Runtime and Third-Party Improvements

Chapel Team, Cray Inc.

Chapel version 1.15

April 6, 2017

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

- **Ugni/Muxed Background**
- **Communication Improvements**
  - Open Source Ugni
  - Ugni Registration Limit
- **Tasking Improvements**
  - Register Qthreads Task Stacks
  - Limit Qthreads Memory Pool
  - Deprecate Muxed
- **Other Runtime Improvements**
- **Other Third-Party Improvements**

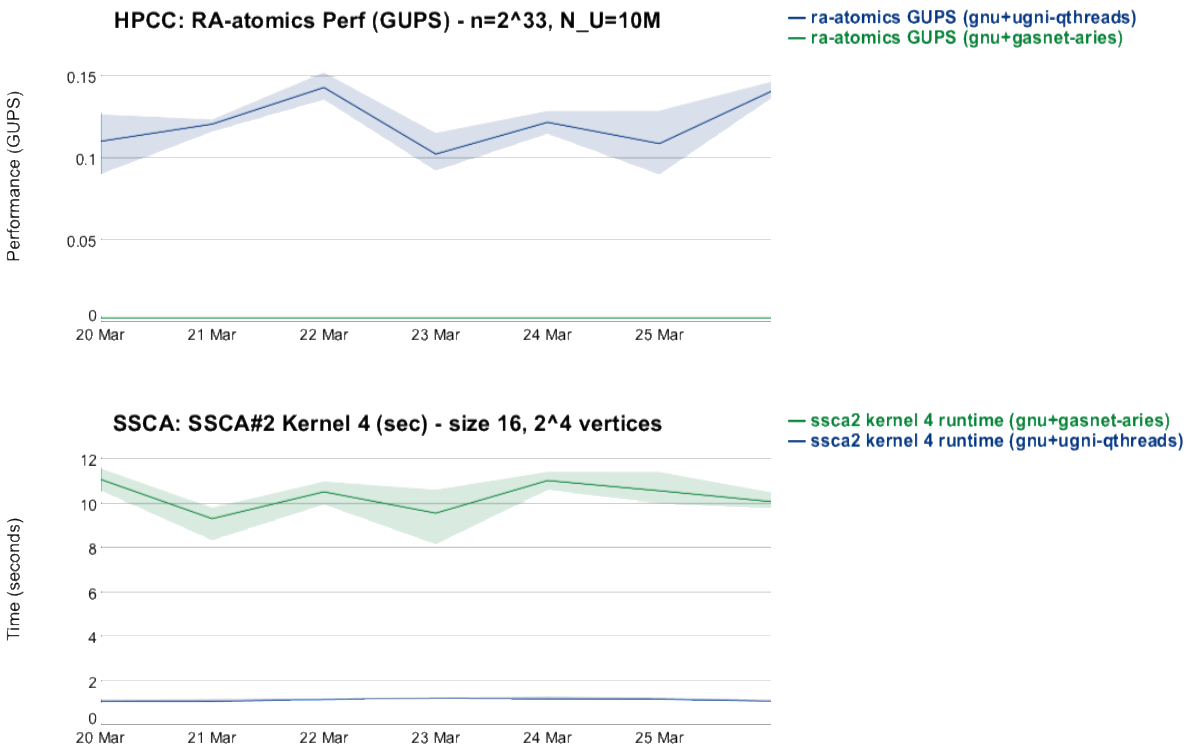
Ugni/Muxed Background

Ugni/Muxed: Introduction

- **Chapel supports Cray-specific comm and tasking layers**
  - CHPL_COMM=ugni
    - interacts with NIC via lightweight uGNI (user Generic Network Interface)
    - provides access to network atomics (CHPL_NETWORK_ATOMICS=ugni)
    - supports a high degree of communication concurrency
  - CHPL_TASKS=muxed
    - implements task-switching in user-space
- **Tuned for fine-grain, latency-bound codes like SSCA/RA**
- **Historically, only available when using pre-built module**
  - closed-source/proprietary for IP reasons
    - patent for IP was granted last year

Ugni: Background

- **Ugni offers significant performance advantages**
  - outperforms gasnet-aries in all studied applications
    - usually significantly, particularly for codes it was originally tuned for



Muxed: Background

- **Muxed used to offer significant performance advantages**
  - user-space task-switching allowed it to vastly outperform fifo
- **Nowadays qthreads typically outperforms muxed**
  - qthreads (our default) is highly optimized
    - also does user-space task switching
  - designed as a general tasking solution
    - whereas muxed was specifically designed/tuned for SSCA
  - qthreads is also numa-aware and has built-in full/empty support
- **Few notable cases where muxed still performed better**
  - as of 1.14: FFT, HPL, and MiniMD

Ugni/Muxed: Background

- **Have wanted to open-source ugni**
  - users building from source will receive performance benefits
- **Have also wanted to retire muxed**
  - qthreads generally performs much better
  - need to improve qthreads for a few benchmarks before retiring muxed
- **Combined, these efforts will simplify development**
  - eliminate development/maintenance cost for muxed
  - ugni development will use public repo, public issue tracker, etc.

Communication Improvements

Open Source Ugni

Open Source Ugni: This Effort and Impact

This Effort: Open-sourced ugni communication layer

- now included in public repository
- ugni is now our default on Cray machines
  - historically, it was only the default with the pre-built module
- also compiles with PGI now
  - mostly for uniformity and ease of documentation

Impact: Easier development, perf benefit for open-source users

- have already seen benefits of ugni being open-sourced
  - open-source developer was able to test ugni-specific bug-fix
  - opened several GitHub issues to track ugni improvements
- additionally, publicizing ugni motivated us to revisit/review the code
  - led to removal of a performance-limiting memory registration limit

Ugni Registration Limit

Ugni Registration Limit: Background

- **Ugni registers the heap with multiple comm domains**
  - access to individual comm domains is serialized
  - having multiple comm domains improves comm concurrency
    - dramatically improves performance of latency-bound codes (SSCA, RA)
  - ideally, want at least one comm domain per core

- **Early "native" slurm couldn't get exclusive access to NIC**
  - this limited the total amount of memory that could be registered
    - total memory is heapSize * numCommDomains

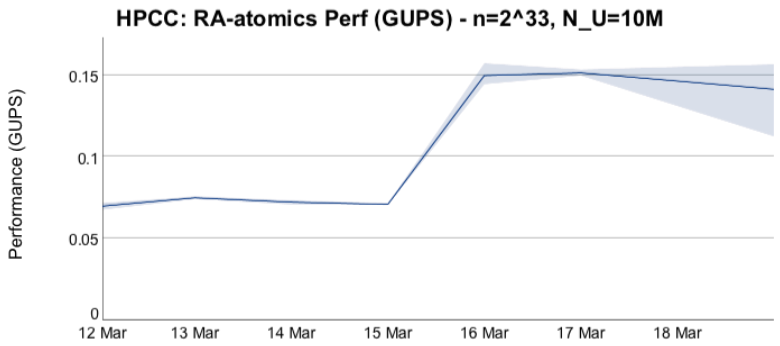
- **Slurm limitation constrained number of comm domains**
  - in practice, could only register up to 15 comm domains
    - ~1/2 the number cores on a modern Xeon-based XC

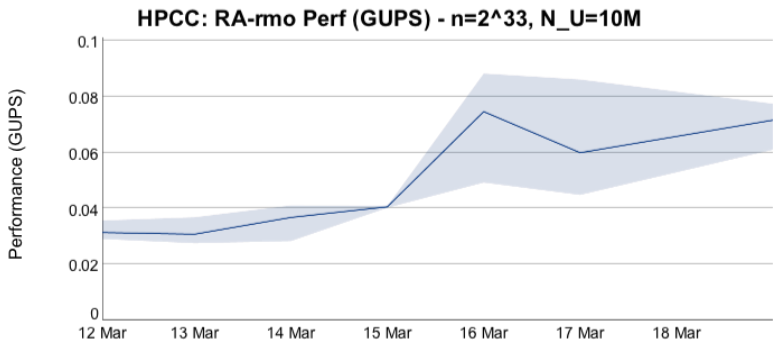
Ugni Registration Limit: This Effort

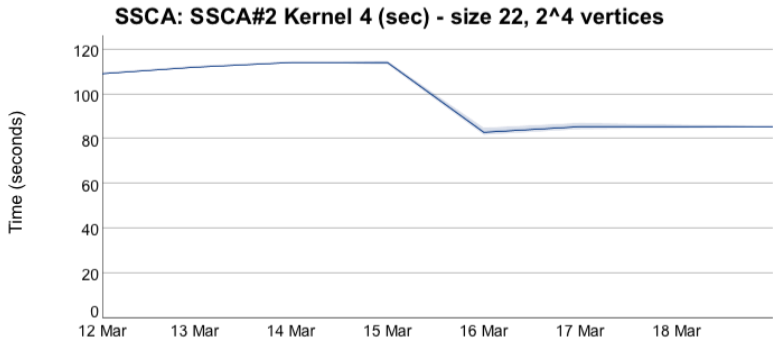
- **Revisited the need for registration limit**
  - learned that slurm can now get exclusive access to the NIC
    - i.e. registration limitation no longer exists
- **Removed comm-domain-limiting code from ugni**
  - limit now uniform between slurm and pbs/aprun systems
- **Comm domain limit is based on the number of cores**
  - currently capped at max of 30 comm domains
    - based on gemini limit; aries supports up to 127
    - doesn't hurt us on current Xeon-based systems
      (but need to revisit for Xeon-Phi and future Xeon systems)

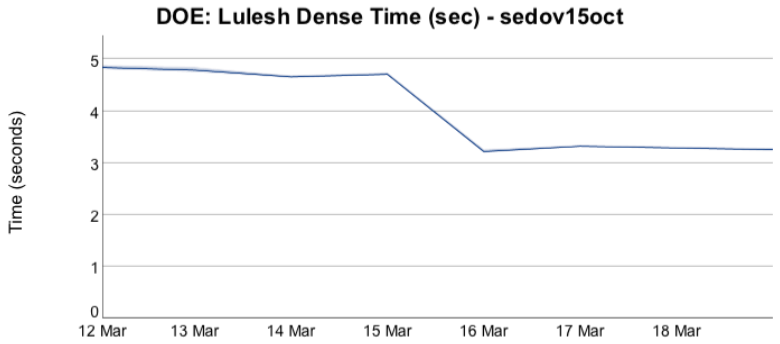
Ugni Registration Limit: Impact

- **Resulted in significant performance improvements**
  - for latency-bound applications run on slurm-managed machines









Ugni Registration Limit: Impact and Next Steps

Impact:

- led to some significant performance improvements

Next Steps:

- continue to improve ugni performance
  - evaluate performance of increasing comm domain limit for aries
- investigate poor gasnet-aries performance
  - evaluate GASNet's --enable-gni-multi-domain support

Tasking Improvements

Register Qthreads Task Stacks

Registered Task Stacks: Background

- **On Crays, only registered memory can be communicated**
  - unregistered memory has to be copied into a registered segment first
  - for best performance, ugni registers the entire Chapel heap
    - memory in the heap can be directly communicated (no copying needed)
- **Muxed tasking uses Chapel's allocator**
  - means task-stacks are part of the registered heap
- **Historically, qthreads used the system allocator**
  - task-stacks were not part of the registered heap (had to be copied)
- **Extra copying to/from qthreads stacks hurt performance**
  - identified as reason muxed was beating qthreads for FFT and HPL

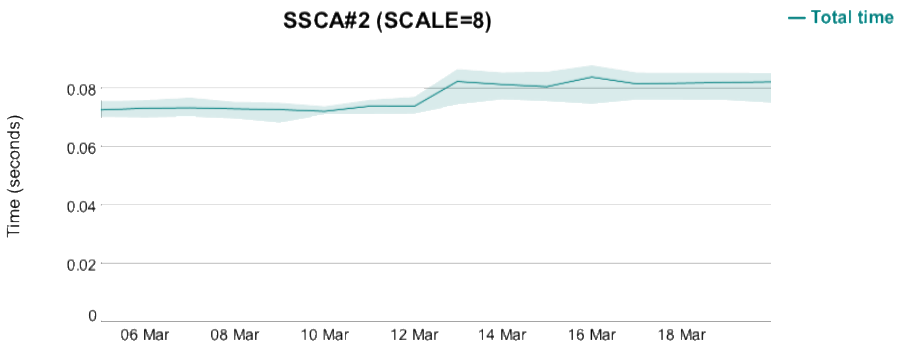
Registered Task Stacks: This Effort

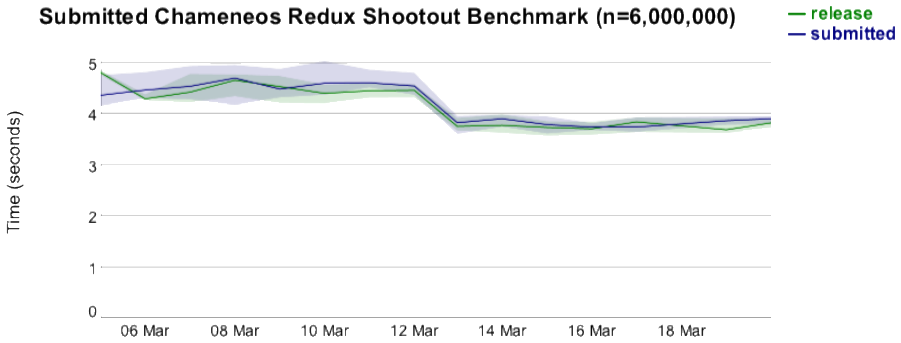
- **Make qthreads use Chapel's allocator**
  - task stacks are now in registered heap (as is all qthreads memory)

- **Worked with qthreads team to add *external allocators***
  - qthreads now has shim layers for qt_malloc, qt_free, etc.
    - default forwards to system allocator (old behavior)
    - new one forwards to Chapel's allocator
  - included in qthreads 1.12 release

Registered Task Stacks: Impact

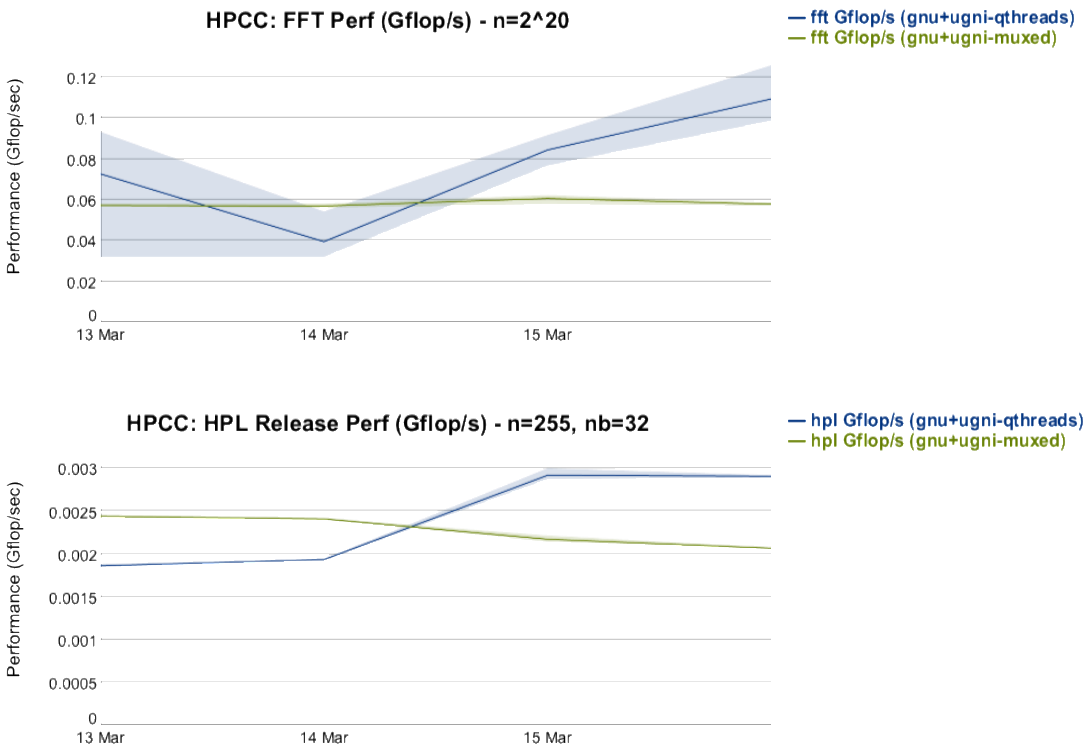
- **No real single-locale performance changes**
  - minor changes (both positive and negative)
    - really just “noise” caused by memory layout differences between allocators





Registered Task Stacks: Impact

- **Improved multi-locale performance**
  - qthreads now outperforms muxed for FFT and HPL



Limit Qthreads Memory Pool

Limit Qthreads Memory Pool: Background

Background: Qthreads aggressively pools memory

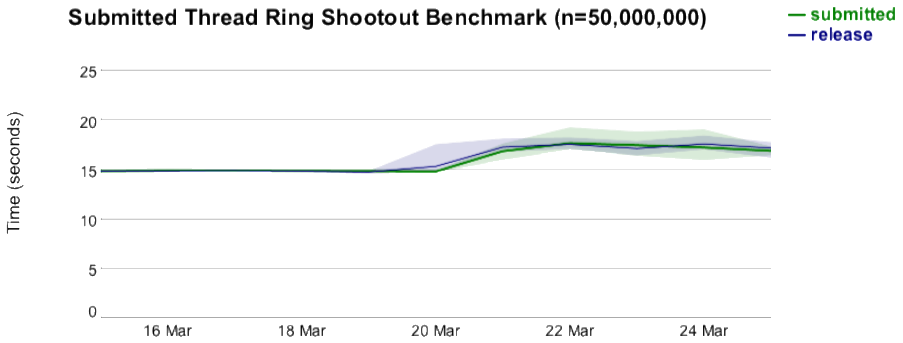
- by default, qthreads allocates space for ~128 items per pool
  - qthreads assumes ~4KB stack size, but our default is 8MB
- Chapel sets limit on qthreads max pool size
  - previously something like: (2 * numCores * stackSize)
  - based on belief that there was a single pool for task stacks
- discovered that there is actually a pool per worker (core)
  - on 68-core KNL, resulted in trying to allocate >30 GB in task-stacks
  - only noticed once qthreads started using our allocator (limited heap size for some apps led to OOM)

This Effort: Further limit qthread pool size

- now just a multiple of the stack size
  - effectively just an upper bound of 65MB (8 default-sized stacks)
  - note that this only affects compute nodes with more than 4 cores

Qthreads Memory Pool Limit: Impact

- **Applications now use less memory**
  - resolved KNL OOMs
- **Minor thread-ring regression for high core-count nodes**
  - thread-ring creates over 500 concurrent tasks
    - new pool limit results in creating more pools



Deprecate Muxed

Deprecate Muxed

Background: No remaining cases where muxed beats qthreads

- array-views work resolved MiniMD difference
- registered task stacks resolved FFT and HPL differences
- task-spawning optimizations further improved qthreads performance

This Effort: Deprecated muxed tasking

- officially deprecated for 1.15 release

Next Steps: Remove muxed source code and documentation

- support will be removed for the 1.16 release

Other Runtime Improvements

Other Runtime Improvements

- **Fixed dynamic linking for gasnet-aries on Cray systems**
- **Fixed support for gasnet+muxed without hugepages**
- **Switched qthreads initializer to run in detached state**
  - contributed by Rob Upcraft
- **Fixed massivethreads for stack-allocated arg-bundles**
  - contributed by Kenjiro Taura

Other Third-Party Improvements

Other Third-Party Improvements

- **Upgraded jemalloc to version 4.5.0**
  - no major performance impact
  - now using a vanilla jemalloc (all our patches accepted upstream)
- **Upgraded Qthreads to version 1.12**
  - added support for external-allocators
  - fixed sleep-interception bug
  - added hybrid spin/condwait for nemesis
- **Upgraded hwloc to version 1.11.6**
  - fixed bug that KNL work encountered
  - improved startup time on high core-count machines by ~25%
- **Upgraded GASNet to version 1.28.0**
- **Made GASNet build amudprun launcher for host machine**

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

THE SUPERCOMPUTER COMPANY