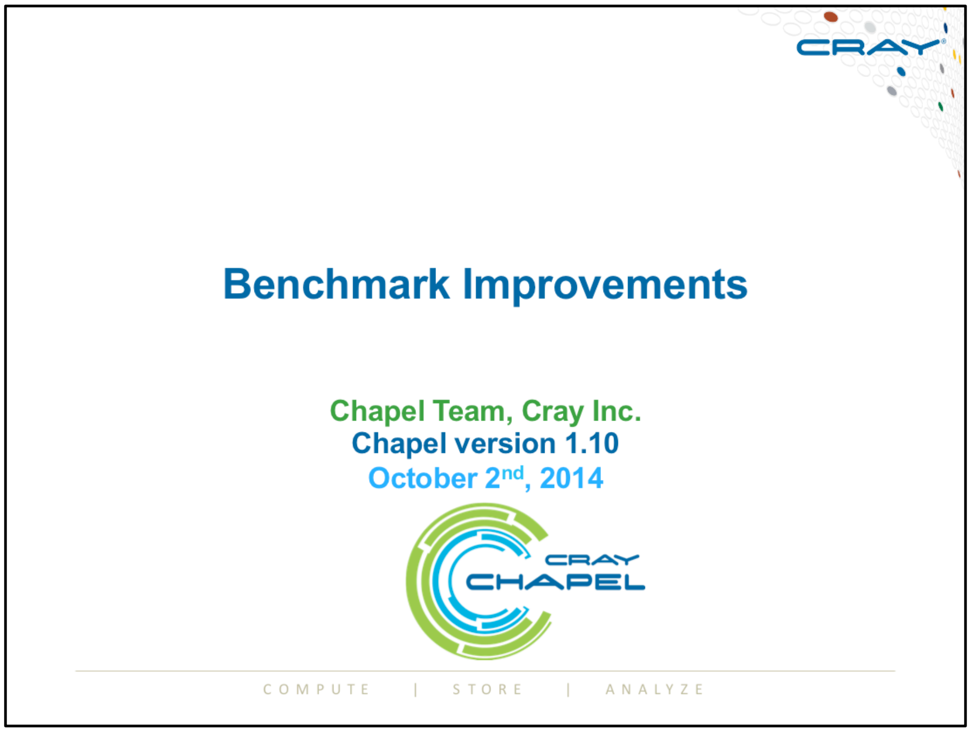
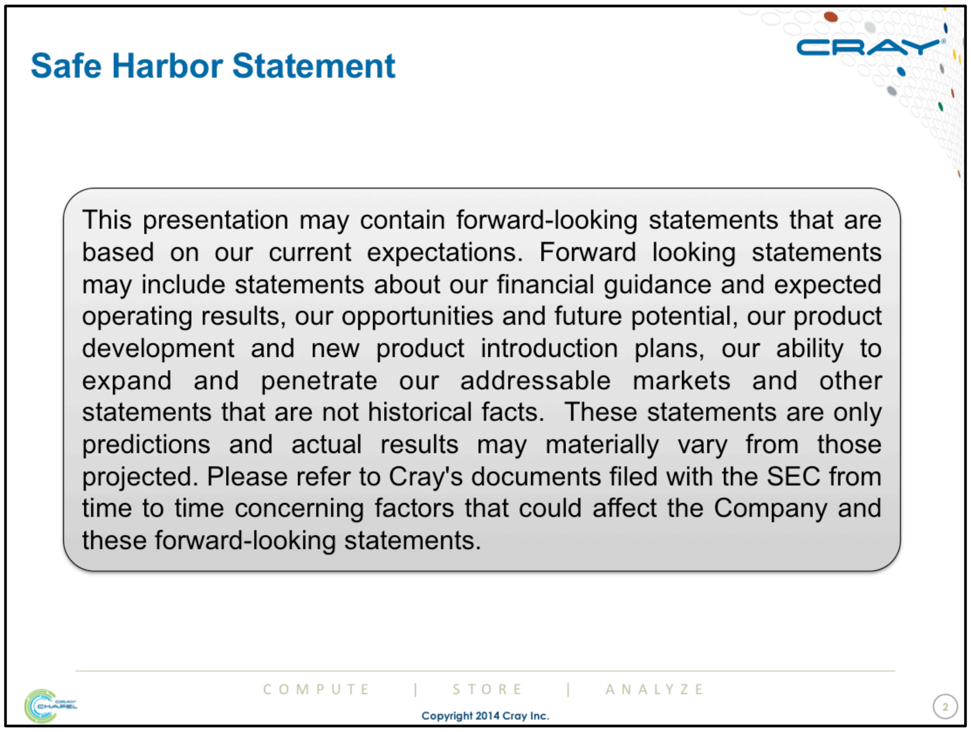


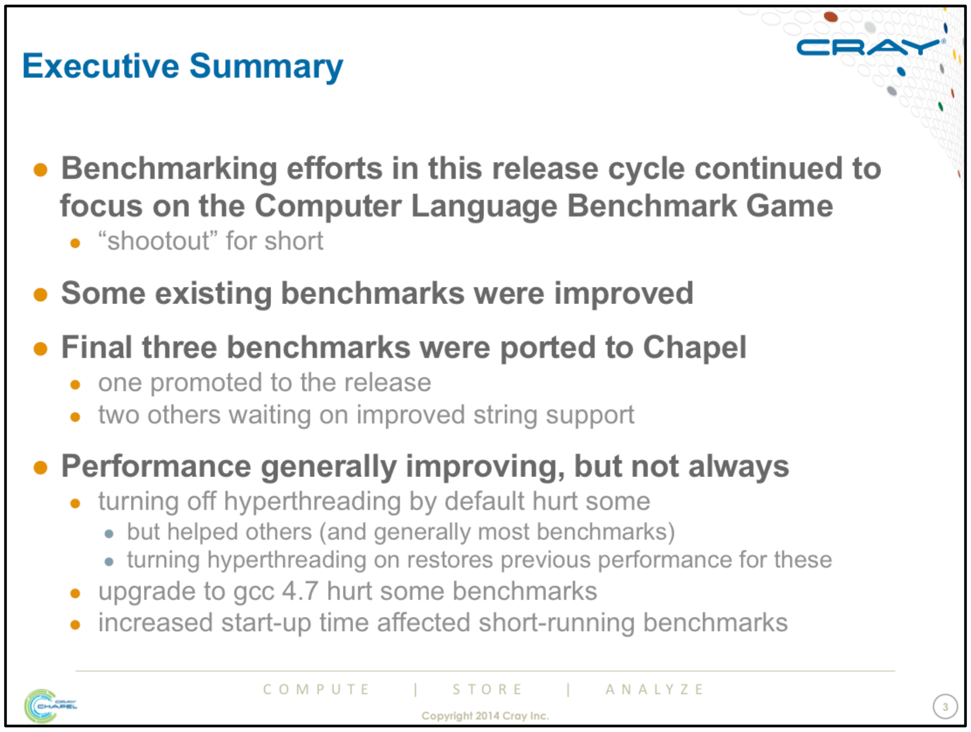
Benchmark Improvements

Chapel Team, Cray Inc.
Chapel version 1.10
October 2nd, 2014



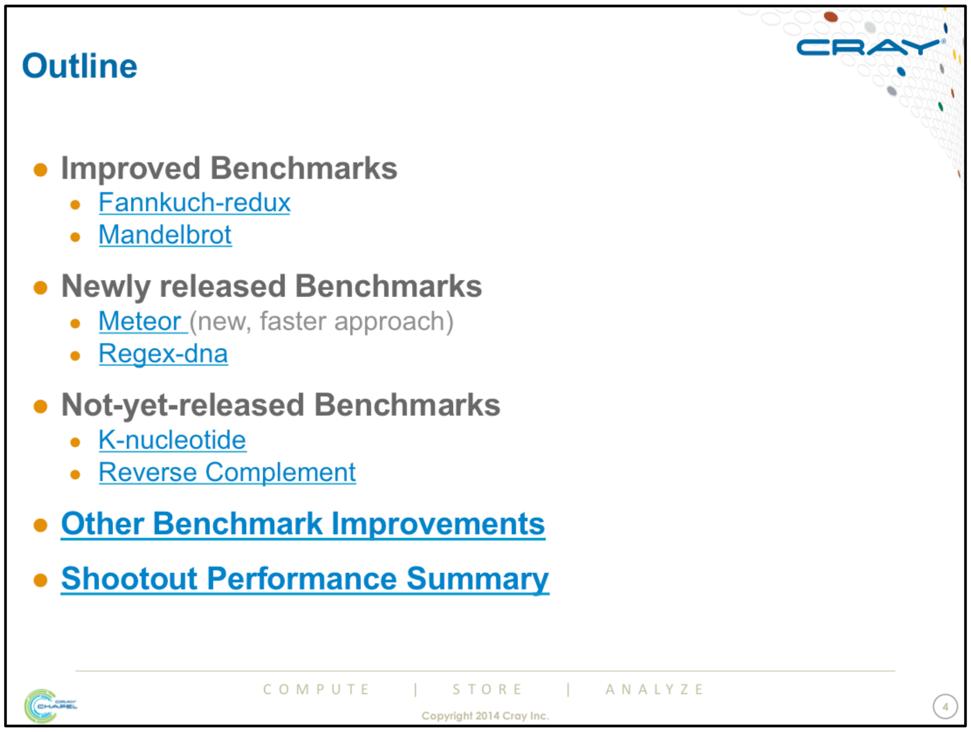
Safe Harbor Statement

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.



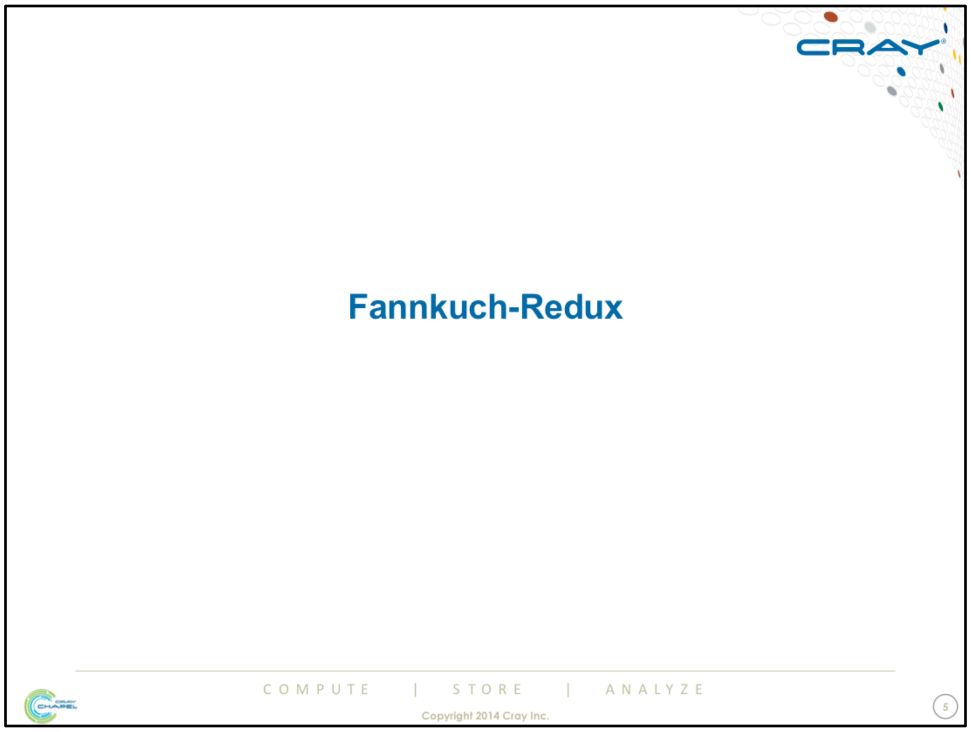
Executive Summary

- **Benchmarking efforts in this release cycle continued to focus on the Computer Language Benchmark Game**
  - "shootout" for short
- **Some existing benchmarks were improved**
- **Final three benchmarks were ported to Chapel**
  - one promoted to the release
  - two others waiting on improved string support
- **Performance generally improving, but not always**
  - turning off hyperthreading by default hurt some
    - but helped others (and generally most benchmarks)
    - turning hyperthreading on restores previous performance for these
  - upgrade to gcc 4.7 hurt some benchmarks
  - increased start-up time affected short-running benchmarks

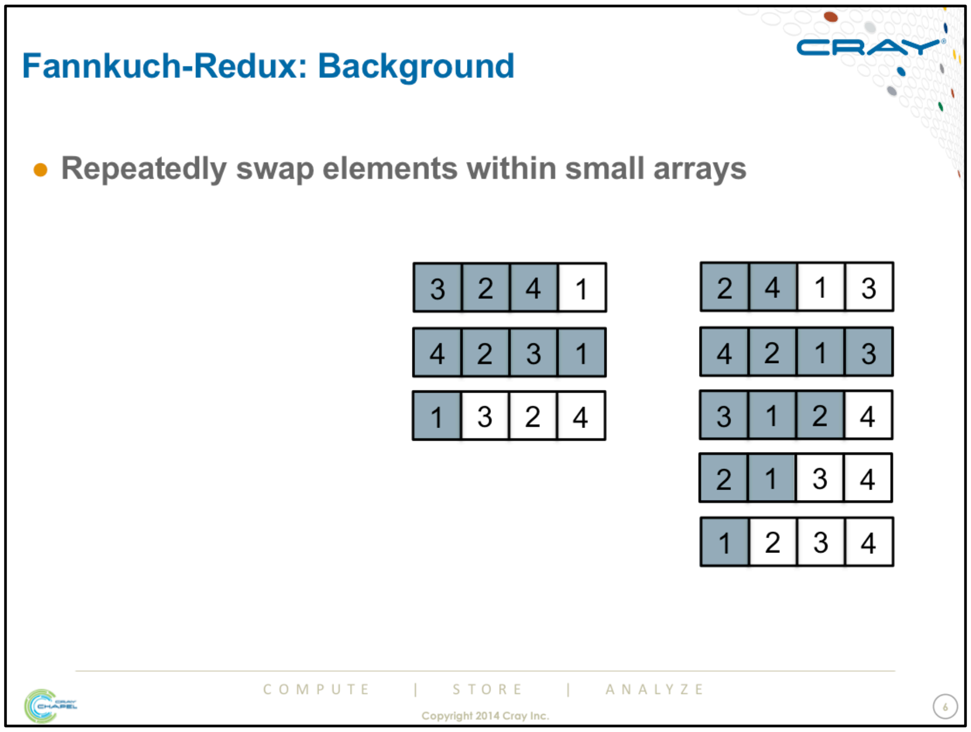


Outline

- Improved Benchmarks
  - Fannkuch-redux
  - Mandelbrot
- Newly released Benchmarks
  - Meteor (new, faster approach)
  - Regex-dna
- Not-yet-released Benchmarks
  - K-nucleotide
  - Reverse Complement
- Other Benchmark Improvements
- Shootout Performance Summary

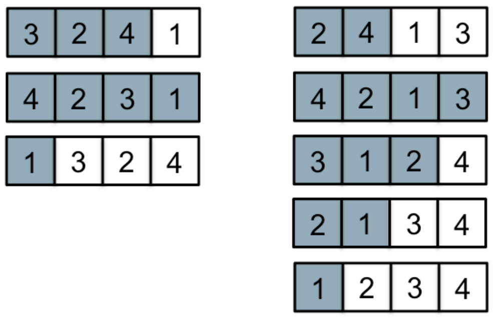


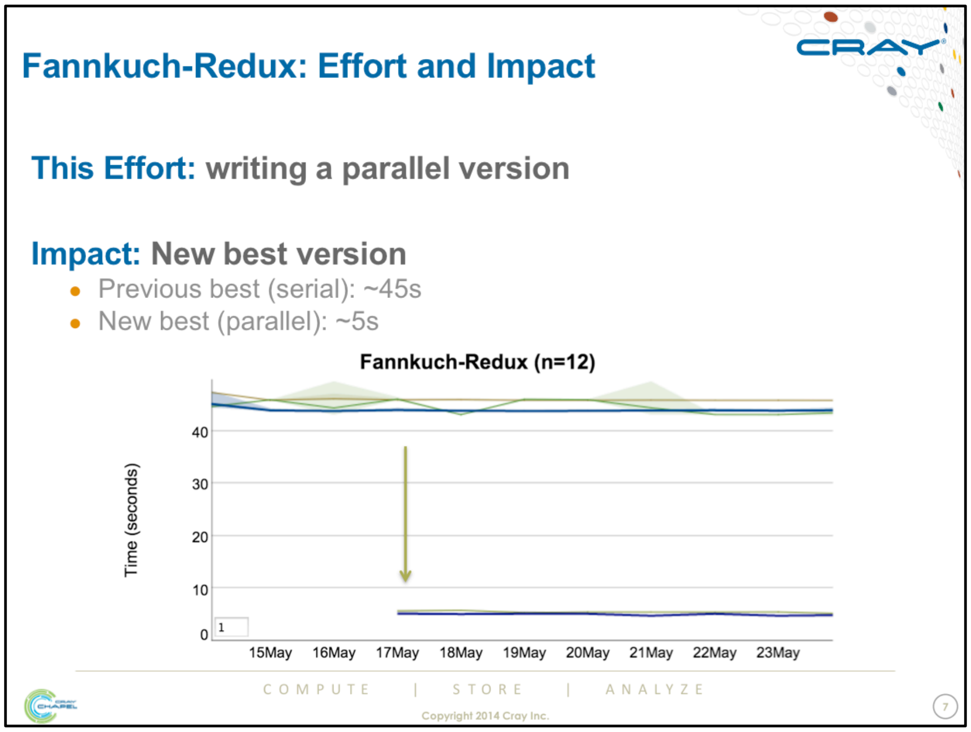
Fannkuch-Redux



Fannkuch-Redux: Background

- Repeatedly swap elements within small arrays



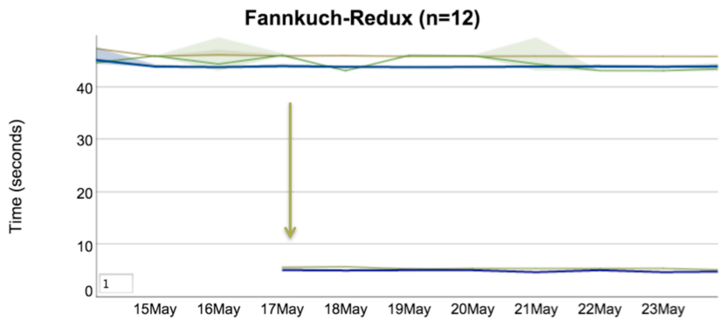


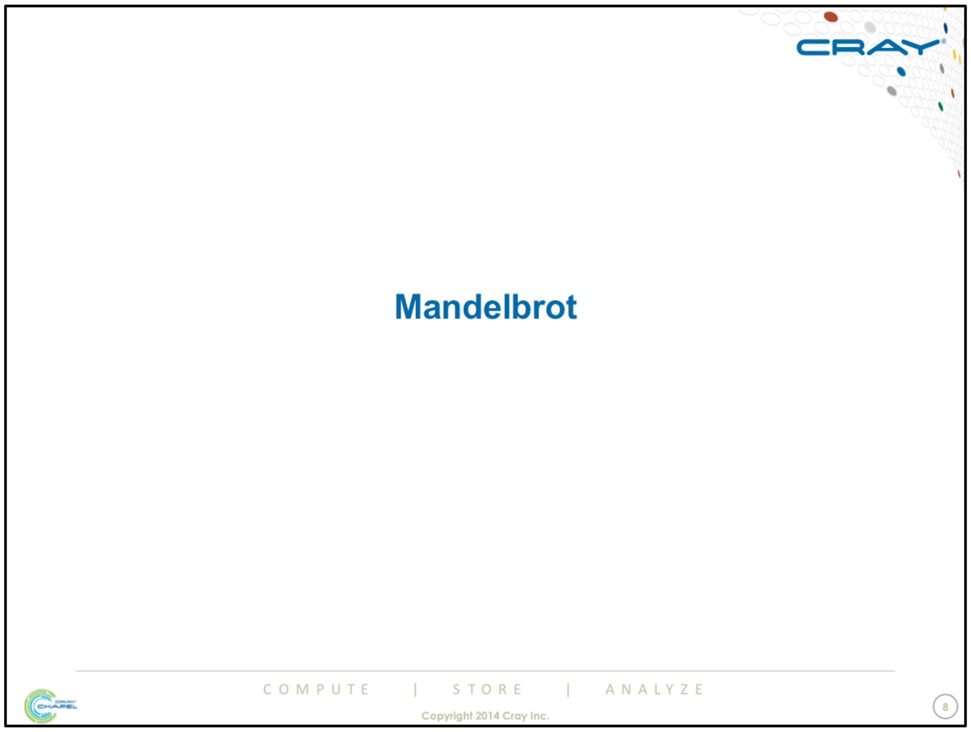
Fannkuch-Redux: Effort and Impact

This Effort: writing a parallel version

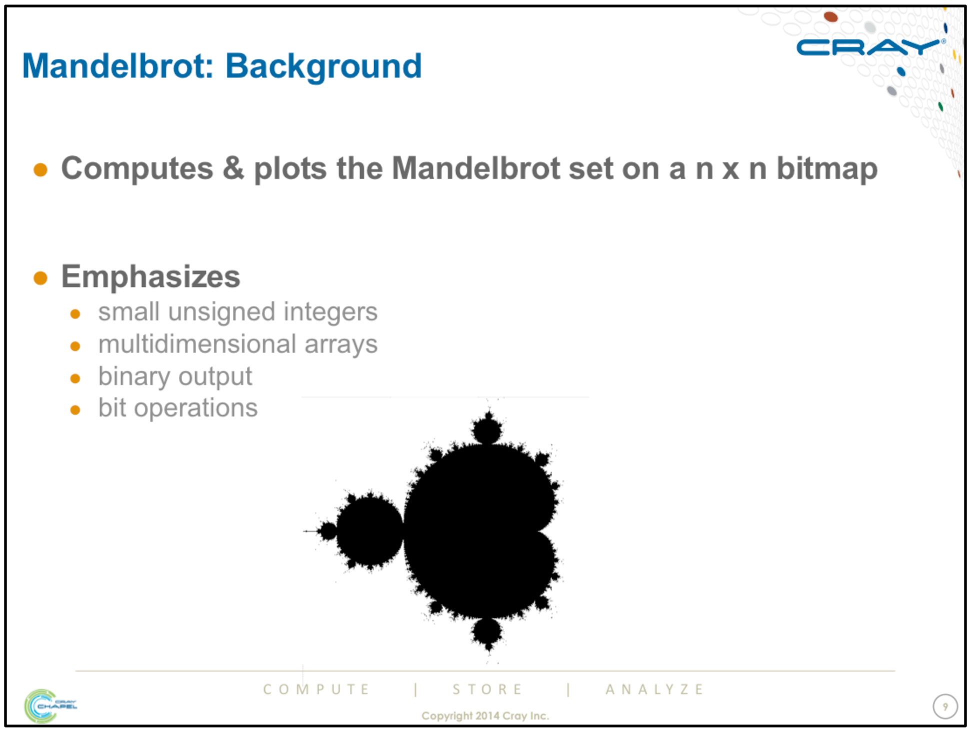
Impact: New best version

- Previous best (serial): ~45s
- New best (parallel): ~5s





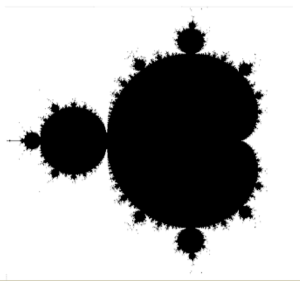
Mandelbrot

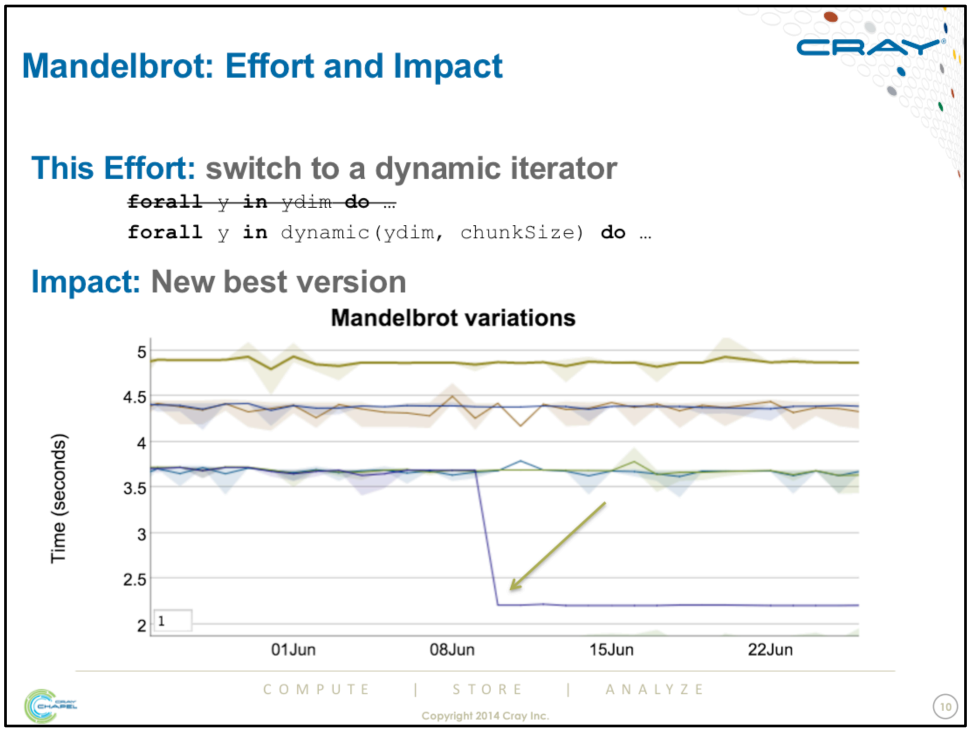


Mandelbrot: Background

- Computes & plots the Mandelbrot set on a n x n bitmap

- Emphasizes
  - small unsigned integers
  - multidimensional arrays
  - binary output
  - bit operations



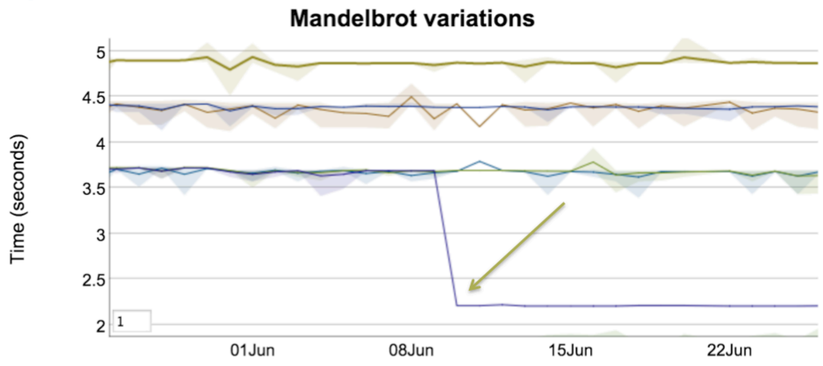


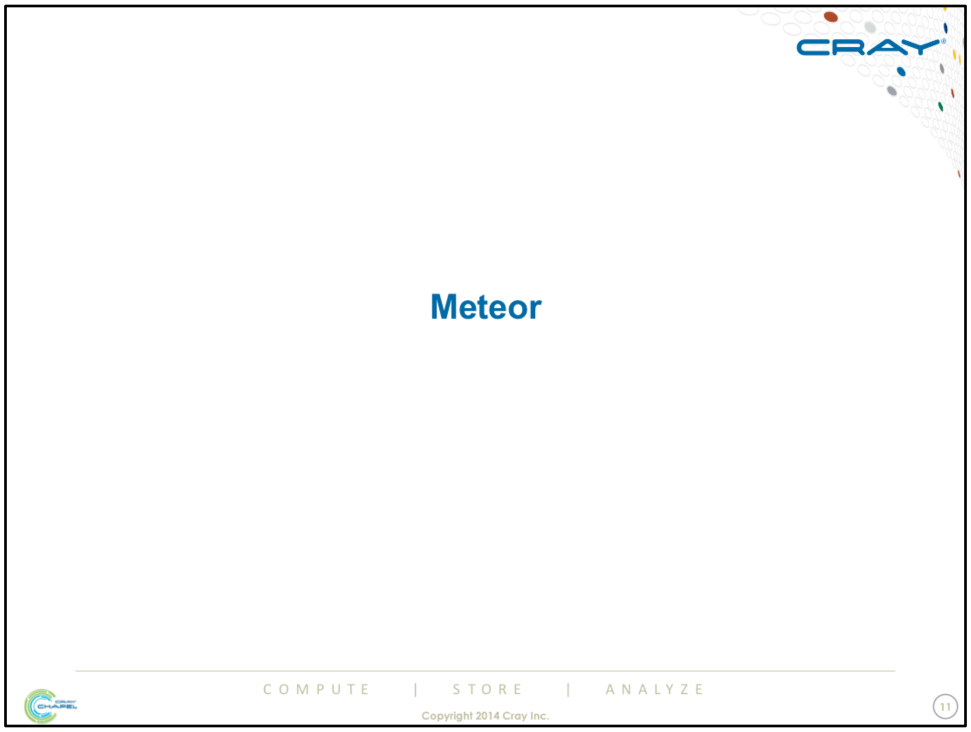
Mandelbrot: Effort and Impact

This Effort: switch to a dynamic iterator

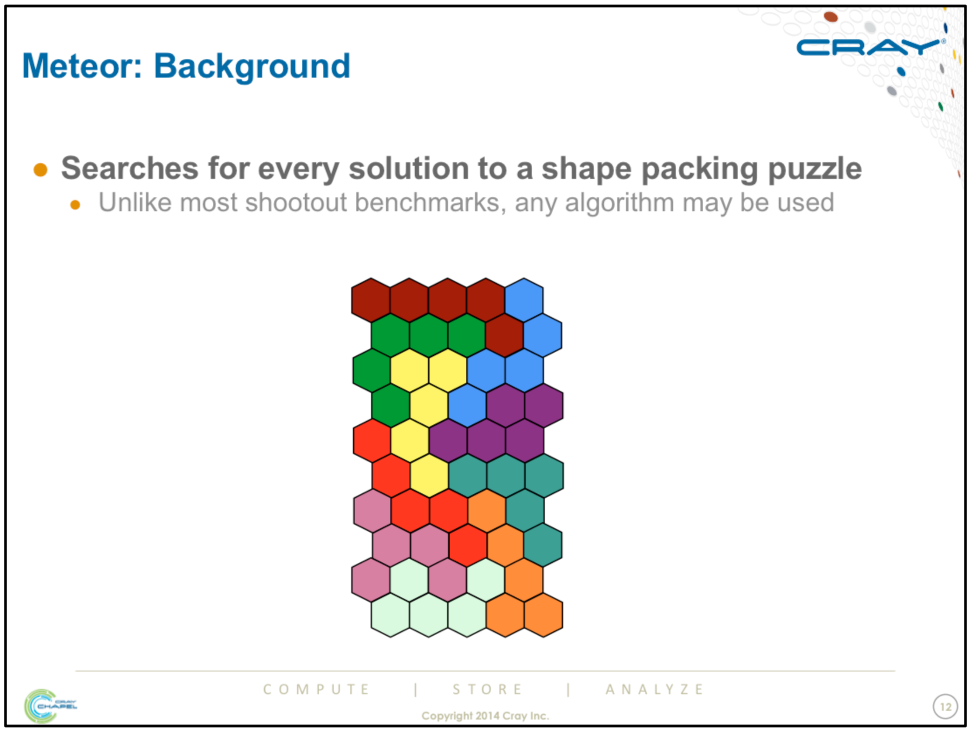
```
~~forall y in ydim do …~~
forall y in dynamic(ydim, chunkSize) do …
```

Impact: New best version



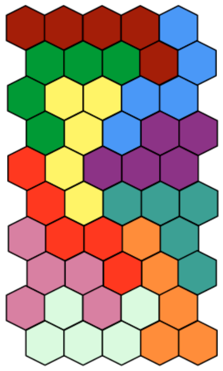


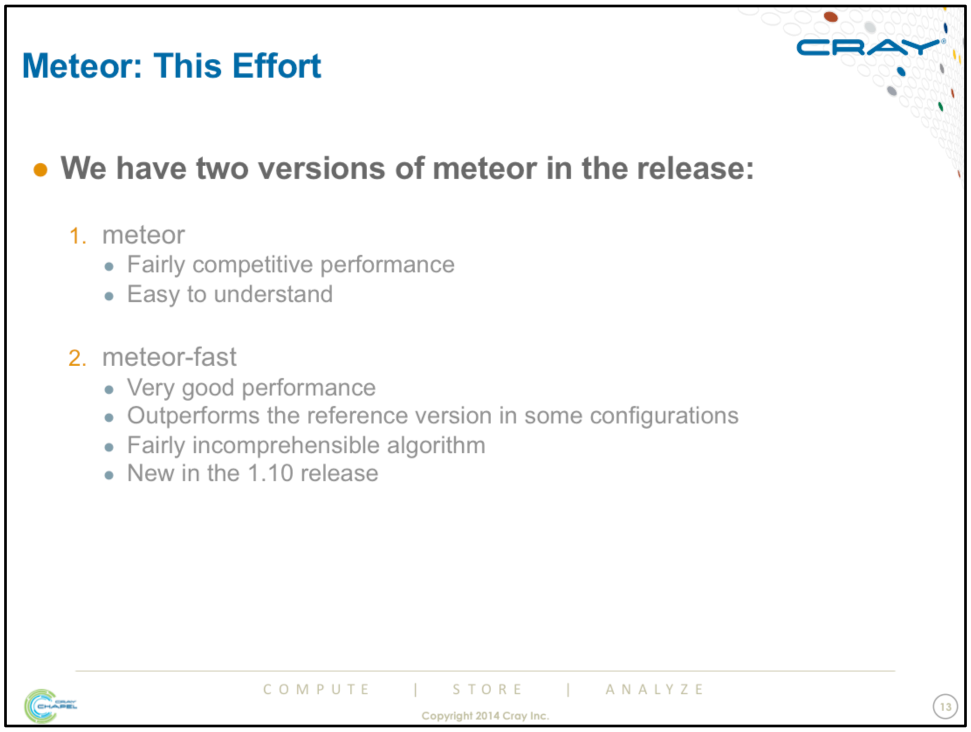
Meteor



Meteor: Background

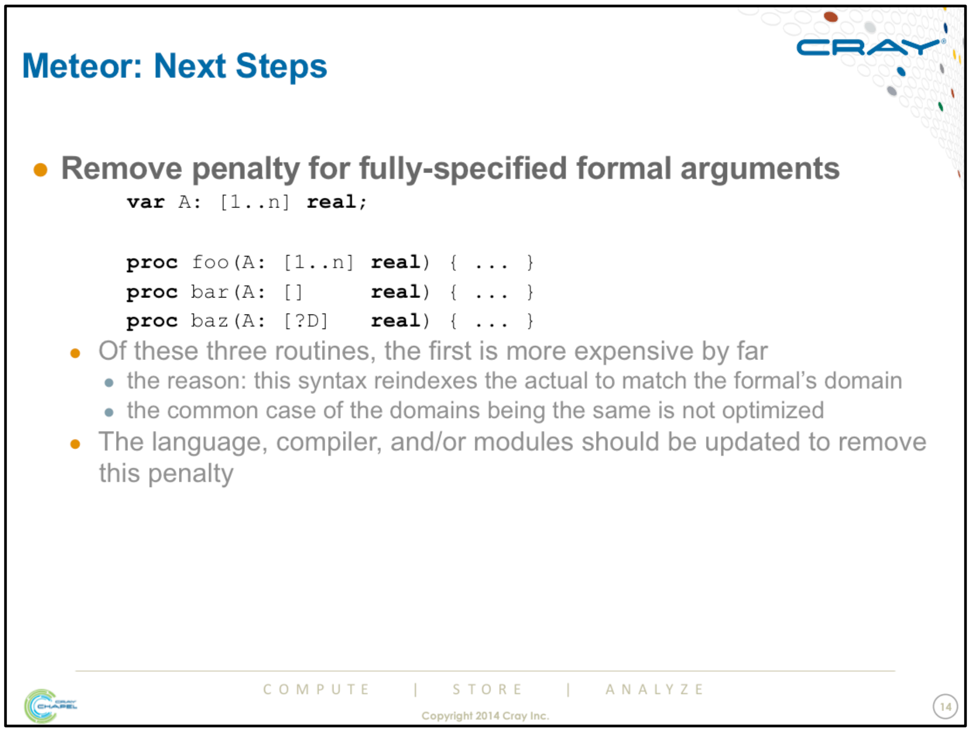
- Searches for every solution to a shape packing puzzle
  - Unlike most shootout benchmarks, any algorithm may be used





Meteor: This Effort

- **We have two versions of meteor in the release:**
  1. meteor
     - Fairly competitive performance
     - Easy to understand
  2. meteor-fast
     - Very good performance
     - Outperforms the reference version in some configurations
     - Fairly incomprehensible algorithm
     - New in the 1.10 release



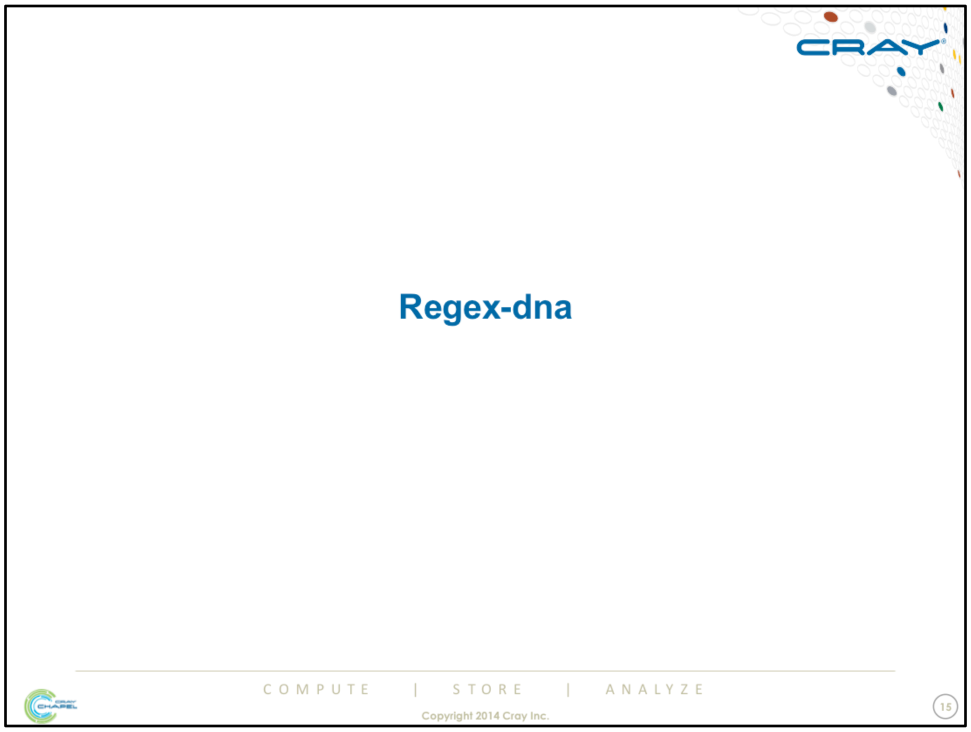
Meteor: Next Steps

- **Remove penalty for fully-specified formal arguments**

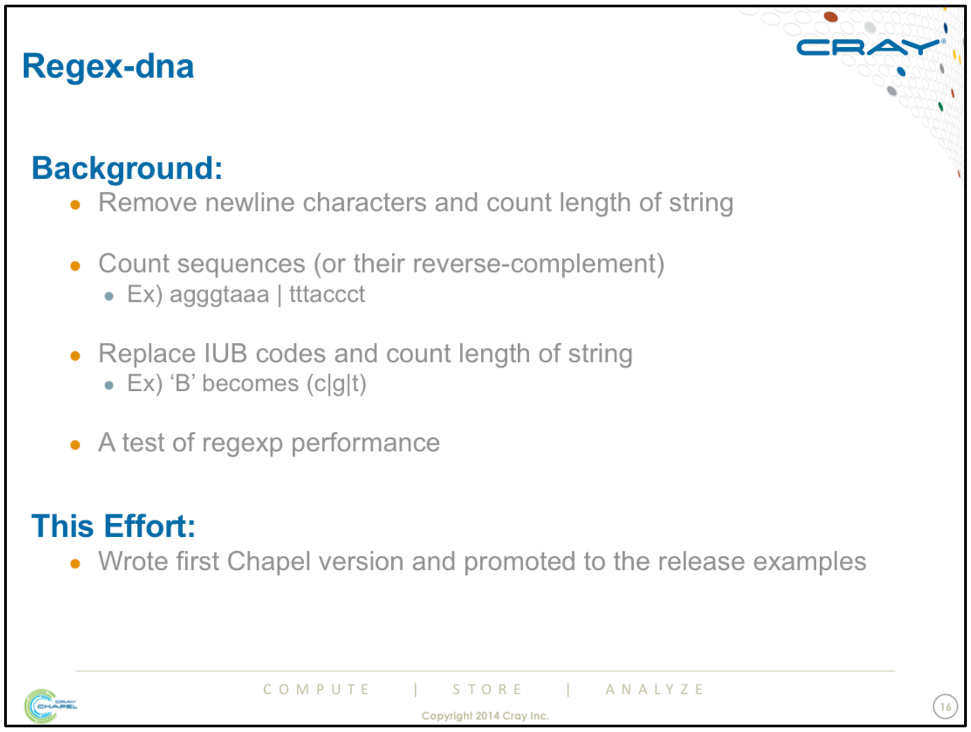
```
var A: [1..n] real;

proc foo(A: [1..n] real) { ... }
proc bar(A: []     real) { ... }
proc baz(A: [?D]   real) { ... }
```

  - Of these three routines, the first is more expensive by far
    - the reason: this syntax reindexes the actual to match the formal's domain
    - the common case of the domains being the same is not optimized
  - The language, compiler, and/or modules should be updated to remove this penalty



Regex-dna



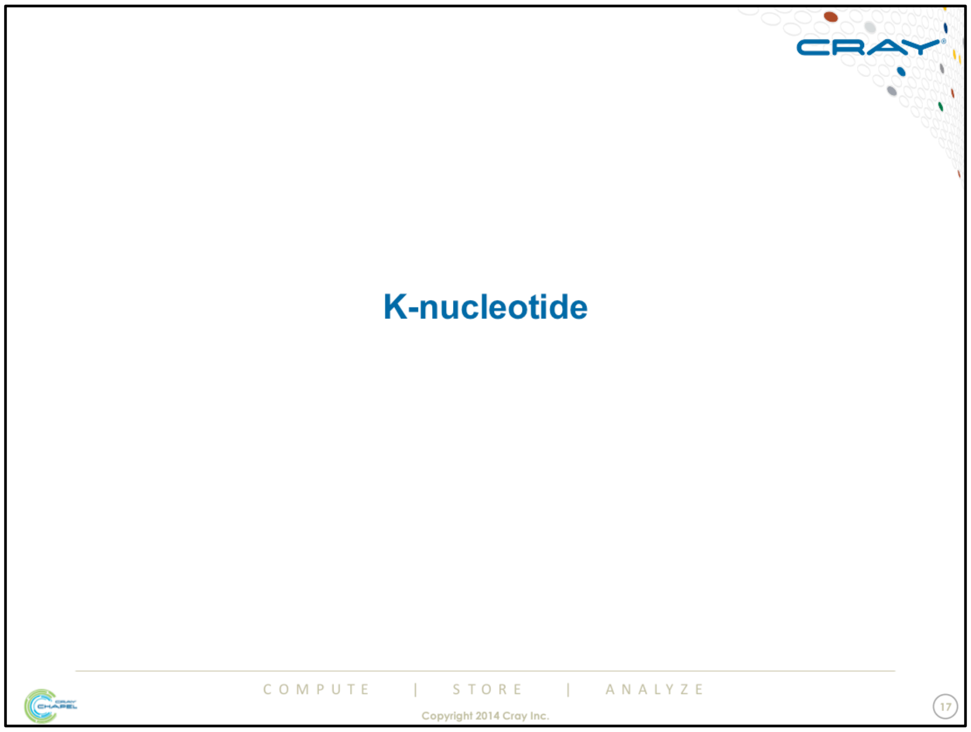
Regex-dna

Background:

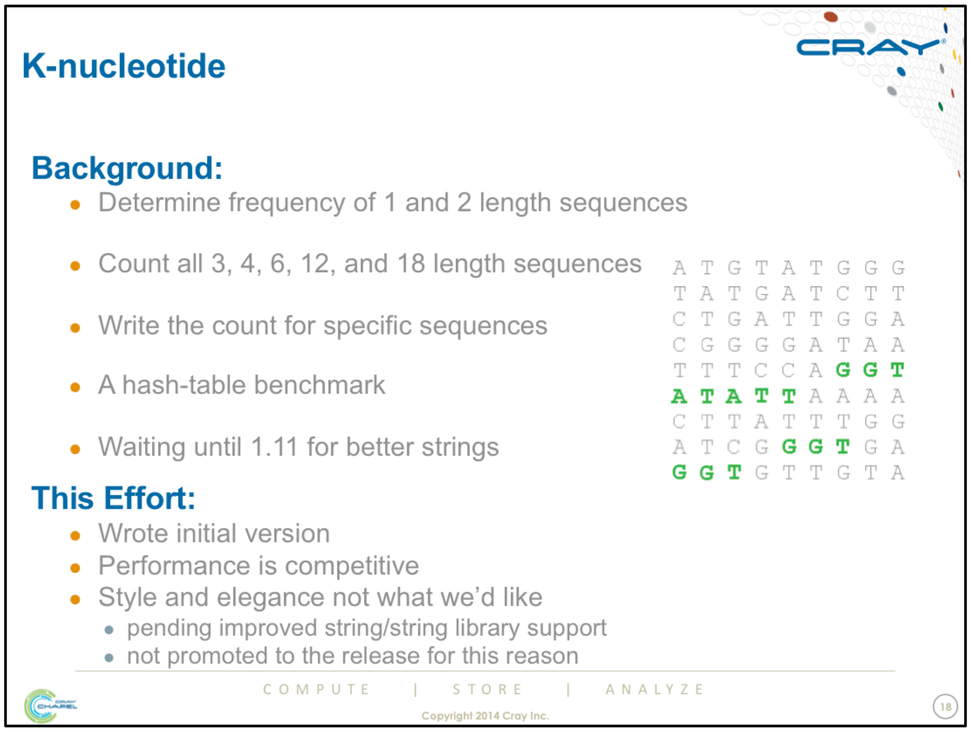
- Remove newline characters and count length of string
- Count sequences (or their reverse-complement)
  - Ex) agggtaaa | tttaccct
- Replace IUB codes and count length of string
  - Ex) 'B' becomes (c|g|t)
- A test of regexp performance

This Effort:

- Wrote first Chapel version and promoted to the release examples



K-nucleotide



K-nucleotide

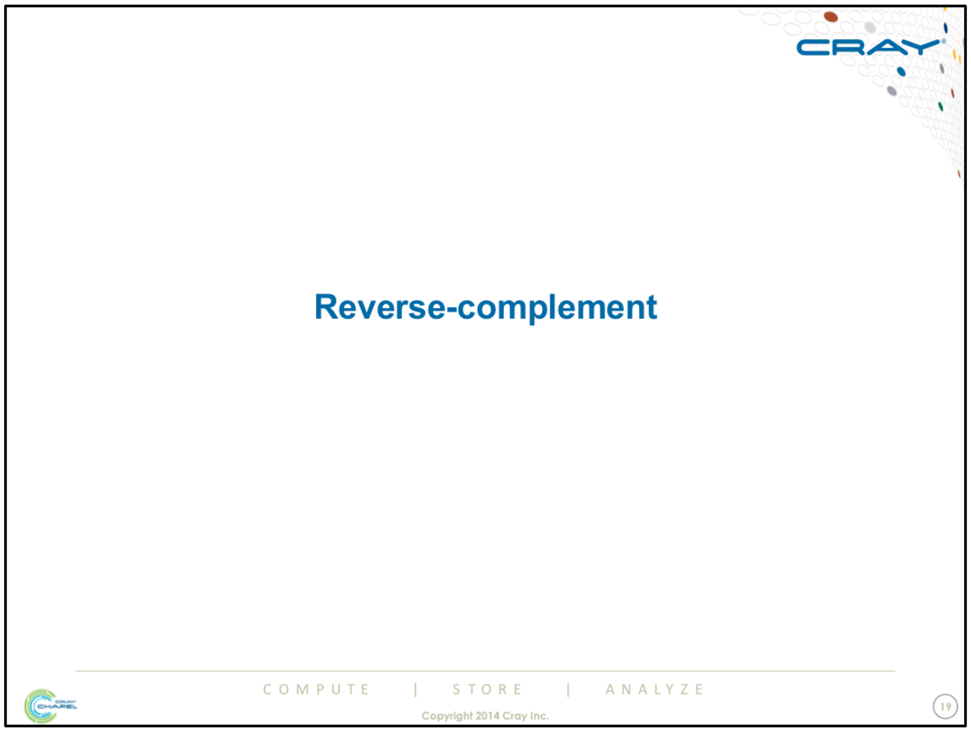
Background:

- Determine frequency of 1 and 2 length sequences
- Count all 3, 4, 6, 12, and 18 length sequences
- Write the count for specific sequences
- A hash-table benchmark
- Waiting until 1.11 for better strings

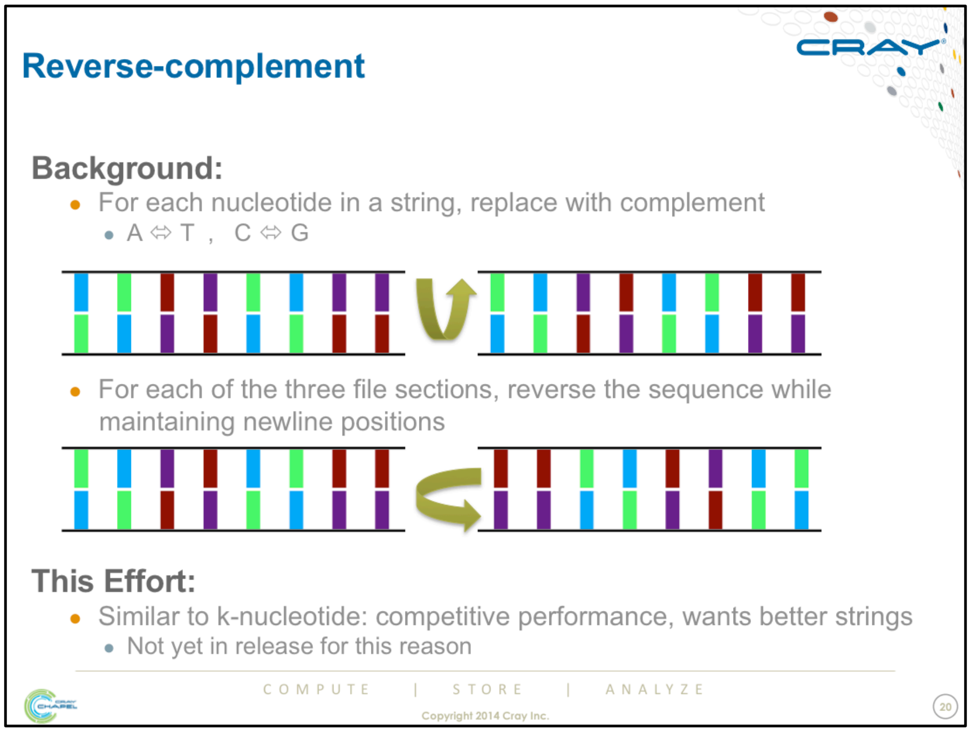
```
A T G T A T G G G
T A T G A T C T T
C T G A T T G G A
C G G G G A T A A
T T T C C A G G T
A T A T T A A A A
C T T A T T T G G
A T C G G G T G A
G G T G T T G T A
```

This Effort:

- Wrote initial version
- Performance is competitive
- Style and elegance not what we'd like
  - pending improved string/string library support
  - not promoted to the release for this reason



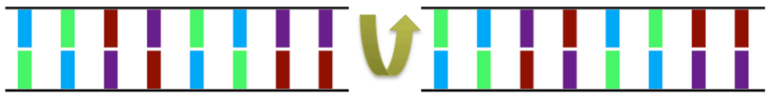
Reverse-complement



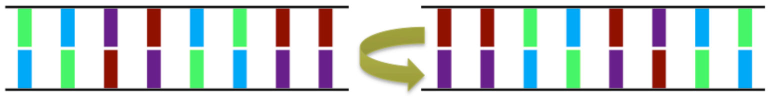
Reverse-complement

Background:

- For each nucleotide in a string, replace with complement
  - A ⇔ T , C ⇔ G

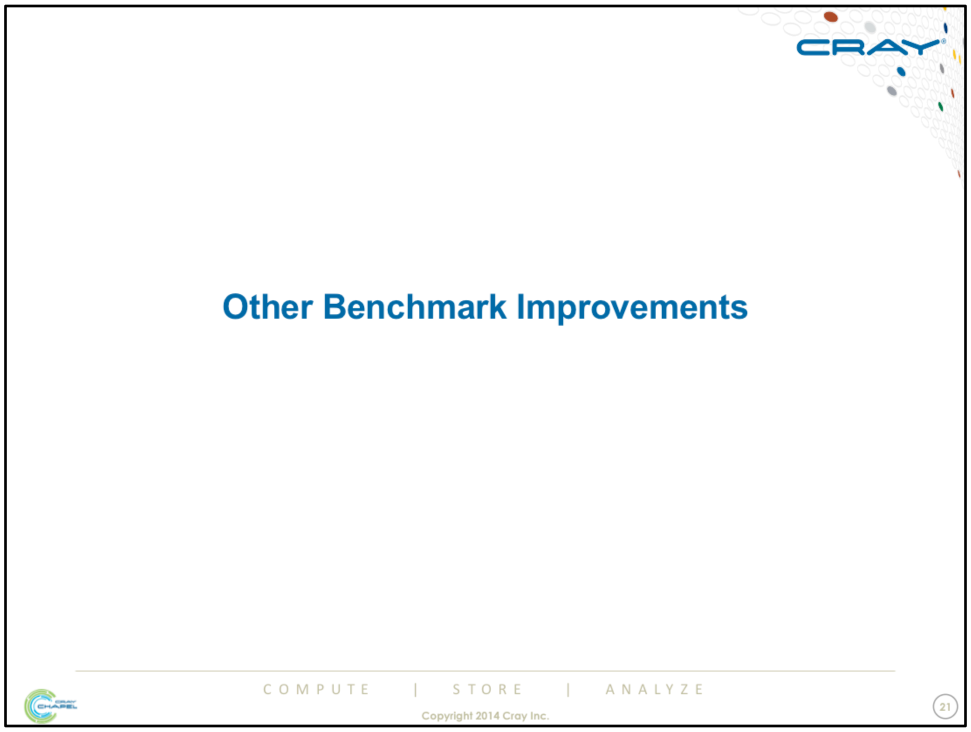


- For each of the three file sections, reverse the sequence while maintaining newline positions

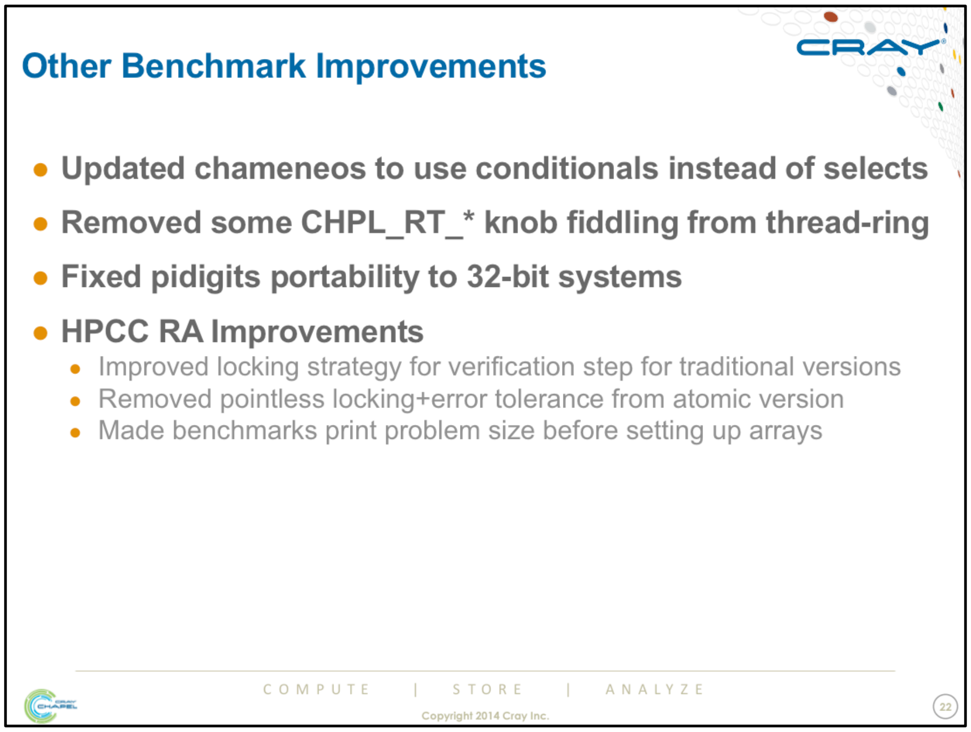


This Effort:

- Similar to k-nucleotide: competitive performance, wants better strings
  - Not yet in release for this reason

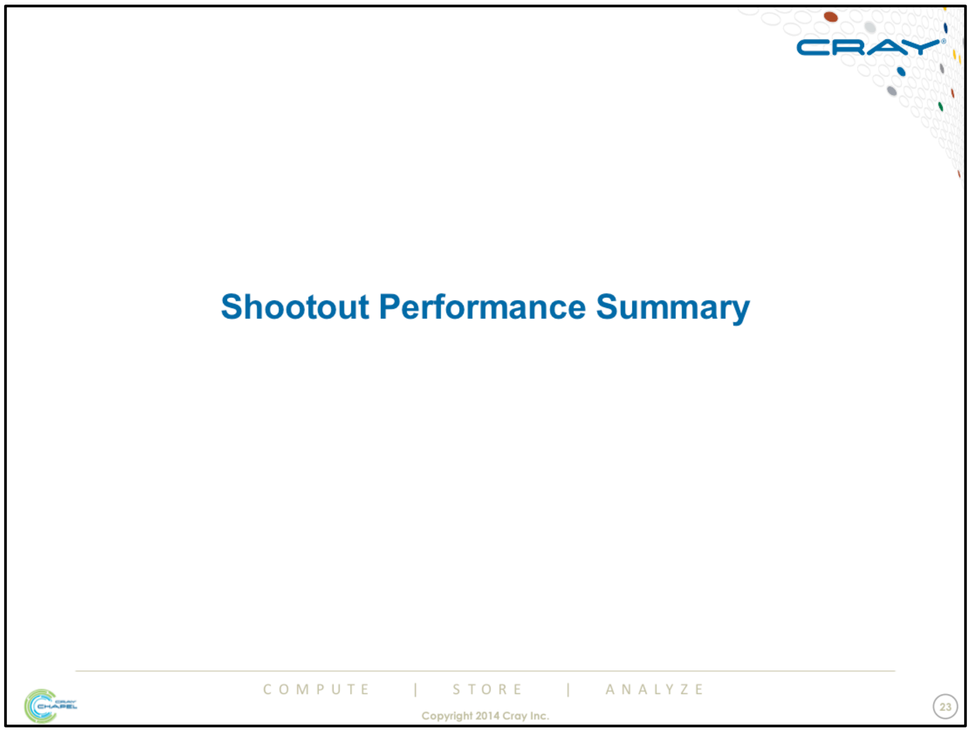


Other Benchmark Improvements

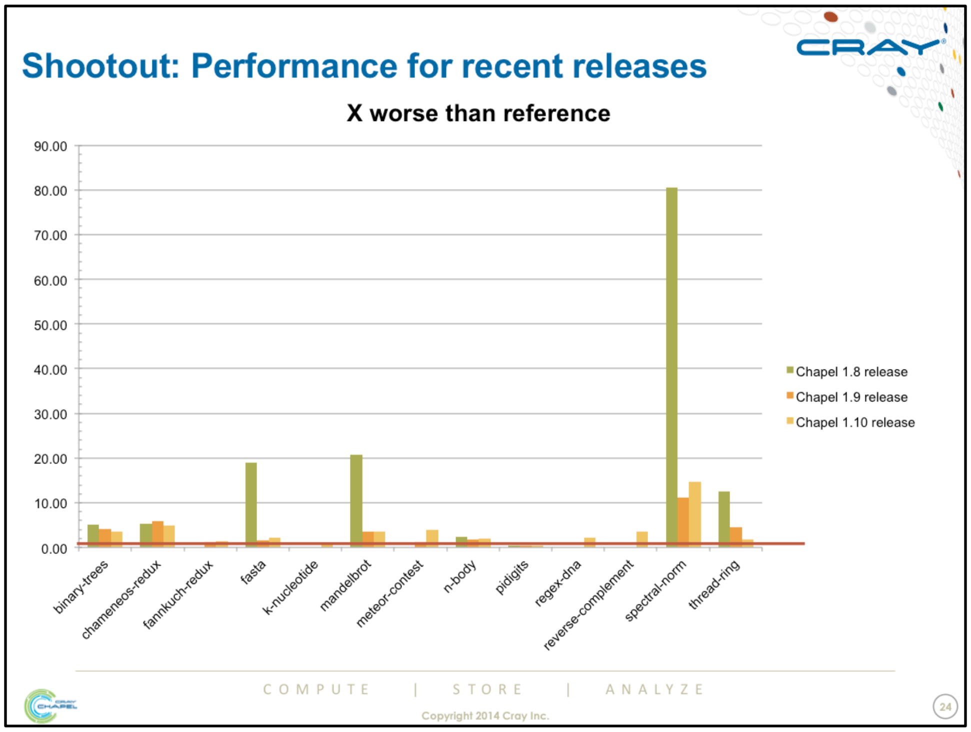


Other Benchmark Improvements

- Updated chameneos to use conditionals instead of selects
- Removed some CHPL_RT_* knob fiddling from thread-ring
- Fixed pidigits portability to 32-bit systems
- HPCC RA Improvements
  - Improved locking strategy for verification step for traditional versions
  - Removed pointless locking+error tolerance from atomic version
  - Made benchmarks print problem size before setting up arrays

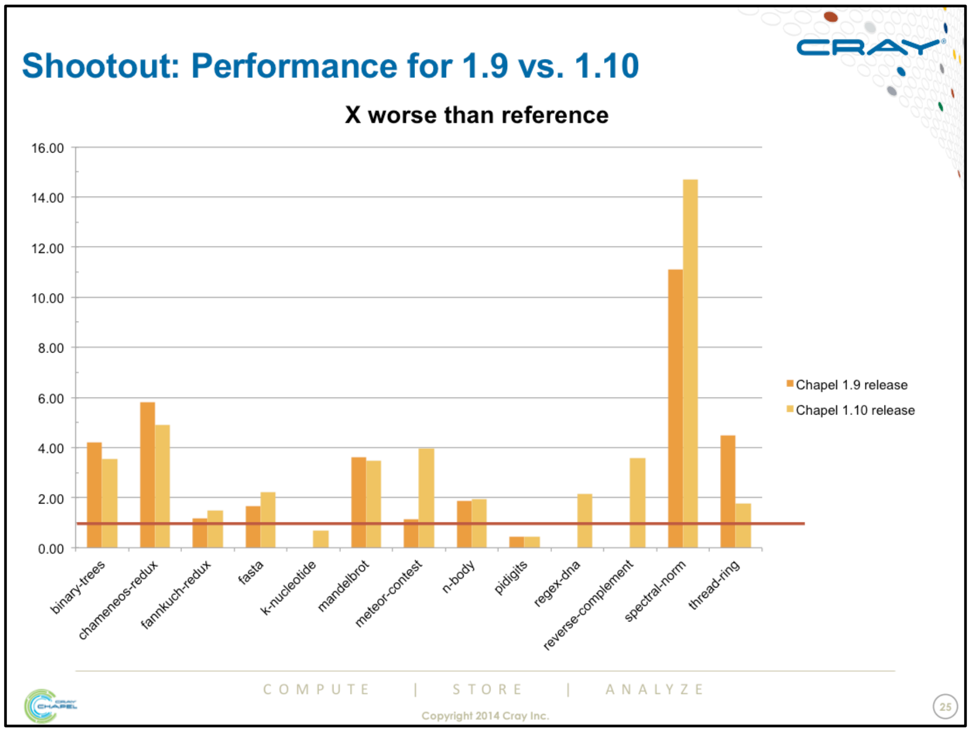


Shootout Performance Summary



Shootout: Performance for recent releases

On this slide, the numbers reported are those measured with the version of gcc available at the time of the release. For 1.8 and 1.9, that version is gcc 4.3, while the version for 1.10 and the reference implementation is gcc 4.7

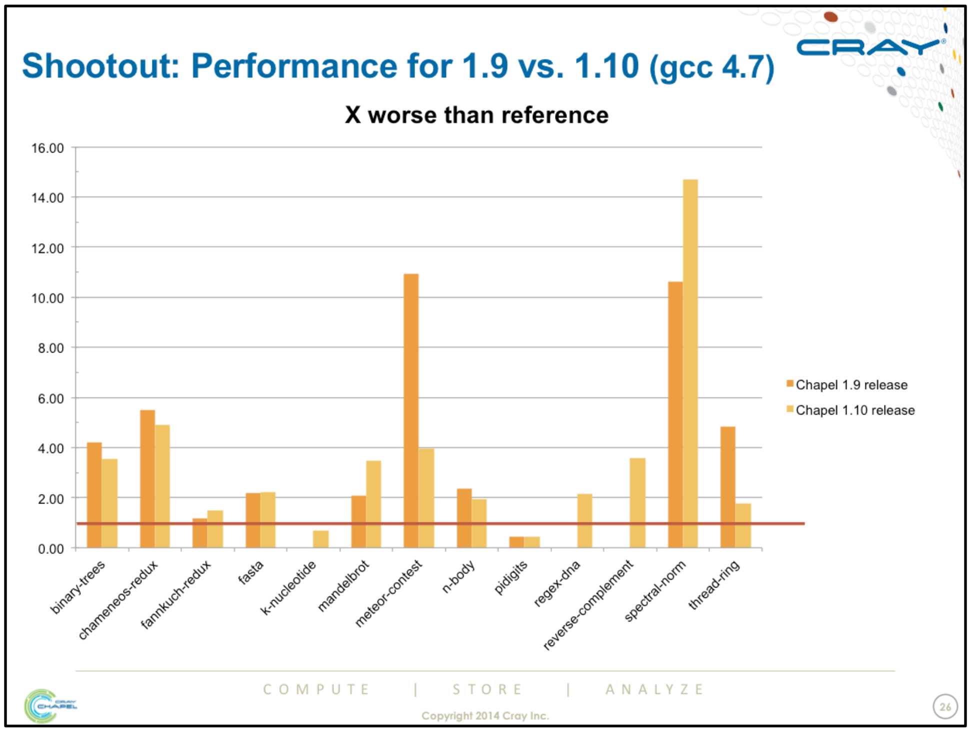


Shootout: Performance for 1.9 vs. 1.10

COMPUTE | STORE | ANALYZE

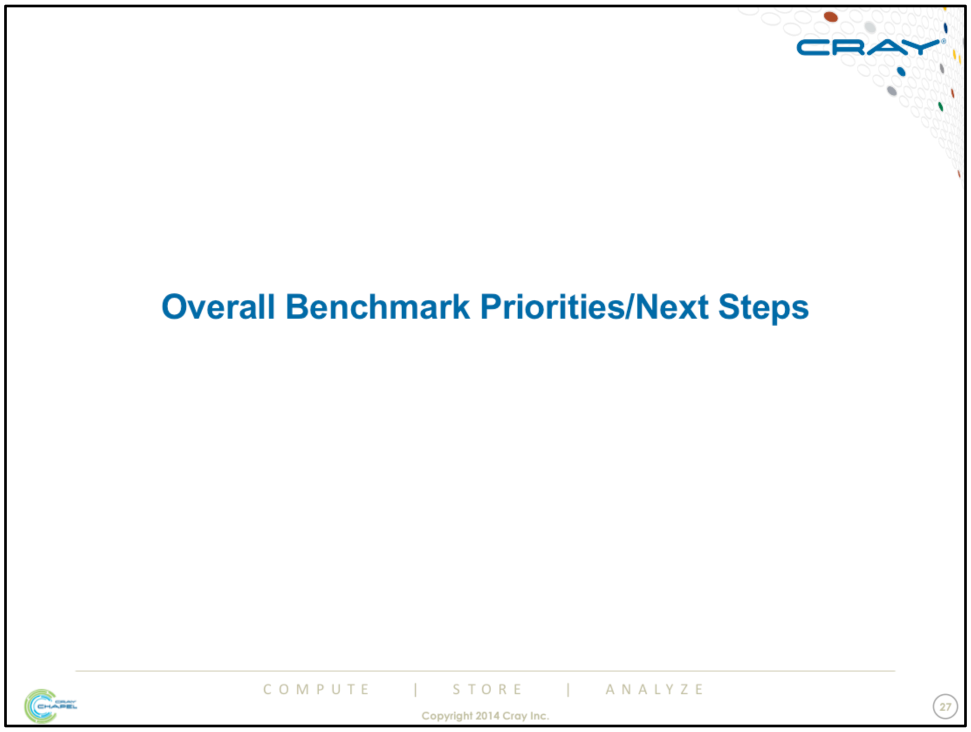
The results on this slide are identical to the previous, but the 1.8 numbers have been removed for clarity

On this slide, the numbers reported are those measured with the version of gcc available at the time of the release. For 1.9, that version is gcc 4.3, while the version for 1.10 and the reference is gcc 4.7

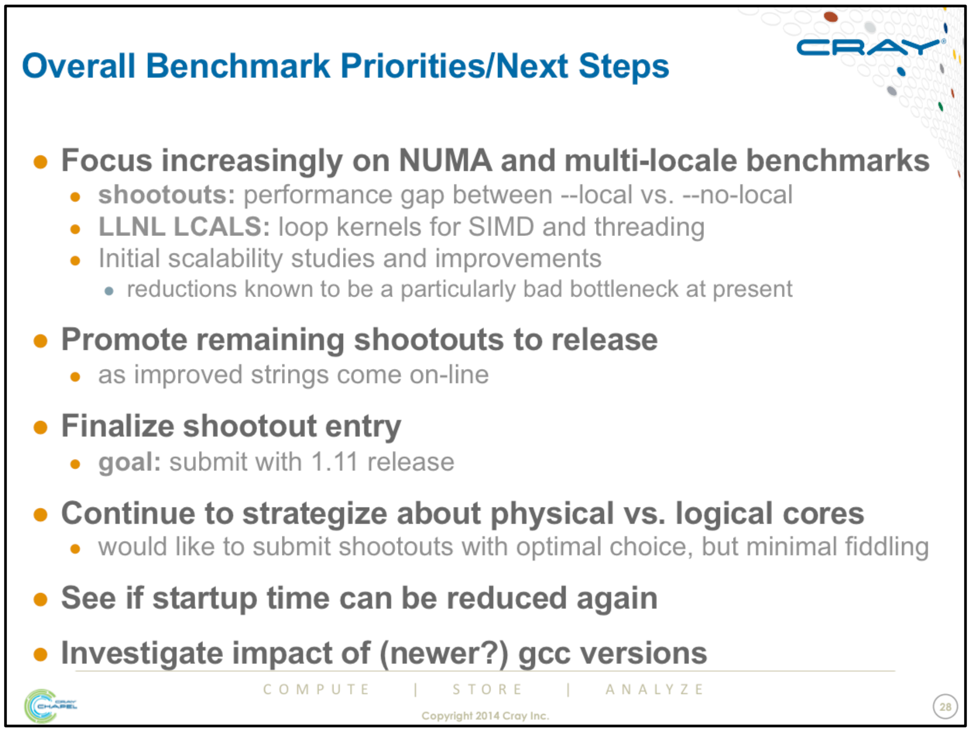


Shootout: Performance for 1.9 vs. 1.10 (gcc 4.7)

On this slide, the version 1.9 numbers were re-gathered with gcc 4.7 to remove differences in the back-end compiler between the last two releases.

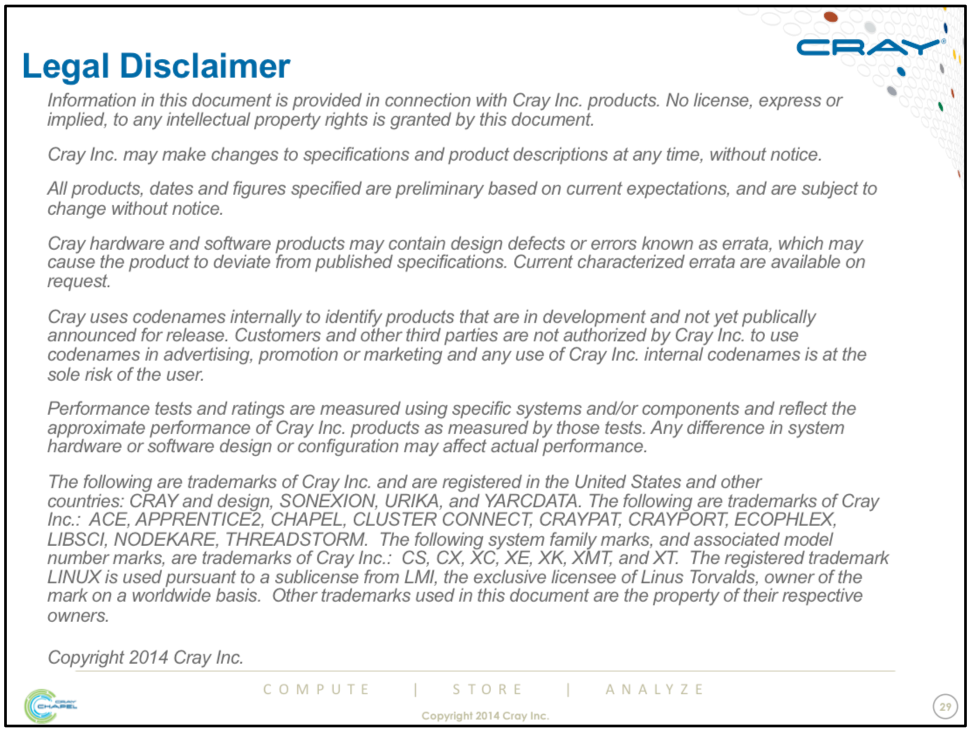


Overall Benchmark Priorities/Next Steps



Overall Benchmark Priorities/Next Steps

- **Focus increasingly on NUMA and multi-locale benchmarks**
  - **shootouts:** performance gap between --local vs. --no-local
  - **LLNL LCALS:** loop kernels for SIMD and threading
  - Initial scalability studies and improvements
    - reductions known to be a particularly bad bottleneck at present
- **Promote remaining shootouts to release**
  - as improved strings come on-line
- **Finalize shootout entry**
  - **goal:** submit with 1.11 release
- **Continue to strategize about physical vs. logical cores**
  - would like to submit shootouts with optimal choice, but minimal fiddling
- **See if startup time can be reduced again**
- **Investigate impact of (newer?) gcc versions**



Legal Disclaimer

CRAY CHAPEL

CRAY
THE SUPERCOMPUTER COMPANY

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