

Chapel: Hands-On Session

Chapel Team



SC08: Tutorial S07 – 11/16/08



Outline

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

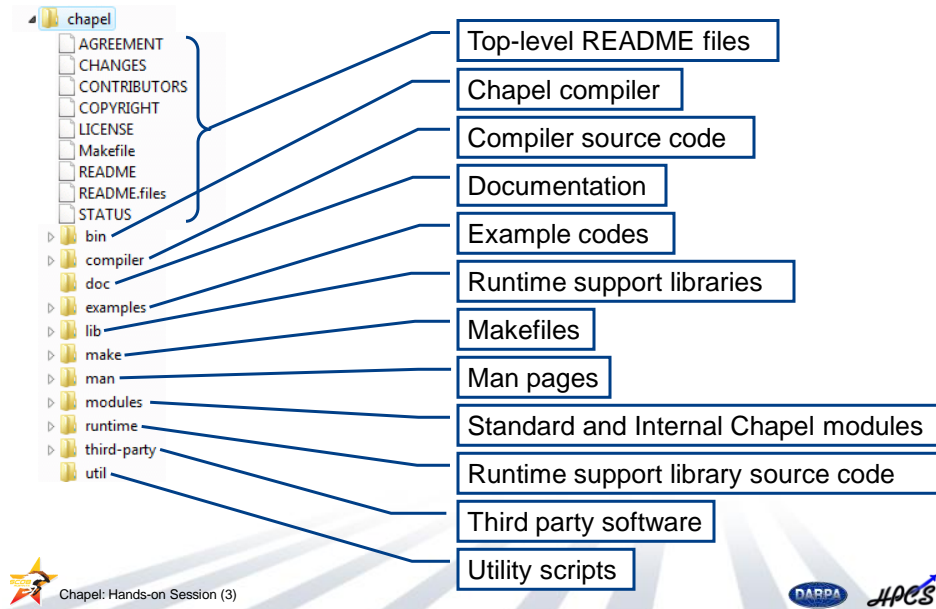


Chapel: Hands-on Session (2)





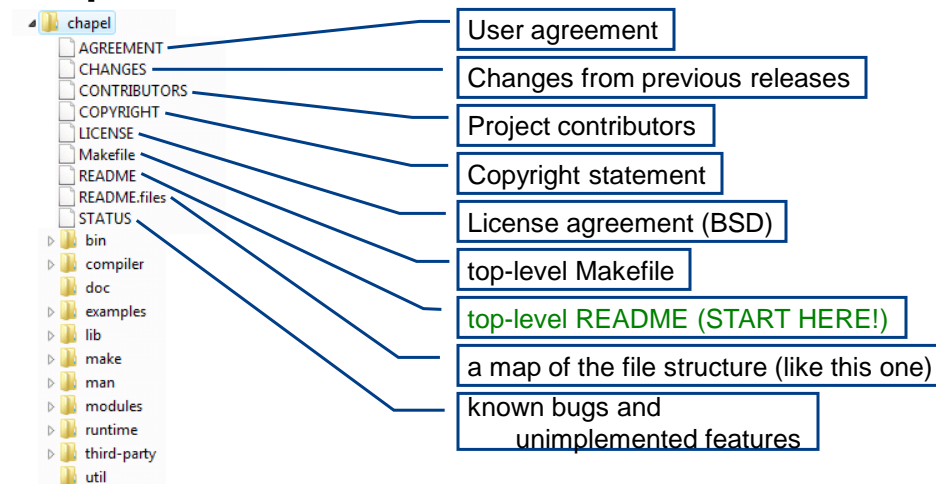
Release Directory Structure



Chapel: Hands-on Session (3)



Top-Level Documentation



Chapel: Hands-on Session (4)





Man Page

- chapel
 - AGREEMENT
 - CHANGES
 - CONTRIBUTORS
 - COPYRIGHT
 - LICENSE
 - Makefile
 - README
 - README.files
 - STATUS
 - bin
 - compiler
 - doc
 - examples
 - lib
 - make
 - man
 - modules
 - runtime
 - third-party
 - util

man page for the Chapel compiler

```

chapel(1)                  chapel(1)

NAME
  chapel - Cray Inc. compiler for the Chapel parallel language

SYNOPSIS
  chapel [-w options] [-m options] [-c flags]
         [-print options] [-print:options]
         [flags...] [source-files...]

DESCRIPTION
  chapel converts one or more Chapel source files into an executable. It
  does this by compiling Chapel to ISO C code and then invoking the
  platform's C compiler to create the executable. However, most users
  will not need to be aware of the use of C as an intermediate format
  during compilation.

  The current implementation of chapel supports the creation of multiple
  threads on a single locale and uses POSIX threads (pthreads) to imple-
  ment its parallelism. Future versions of the compiler will support
  distributed memory architectures.

  This version of chapel focuses primarily on demonstrating Chapel lan-
  guage features. In order to support experimentation with the language
  and generate user feedback, the performance and memory utilization of
  the generated code is known to be suboptimal and should not be consid-
  ered representative of Chapel's potential. Future versions of the
  compiler will continue to result in improved performance.

OPTIONS
  Compiler Information Options
  -h, --help
    Print a brief list of the command line options, listing the
    arguments that they expect, and a summary of their purpose.
  --copyright
    Print the compiler's copyright information.
  --license
    Print the compiler's license information.
  --version
    Print the version number of the compiler.
  Compilation Inace Options
  --print-commands
    Prints the system commands that the compiler executes in order
    to execute the program compilation.
  --print-passes
    Prints the compiler passes during compilation and the amount of
    wall clock time required for the pass.
  Code Size Options
  --count-tokens
    Prints out the number of static lexical tokens in the user's
    Chapel code.
  --print-code-size
    Prints out the size of the user's code in great detail: For
    each code file, first the code is echoed back to the screen,
    and code lines (line with the number of lexical tokens in com-
    cents, or 'C' if the line only contains comments, or 'M' if the
    line is blank), then the total number of tokens for the file is
    printed. Then the number of lines is displayed, broken down
    into code lines, comment-only lines, and blank lines. Then the
  
```



Chapel: Hands-on Session (5)



Doc Directory: Main files

- chapel
 - doc
 - chapelLanguageSpec.pdf
 - hpccOverview.pdf
 - hpccTutorial.pdf
 - quickReference.pdf
 - README
 - README.bugs
 - README.building
 - README.chplenv
 - README.compiling
 - README.cygwin
 - README.executing
 - README.extern
 - README.format
 - README.multilocale
 - README.prereqs
 - README.threads
 - README.xt-cnl

- 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



Chapel: Hands-on Session (6)





Doc Directory: Other files

chapel

- doc
 - chapelLanguageSpec.pdf → map of this directory (like these slides)
 - hpccOverview.pdf → how to report bugs
 - hpccTutorial.pdf → notes for Cygwin users
 - quickReference.pdf → technical note on stopgap measure for calling external C routines
 - README
 - README.bugs
 - README.building
 - README.chplenv
 - README.compiling
 - README.cygwin
 - README.executing
 - README.extern → technical note on stopgap measure for formatting string conversions
 - README.format
 - README.multilocale
 - README.prereqs → prerequisites for using Chapel
 - README.threads
 - README.xt-cn1 → technical notes for Cray XT users



Chapel: Hands-on Session (7)



Examples Directory: Notable files

chapel

- examples
 - hello.chpl → example Chapel source code
 - hello.good → expected output for test system
 - Makefile → Makefile to build all examples
 - README
 - README.features → directory structure and overview of example codes
 - README.testing → feature grid mapping examples to language features
 - start_test → information about the Chapel testing system

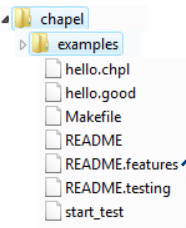


Chapel: Hands-on Session (8)





Examples Directory: Feature Grid



	Data Parallel Features							
test	frall	prmtc	slice	reind	reduc	scan	xxxxx	xxxxx
hello								
hello-module								
beer	x							
blockUP	x		x	*	x			
fileIO					x			
genericStack								
iterators								
jacobi	*		*		*			
linkedList								
norm	x	x						
prodCons								
quicksort			x					
reductions		*	*		x			
slices	*	*	x	x				
sparse	x				x			
tree								
hpcc/stream	x	*			x			
hpcc/ra	x				x			
hpcc/fft	x	x	x		x			

key:
frall = uses forall loops
prmtc = uses promotion and/or whole-array operators/assignment
slice = uses array slices
reind = uses array reindexing/array views
reduc = uses reductions
scan = uses scans



Chapel: Hands-on Session (9)



Outline

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



Chapel: Hands-on Session (10)



CRAY

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



Chapel: Hands-on Session (11)

DARPA

HPCS

CRAY

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...

<code>...csh, tcsh</code>	<code>source util/setchplenv.csh</code>
<code>...bash</code>	<code>source util/setchplenv.bash</code>
<code>...sh</code>	<code>. util/setchplenv.sh</code>
 - ...something else? Come talk to us
4. Compile an example program using:


```
chpl -o hello examples/hello.chpl
```
5. Execute the resulting program:

```
./hello
```



Chapel: Hands-on Session (12)

DARPA

HPCS

CRAY

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



Chapel: Hands-on Session (13)

DARPA

HPCS

CRAY

Outline

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



Chapel: Hands-on Session (14)

DARPA

HPCS



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)



Chapel: Hands-on Session (15)



Cross-Compilation Environment Variables

CHPL_TARGET_PLATFORM: architecture for which Chapel is compiled

- **default:** `$CHPL_HOST_PLATFORM`
- **typical values:** mta, x1, x2, xmt, xt-cnl

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-mta, cray-vec, cray-xt-gnu, cray-xt-pathscales, 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)



Chapel: Hands-on Session (16)





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)



Chapel: Hands-on Session (17)



Questions?



THE SUPERCOMPUTER COMPANY