Memory Leaks

Chapel Team, Cray Inc.
Chapel version 1.14
October 6, 2016

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

- **Sync/Single: A record-wrapped class**
- **Memory Leaks**

Sync/Single: A record-wrapped class

Sync/Single: Background

- **Historically a type with special compiler support**
- **Defined as a class**
  - with two fields
    - a generic field :- constrained to primitive types and classes
    - an internal synchronization field
  - critical methods implemented using compiler primitives
  - compiler-based memory management
    - but only worked well for simpler cases
- **A major source of leaks**
  - The third largest category when counting tests with leaks
    - accounted for approximately 14% of leaking tests
  - Not intended to be a type that should be deleted by user

Sync/Single: This Effort

- **Convert to a record-wrapped class**
  - The record:
    - implements the user facing API
    - wraps an instance of a class
    - the defining record *owns* memory management of the instance
    - a copy of the record merely references the instance
    - Chapel semantics ensure copies will not outlive the owning record
  - The class
    - provides the unique *identity* required for the synchronization state
    - is derived from the previous implementation
    - uses extern procedure declarations in place of former compiler primitives

Sync/Single: This Effort

- **Modified the handling of default intents**
  - The default formal intent for sync/single is **ref**
  - The default formal intent for user defined records is **const ref**
  - Introduced a pragma to override the default intent

- **Modified the Remote Value Forwarding optimization**
  - Goal: send variables' values with active messages for on-clauses
    - avoids communication to read such variables later
    - can only be done when safe according to MCM
  - Disabled when body of on-statement includes sync/single (recursively)
    - old approach: identify functions with certain sync primitives
    - new approach: identify methods on sync/single types

Sync/Single: Status and Next Steps

- **Status**
  - Removed leaks for approximately 200 tests
  - Removed special compiler logic/primitives for sync and single
  - No evidence of performance regression

- **Next steps**
  - Revisit as a use case for delegation / smart pointers

Memory Leaks

Memory Leaks: Background

- **Memory leak statistics are collected every night**
  - Performance team reviews every week
  - Currently gathering single locale leaks only
- **Two metrics are tracked**
  1. Total bytes leaked
     - Impacted by test parameters (e.g., choice of array sizes)
  2. Number of tests with leaks
     - Some tests run in multiple variations, so one oversight leads to many leaks

	1.13
Tests run	4,804
Total memory allocated (MiB)	36,749
Total memory leaked (MiB)	942
Tests with leaks	1,193

1 MiB = 1024 x 1024 bytes

Memory Leaks: This Effort

- **Categorized primary causes of leaks (April 2016)**

Source	Count	%	Status
User fails to reclaim memory	~400	37.3	Largely fixed
Sync/single	~155	14.4	Fixed
Tuples of records	~100	9.3	Fixed
main(args : [] string)	~20	1.9	Fixed
Distributed arrays	~190	17.7	Soon
Initialization of generic fields	~80	7.5	Unchanged
Field initializer	~40	3.7	Unchanged
First-class functions	~25	2.3	Unchanged
Runtime types	~15	1.4	Unchanged
Misc and further classification required	~50	4.7	Unchanged
Total	**1,073**		

Memory Leaks: This Effort

- **Reduced total bytes (MiB) leaked**
  - Dominated by a few tests of distributed arrays
    - Continues to be true in release
    - Wrapping up work with a major impact on array/domain leaks*

	1.13	1.14	Soon*
Total memory leaked (MiB)	942	951	47
Num tests that leak > 5 MiB	7	7	2
Fraction of all leaks	92.0%	91.1%	34.8%
Num tests that leak > 1MiB	31	35	16
Fraction of all leaks	97.8%	97.8%	80.1%

1 MiB = 1024 x 1024 bytes

* This refers to the array reimplementation work described in the ongoing efforts slides, now on master, but still underway when these numbers were gathered.

Memory Leaks: This Effort

- Reduce number of tests with leaks

1.13	1.14	Soon
1,193	539	330

- Coarse counts of tests whose primary leaks are due to:

Source	1.14	Soon
Distributed arrays	~200	~40
Initialization of generic fields	~80	~80
App fails to reclaim memory	~45	~50
First class functions	35	35
Various/unclassified	~180	~125
	539	330

Memory Leaks: Status and Next Steps

Status:

- Release 1.14
  - Leak by total bytes largely unchanged
  - Leak by number of tests less than 1/2 of 1.13 (45%)
- Soon
  - Leak by total bytes dramatically reduced
  - Leak by number of tests less than 1/3 of 1.13 (28%)

Next Steps:

- Continue to eliminate remaining leaks
  - prioritize based on impact and complexity

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY