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
Introduction to Chapel: the Cascade High-Productivity Language

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
Chapel 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

```
|                   Data Parallel Features |
|-------------------|----------------------------------------|
| test              | frall | poste | slice | reduc | scan |
| hello             |       |       |       |       |      |
| hello-module      |       |       |       |       |      |
| bee               | x     |       |       |       |      |
| blockIdx         |       | x     |       |       |      |
| fileIO            |       |       |       |       |      |
| genericStack      |       |       |       |       |      |
| jacobi            |       | x     |       |       |      |
| linkedList        |       |       |       |       |      |
| smoke             | x     |       |       |       |      |
| quicksort         |       | x     |       |       |      |
| reductions        |       |       | x     |       |      |
| slices            | x     | x     |       |       |      |
| space             | x     |       |       |       |      |
| tree              |       | x     |       |       |      |
| fcg/cfg           |       | x     |       |       |      |
| fcg/fft           |       | x     |       |       |      |

Key:
- frall = uses forall loops
- poste = uses promotion and/or whole-array operators/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

- **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
    - get a copy of README.cygwin from us before getting started
    - we can help you install Cygwin if you’re not familiar with it
  - ssh/telnet into a UNIX platform and work there
  - find someone to buddy up with

- **No computer?** find someone to buddy up 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...
     - `csh`, `tcsh`
     - `bash`
     - `sh`
     - ...something else? Come talk to us
     - then type...

     ```
     ...csh, tcsh
     source util/setchplenv.csh
     ...
     ...bash
     source util/setchplenv.bash
     ...
     ...sh
     . util/setchplenv.sh
     ...
     ```

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

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

- Whatever you like:
  - Look at, compile, execute other example programs
  - Explore the release -- see the bottom of the README for pointers
  - 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 3pm
  - we'll reconvene at 4:30pm for a final Q&A and to get your feedback

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

**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-cntl

**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-xt-gnu, cray-xt-pathscale, cray-xt-pgi

**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_TARGETPLATFORM
- **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)*

Questions?