Improved Support for Locale Models

Chapel Team, Cray Inc.
Chapel version 1.9 summary
April 17th, 2014 (released) / May 2014 (documented)
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.
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
• 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.
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.
• 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
• 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 multilocal
- Composable Locale Models