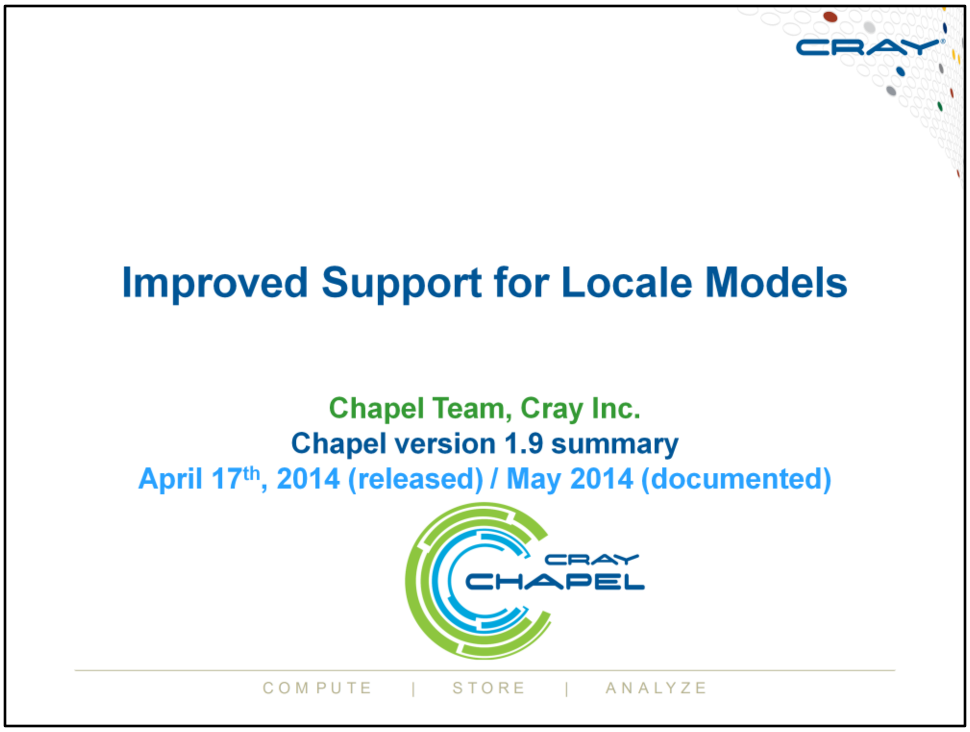
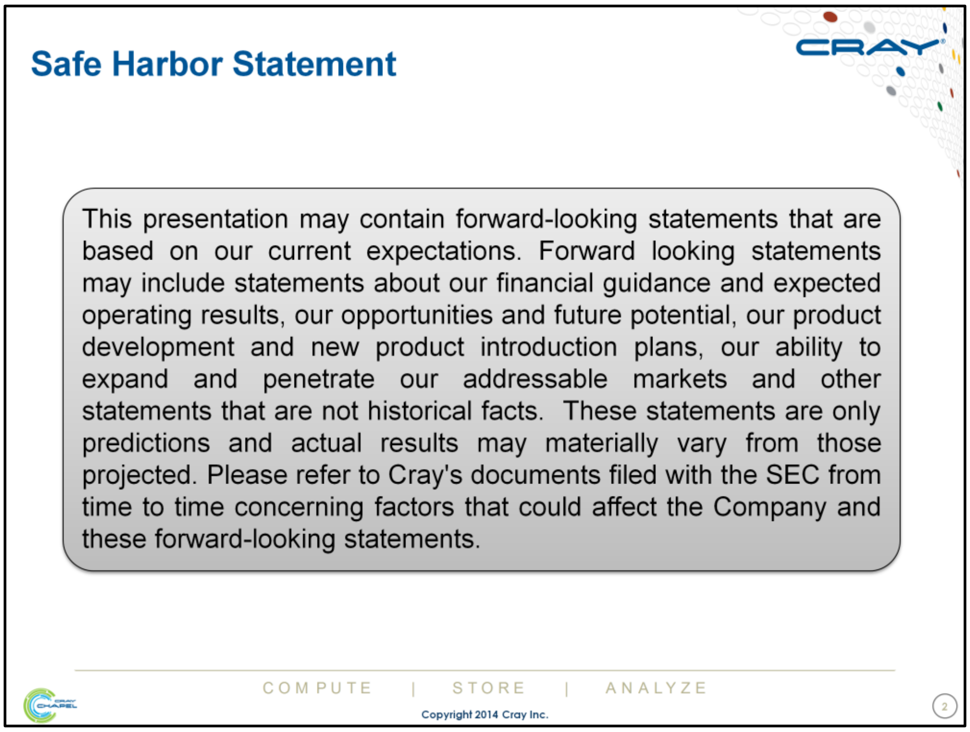


Improved Support for Locale Models

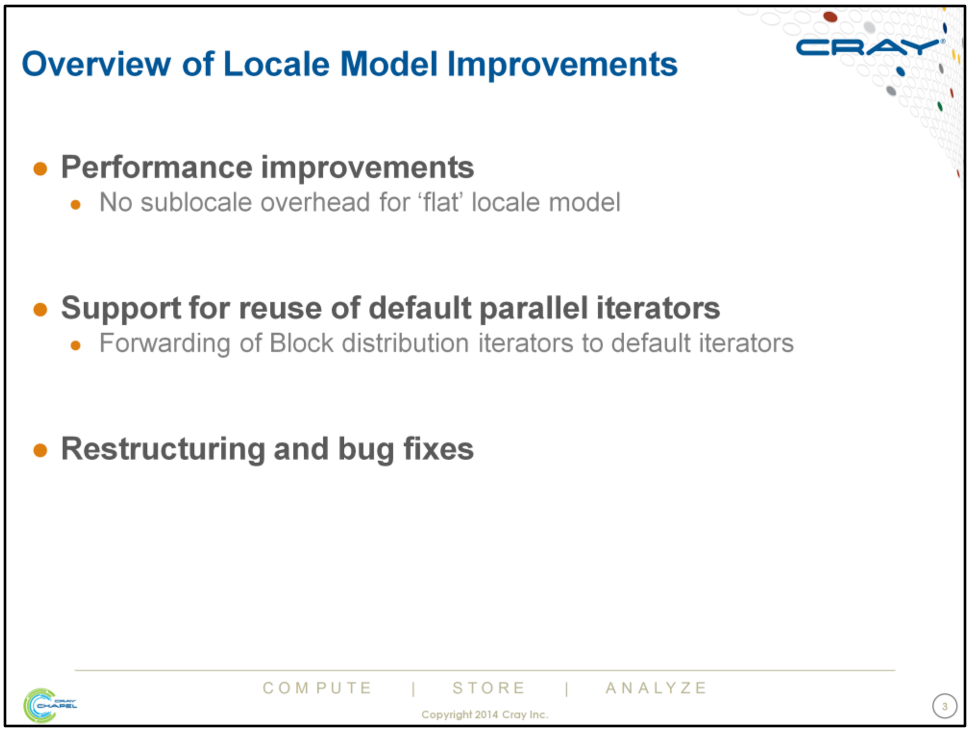
Chapel Team, Cray Inc.
Chapel version 1.9 summary
April 17th, 2014 (released) / May 2014 (documented)

COMPUTE | STORE | ANALYZE



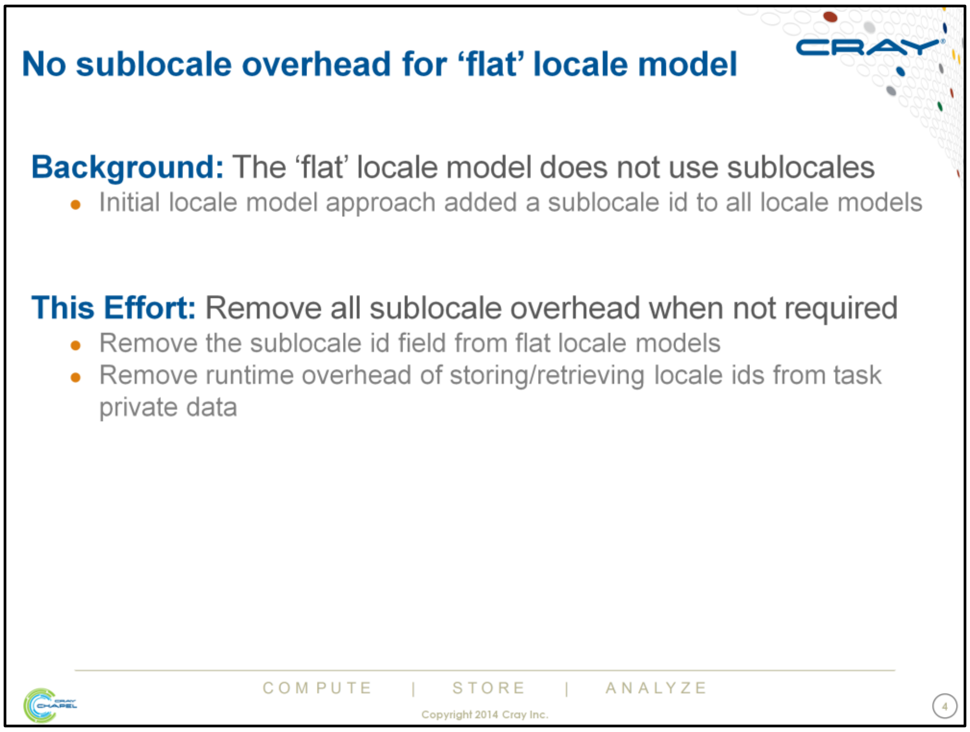
Safe Harbor Statement

COMPUTE | STORE | ANALYZE



Overview of Locale Model Improvements

- **Performance improvements**
  - No sublocale overhead for ‘flat’ locale model
- **Support for reuse of default parallel iterators**
  - Forwarding of Block distribution iterators to default iterators
- **Restructuring and bug fixes**



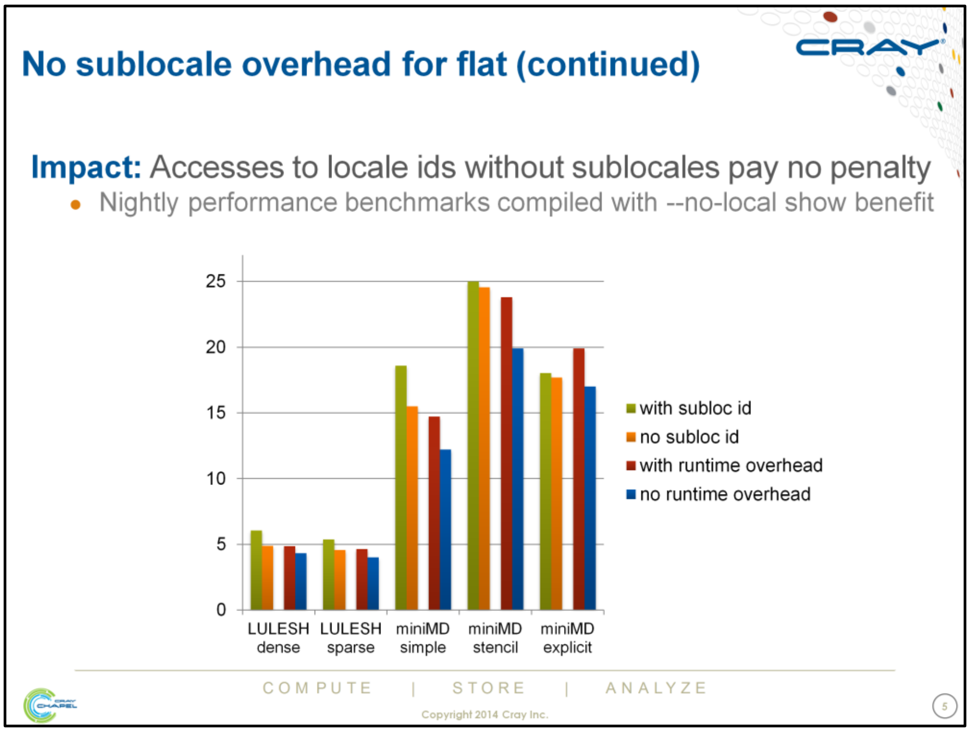
No sublocale overhead for 'flat' locale model

Background: The 'flat' locale model does not use sublocales

- Initial locale model approach added a sublocale id to all locale models

This Effort: Remove all sublocale overhead when not required

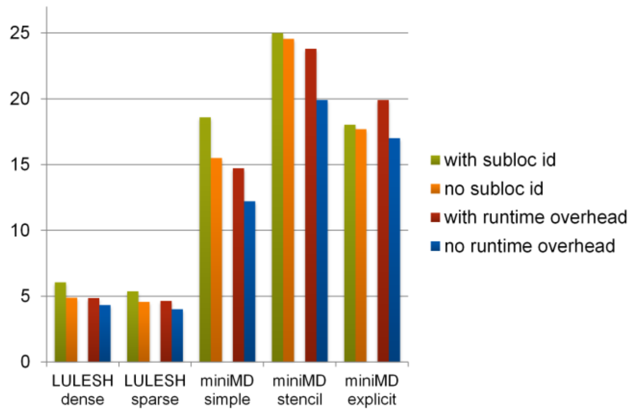
- Remove the sublocale id field from flat locale models
- Remove runtime overhead of storing/retrieving locale ids from task private data



No sublocale overhead for flat (continued)

Impact: Accesses to locale ids without sublocales pay no penalty

- Nightly performance benchmarks compiled with --no-local show benefit



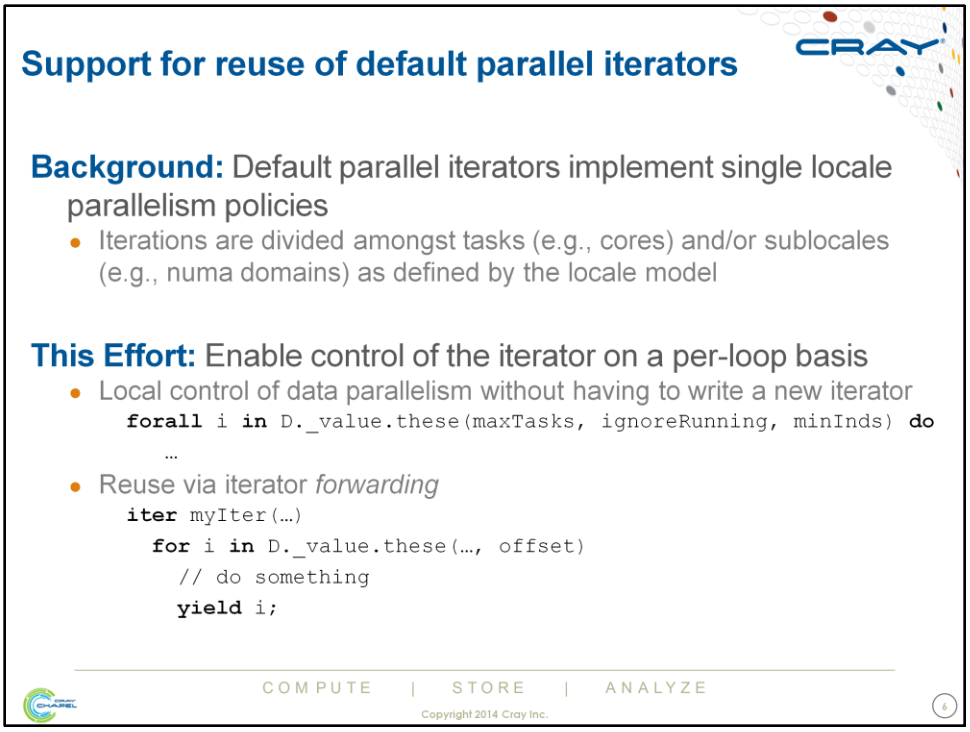
COMPUTE | STORE | ANALYZE

5

- All multilocale execution would see benefit from these changes, but we currently do not do nightly multilocale performance testing. Single locale performance tests compiled with the –no-local flag give a sense of the penalties paid for multilocale execution.

- The bars correspond to execution time before and after each of the two changes described in the previous slide.

- The lulesh versions use block distribution (dense and sparse). The miniMD versions use the block distribution.



Support for reuse of default parallel iterators

Background: Default parallel iterators implement single locale parallelism policies

- Iterations are divided amongst tasks (e.g., cores) and/or sublocales (e.g., numa domains) as defined by the locale model

This Effort: Enable control of the iterator on a per-loop basis

- Local control of data parallelism without having to write a new iterator

```
forall i in D._value.these(maxTasks, ignoreRunning, minInds) do
  …
```

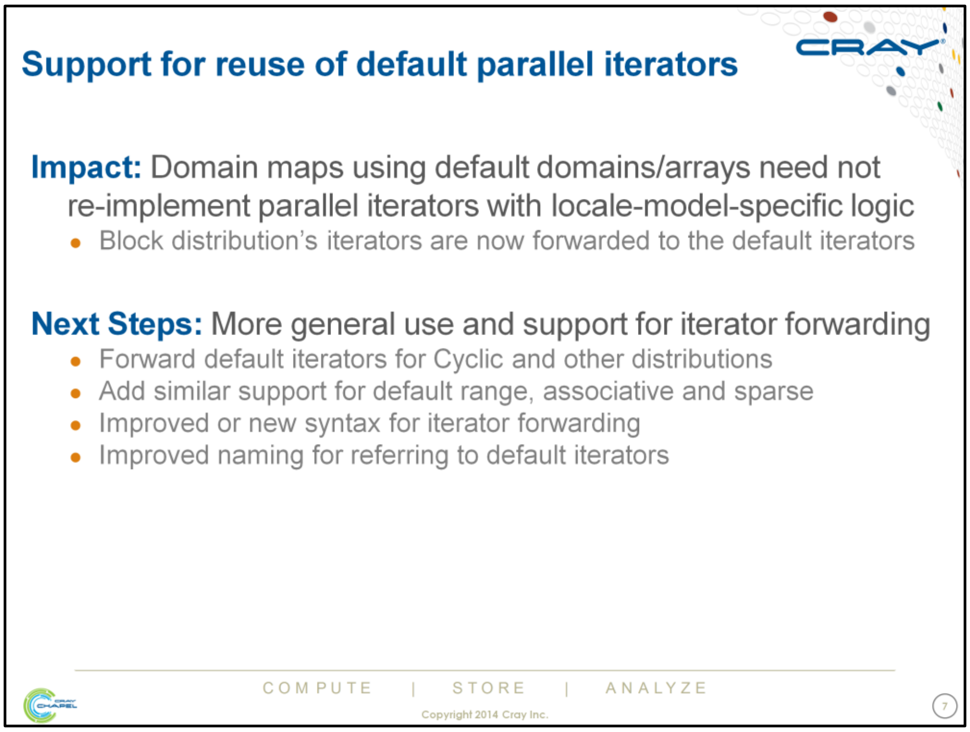
- Reuse via iterator *forwarding*

```
iter myIter(…)
  for i in D._value.these(…, offset)
    // do something
    yield i;
```

COMPUTE | STORE | ANALYZE

- Prior to this change, parallel iteration was controlled by the global configuration variables (dataParTasksPerLocale, dataParIgnoreRunning, and dataParMinGranularity) and could not be changed on a per loop basis. Adding these control knobs as well as a base offset as optional arguments to the iterator's signature enabled the ability to reuse the default iterators.

- Iterator forwarding is a way for an iterators that yield the same or similar values as an existing iterator *forward* to the original iterator, enabling reuse.


Support for reuse of default parallel iterators

Impact: Domain maps using default domains/arrays need not re-implement parallel iterators with locale-model-specific logic

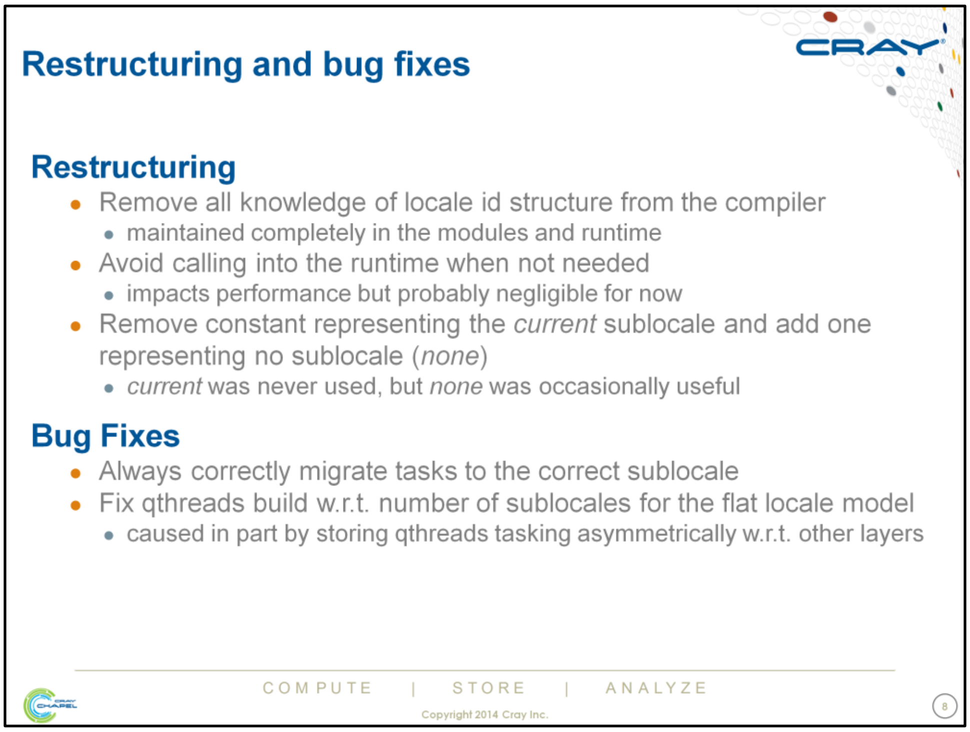
- Block distribution's iterators are now forwarded to the default iterators

Next Steps: More general use and support for iterator forwarding

- Forward default iterators for Cyclic and other distributions
- Add similar support for default range, associative and sparse
- Improved or new syntax for iterator forwarding
- Improved naming for referring to default iterators

COMPUTE | STORE | ANALYZE

- In the Block distribution's iterators, we now simply call out to the default iterators. Prior to this change, the logic in the default iterator was replicated (and not precisely maintained, so improvements to the default case were not automatically propagated to / inherited by Block).
- The point about improved naming refers to the need to reference D._value.these on the previous slide in order to forward



Restructuring and bug fixes

Restructuring

- Remove all knowledge of locale id structure from the compiler
  - maintained completely in the modules and runtime
- Avoid calling into the runtime when not needed
  - impacts performance but probably negligible for now
- Remove constant representing the *current* sublocale and add one representing no sublocale (*none*)
  - *current* was never used, but *none* was occasionally useful

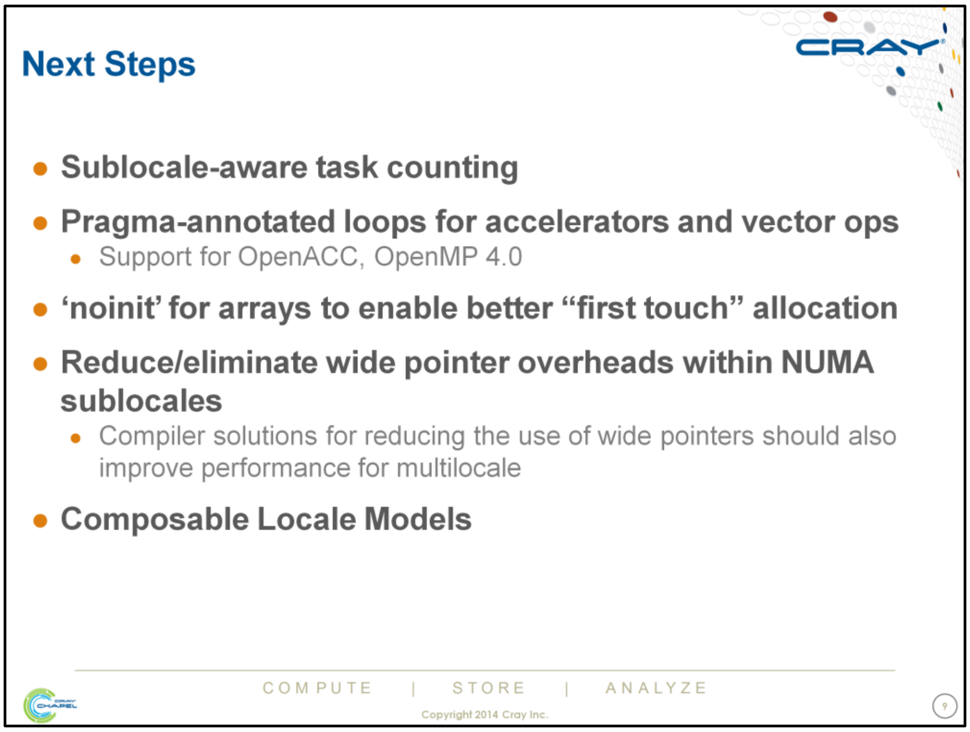
Bug Fixes

- Always correctly migrate tasks to the correct sublocale
- Fix qthreads build w.r.t. number of sublocales for the flat locale model
  - caused in part by storing qthreads tasking asymmetrically w.r.t. other layers

COMPUTE | STORE | ANALYZE

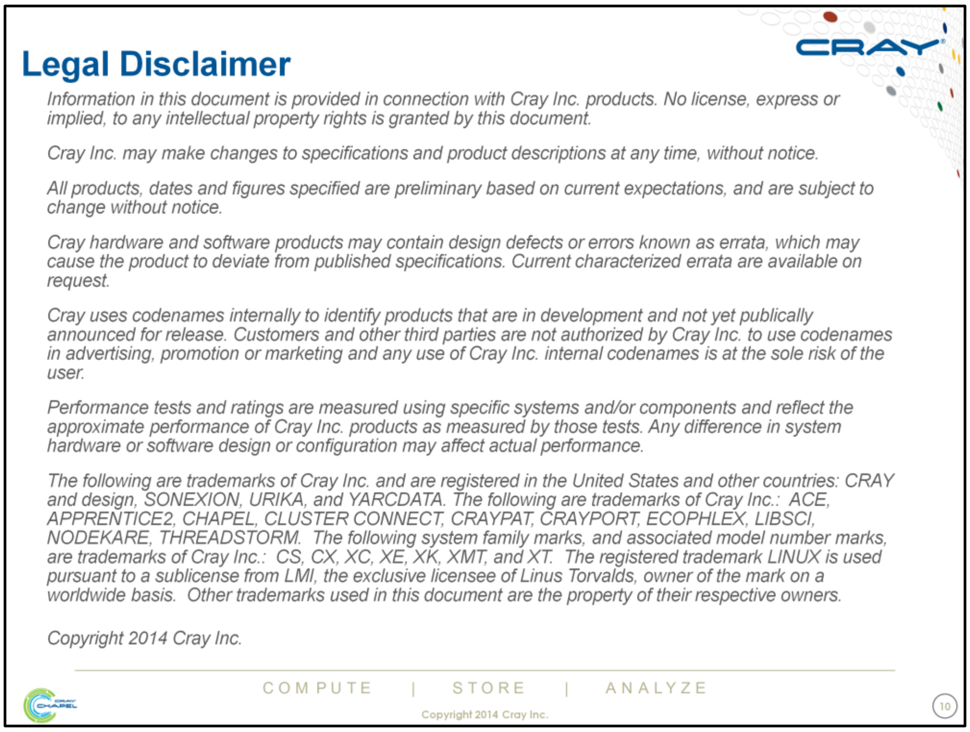
8

- As we gained more experience with locale models, we were able to fix or improve some of our earlier design choices.
- The asymmetric storage issue relates to the fact that, for historical reasons, the Chapel runtime interface for Qthreads lives in the third-party directory rather than in the runtime directory as a sibling to fifo, massivethreads, etc. Contributor agreement issues have prevented us from fixing this to date, but moving to the Apache contributor agreement should alleviate this.



Next Steps

- **Sublocale-aware task counting**
- **Pragma-annotated loops for accelerators and vector ops**
  - Support for OpenACC, OpenMP 4.0
- **‘noinit’ for arrays to enable better “first touch” allocation**
- **Reduce/eliminate wide pointer overheads within NUMA sublocales**
  - Compiler solutions for reducing the use of wide pointers should also improve performance for multilocale
- **Composable Locale Models**



Legal Disclaimer

CRAY CHAPEL

CRAY
THE SUPERCOMPUTER COMPANY

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