Other Miscellaneous and Notable Changes

Chapel Team, Cray Inc.

Chapel version 1.12

October 1st, 2015

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Outline

Packaging Improvements

Packaging Improvements

- **Rewrote 'make check' in bash**
  - removes reliance on start_test/tcsh/Python which hurt portability
  - in hands-on sessions, 'make check' failed, though compiler worked
- **Made 'printchplenv' indicate set vs. inferred values**
  - '*' now indicates a value set by environment variable:

```
CHPL_HOST_PLATFORM: darwin *
CHPL_HOST_COMPILER: gnu *
CHPL_TARGET_PLATFORM: darwin
CHPL_TARGET_COMPILER: gnu
CHPL_TARGET_ARCH: none *
CHPL_LOCALE_MODEL: flat
CHPL_COMM: none
…
```

Other Compiler Improvements

Other Compiler Improvements

- **Improved message for internal error messages**
  - more “it’s us, not you” in tone
  - includes best stab at source code location causing problem
  - points to web documentation for filing bugs
- **Made --fast no longer imply --no-ieee-float**
- **Added --ieee-float support for ‘clang’ and ‘intel’**
- **Made --ccflags arguments stack**
- **When using LLVM back-end…**

  …enabled optimizations and streamlined code

  …added support for --print-emitted-code-size

Example Code Changes

Changes to examples/ programs

- **Added learnChapelInYMinutes.chpl to examples/primers/**
  - local copy of: http://learnxinyminutes.com/docs/chapel/
  - contributed by Ian Bertolacci (Colorado State University)
- **Created a new examples/patterns directory**
  - goal: create a cache of “How would I write X in Chapel?” patterns
  - Only one program here so far… ☹
    - recordio.chpl: How to read file of records with tab-separated fields
- **Updated nbody shootout program to use ‘ref’ variables**
- **Removed ‘local’ block from Stream EP and related cleanup**
- **Replaced ‘format()’ calls with ‘writef()’ in SSCA#2**
- **Improved numerical tolerance in fileIO and FFTW primers**
- **Removed ‘param’ from LULESH loops**

Bug Fixes: Scope Resolution

Scope Resolution: Incorrect Method Shadowing

- **Shadowing bug: Method hid outer vars and functions**
  - Desirable when in method on same type
  - Wrong when in function, or method on different type!
- **Why did this happen?**
  - Method stored in symbol table for scope by base name only
    - i.e. someRec.foo stored as “foo” not “someRec.foo”
    - Helpful for inheritance, use in other methods
  - But didn’t check if in method on same type!
    - That check happens in function resolution

```
module Mod {
  proc someRec.foo {…}
}

var foo: int;

proc bar(arg) {
  use Mod;

  return arg * foo;
}
```

FIXED

Scope Resolution: Module Use Shadowing

- **Another shadowing issue:**
  - Consider the following code:

```
proc bar(foo) {
  use Mod;
  return callon(foo);
}
```

IMPROVED

  - User expects *foo* refers to argument foo
  - But if *Mod* also defines a foo, that symbol is more in scope
    - This is potentially confusing
    - And likely not what the user intended

- **Solution: Warn user when this happens**

… so they can rename the argument
… or limit the symbols they use (once **except** keyword available)

Scope Resolution: Single-namespace Issues

- **Chapel is a single namespace language**
  - Except when it unintentionally isn't …

```
module foo {
   …
}

proc foo (…) {
   …
}
```

This compiled successfully

And so did this

```
var foo: [1..10] real;

proc foo (i) {
   …
}
```

Scope Resolution: Single-namespace Issues

- **Chapel is a single namespace language**
  - Except when it unintentionally isn't …

```
module foo {
   …
}

proc foo (…) {
   …
}
```

Now both complain about naming conflicts

```
var foo: [1..10] real;

proc foo (i) {
   …
}
```

FIXED

Bug Fixes: Standalone Parallel Iterators

Standalone Parallel Iterators

Background: `forall` loops over a single array should use its **standalone** parallel iterator

- it did not when the loop referenced an outer variable, e.g.:

```
var outer = 5;
forall a in A do
  a = outer;
```

```
for followThis in A.these(leader) do
  for a in A.these(followThis,
                   follower) do
    a = outer;
```

This Effort: Fixed that bug

```
var outer = 5;
forall a in A do
  a = outer;
```

```
for a in A.these(standalone) do
  a = outer;
```

Impact: Improved generated code

- smaller size
- potentially faster execution

Bug Fixes: Outer Variable Capture

Outer Variable Capture: Background

- **Given**
  - a **`cobegin`** or **`coforall`** statement
  - an outer variable with **`in`**-like intent

```
var outer = 5;
proc update() { outer = 6; }
coforall i in 1..2 do
  if i==1 then update();          //  in task 1
  else         writeln(outer);    //  in task 2
```

*`outer` is implicitly passed into task functions by default intent, which is **`const in`** for integers*

could observe 5 or 6 depending on timing of tasks

- **observed value of outer variable could vary**
  - in presence of concurrent updates
  - task 2 could capture `outer` before or after task 1 updated it

Outer Variable Capture: This Effort

- **Capture outer variable right before statement**
  - guarantees consistent value in all tasks – the desired semantics
  - for **`in`**-like intents only

```
var outer = 5;
proc update() { outer = 6; }
coforall i in 1..2 do
  if i==1 then update();          // in task 1
  else         writeln(outer);    // in task 2
```

guaranteed to observe 5
i.e. its value at start of **`coforall`**

Outer Variable Capture: Status and Next Steps

Impact:

- ensures correct semantics
- prevents hard-to-find data races
  - note: semantics allows races for **records** that are passed by default intent

Status:

- implemented for task-parallel constructs

Next Steps:

- extend to `forall` loops
- optimize away unnecessary copies

Bug Fixes: Reduce Intents Over Arrays, Domains

Reduce Intents: Background–Semantics

a variable passed into a `forall` loop with a reduce intent will aggregate values from individual loop iterations

`x` is passed into the loop by reduce intent, will aggregate using +

inside the loop, `x` is implicitly a task-private shadow *variable*

```
var x: int;
forall i in myIterator() with (+ reduce x) {
  x += i;
}
writeln(x);
```

after the loop, `x` contains the aggregated result

Reduce Intents: Background–Implementation 1

user `forall` loop

```
var x: int;
forall i in myIterator() with (+ reduce x) {
  x += i;
}
writeln(x); // prints sum of values yielded by myIterator()
```

implementation

```
var x: int;
const xOp = new SumReduceScanOp();
for zip(i, ref xShadow) in myIterator(xOp, standalone) {
  xShadow += i;
}
xOuter = xOp.generate()
writeln(x);
```

alias for a shadow variable created by compiler-modified `myIterator()` – see next ...

Reduce Intents: Background–Implementation 2

user parallel iterator

```
iter myIterator(param tag) where tag == standalone {
  coforall ... {
    yield expr;
} }
```

create a task-private shadow variable…

... for use in loop body

implementation

```
iter myIterator(xOp, param tag) where tag == standalone {
  coforall ... {
    const currOp = xOp.clone();
    var xShadow = currOp.identity;
    yield (expr, ref xShadow);
    currOp.accumulate(xShadow);
    xOp.combine(currOp);
    delete currOp;
} }
```

accumulate value of `xShadow` *at end of task*

Reduce Intents: Background–Missed Cases 1

- **We implemented an important case first**
  - parallel iterator has `yield`(s) within task-parallel constructs
    - `begin`, `cobegin`, `coforall`

- **Needed to implement other cases**
  - seen in iterators invoked by `forall` over a domain or array
    a. domain iterator: a `yield` outside any parallel construct
    b. array iterator: a `yield` in `for` loop over another parallel iterator

Reduce Intents: Background–Missed Cases 2

these cases were not handled before

a. **`yield`** outside any parallel construct – e.g. in domain iterator

```
iter _domain.these(param tag) where tag == standalone {
  if numChunks <= 1 {   ... yield expr1; ... }
  else  task-parallel case, handled already
}
```

was not handled

b. **`yield`** in **`for`** loop over other parallel iterator – e.g. in array iterator

```
iter _array.these(param tag) where tag == standalone {
  for i in dom.these(tag) do
    yield dsiAccess(i);
}
```

was not handled

another parallel iterator

Reduce Intents: This Effort – Handle Case A

yield *outside any parallel construct*

source code: domain parallel iterator

```
iter _domain.these(param tag) where tag == standalone {
  if numChunks <= 1 {  ... yield expr1; ... }
  else task-parallel case
}
```

shadow variable for non-parallel yields

implementation

```
iter _domain.these(xOp, param tag) where tag == standalone {
  var xShadow = xOp.identity;
  if numChunks <= 1 {  ... yield (expr, ref xShadow); ... }
  else task-parallel case, handled as before
  xOp.accumulate(xShadow);
}
```

Reduce Intents: This Effort – Handle Case B

yield *in* **for** *loop over another parallel iterator*

source code: array parallel iterator

```
iter _array.these(param tag) where tag == standalone {
  for i in dom.these(tag) do
    yield dsiAccess(i);
}
```

propagate shadow variable from the other iterator

implementation

```
iter _array.these(xOp, param tag) where tag == standalone {
  for zip(i, ref xShadow) in dom.these(xOp, tag) do
    yield (dsiAccess(i), ref xShadow);
}
```

- also: modify a copy of `dom.these()` *as if* for a **`forall`** loop

Reduce Intents: Status and Next Steps

Impact:

- can use `reduce` intents with `forall` loops over domains and arrays

Status:

- reduce intents with forall loops over arrays/domains working
- reduce intents with forall loops over ranges not working with 1.12
  - has since been fixed on master, though

Next Steps:

- re-implement reductions using `forall` loops and `reduce` intents
- tune performance
- design and implement partial reductions

Other Notable Bug Fixes

Other Notable Bug Fixes

- Overloads of '|' no longer break internal modules
- Extern variables of type c_ptr(c_int) now work better
- Classes can now call parent class' destructor
- Fixed large array copies where size > max(int(32))
- Function calls of the form <type>(<args>) no longer error
- Non-blocking 'on's no longer counted as local tasks
- Fixed compiler exception when dividing by param 0
- FileSystem is*() routines handle invalid paths/links better
- Made chpldoc better handle directory creation failures
- Added an error message for too-long compiler flags

More Notable Bug Fixes

- Closed leaks for heap-allocated cobegin/coforall vars
- Improved support for malloc/free in extern blocks
- Fixed occasional ‘text file busy’ error when making ‘chpl’
- Stopped permitting overloading via argument intents
- Fixed an occasional segfault when zippering glob()
- Fixed source locations passed to string routines
- Fixed source locations for cobegin statements
- Improved inlined iterators for generic array fields
- Improved passing c_strings to extern C functions

I/O Bug Fixes

- I/O on integers works with '%{##.##}'-style formats now
- when skipping whitespace, illegal characters handled
- made readf() calls halt on mismatches when no error arg
- trailing whitespace is now consumed less aggressively
- fixed EOF bugs in Reader/Writer types
- channel.read(<style>) no longer ignores style argument

Error Message Improvements

Error Message Improvements

- Improved error messages for runaway comments
- Improved source locations for ‘noinit’ warnings

Runtime Changes

Runtime Changes

- Moved polling thread to last CPU to avoid contention
- Added support for out-of-segments puts/gets
- Changed I/O to allocate buffers from Chapel heap

Third-Party Changes

Third-Party Changes

- Added ‘fltk’ to third-party directories for use by ‘chplvis’
- Enabled use of GMP with the LLVM back-end
- Made LLVM build in non-debug mode by default
- Improved cross-compilation of third-party on ‘cray-x*’
- Switched to storing RE2 in an unbundled form
- Fixed a valgrind issue in RE2

Platform-Oriented Improvements

Platform-specific Changes

- Made ‘cray-x*’ systems default to ‘qthreads’ over ‘muxed’
- ‘muxed’ now supports guard pages for non-hugepages
- Added support for ‘clang-included’ with GASNet on Crays
- Removed support for ‘cray-prgenv-pgi’ from Cray module
- Stopped throwing –hipa2 by default for ‘cray-prgenv-cray’

Portability Fixes/Platform-Specific Bugfixes

- Fixed [_BSD|_SVID]_SOURCE deprecation issues
- Improved building of SysCTypes.chpl for Fedora 22
- Fixed a pair of stack-related bugs in ‘muxed’ tasking
- Removed symmetric address assumptions in error code
- Fixed a number of I/O issues on Cygwin
- Fixed tcmalloc for clang 3.6 when used from C++
- Fixed I/O for 32-bit Ubuntu 14.04
- Added support for building GASNet segment fast on OS X
- Fixed hwloc’s Cairo detection for certain OS X cases
- Eliminated Xcode-specific warnings

Launcher Changes

Launcher Changes

- Improved ‘slurm’s handling of non-zero exit codes
- Changed how ‘amudprun’ deals with quoted arguments

Test System Improvements

Correctness Test System Improvements

- Rewrote key scripts in Python (from ‘csh’)
- Improved mechanism for suppressing expected failures
- Added ability to write ‘chpldoc’ and ‘chpl-ipe’ tests
- Made parallel testing print estimated end time
- Added support for multi-option COMPOPTS files
- Made improvements to C code testing feature
- Extended timeout mechanism to work for Cygwin
- Added recognition of certain launcher failures

Performance Testing/Graphing Improvements

- Added a ‘screenshot’ capability for performance graphs
- Added annotations to ‘cray-xc’ performance graphs
- Improved resilience to missing annotations file
- Retired the code for the old gnuplot-based graphs

Legal Disclaimer

http://chapel.cray.com chapel_info@cray.com http://sourceforge.net/projects/chapel/