** a best guess is made using the PLATFORM variables
- **typical values:** gmake, make

(See $CHPL_HOME/doc/README.chplenv for more detail)
Other Environment Variables

**CHPL_THREADS**: threading layer to use for the generated code
- **default**: a best guess is made using $CHPL_TARGET_PLATFORM
- **typical values**: none, pthreads, mta

**CHPL_COMM**: communication layer to use for the generated code
- **default**: none
- **typical values**: none, gasnet, armci

**CHPL_***: most compiler options can be set using an environment variable
- see `chpl --help-env` and `--help` for details

*(See $CHPL_HOME/doc/README.chplenv for more detail)*

To Download:

http://chapel.cs.washington.edu

Example starting points (and sample solutions):

http://chapel.cs.washington.edu/PRACE/exercises

Emacs and vim modes:

http://chapel.cs.washington.edu/publicRelease/editors.html
Questions?