Locale Model 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.

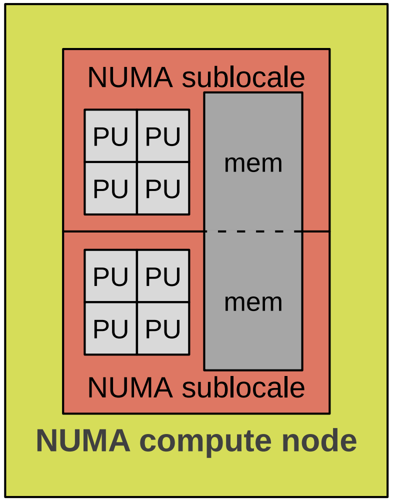
Outline

- NUMA Locale Model Improvements
- Chapel on KNL

NUMA Locale Model Improvements

NUMA: Background

- **‘numa’ locale model describes NUMA compute nodes**
  - NUMA: Non-Uniform Memory Access
  - a NUMA node has 2 or more *sublocales**
    - represented as child locales of network-level locales
    - each has processors and memory
  - access to own-sublocale memory is faster than access to other-sublocale memory



- **Performance with ‘numa’ locale model was poor**
  - DefaultRectangular arrays were not fully optimized for sublocales
  - parallel iterator placed tasks on sublocales as it wished
  - array data was not always similarly placed
  - thus: unreliable task/data affinity

* = these are typically called numa *domains*, but we’re avoiding that term here to avoid confusion with Chapel’s domains

NUMA: Background (DefRect Domain Iteration)

- **How DefaultRectangular domain parallel iteration works**
  - block one dimension of the domain, creating #sublocales subdomains

```
forall i in {1..2, 1..8, 1..N} { … }
```

example:
{1..2, 1..8, 1..N}

	1	2	3	4	5	6	7	8
1	1..N	1..N	1..N	1..N	1..N	1..N	1..N	1..N
2	1..N	1..N	1..N	1..N	1..N	1..N	1..N	1..N

NUMA: Background (DefRect Domain Iteration)

- **How DefaultRectangular domain parallel iteration works**
  - block one dimension of the domain, creating #sublocales subdomains

```
forall i in {1..2, 1..8, 1..N} { … }
```

example:
{1..2, 1..8, 1..N}

on 4 sublocales:

	1	2	3	4	5	6	7	8
1	1..N	1..N	1..N	1..N	1..N	1..N	1..N	1..N
2	1..N	1..N	1..N	1..N	1..N	1..N	1..N	1..N

rewritten

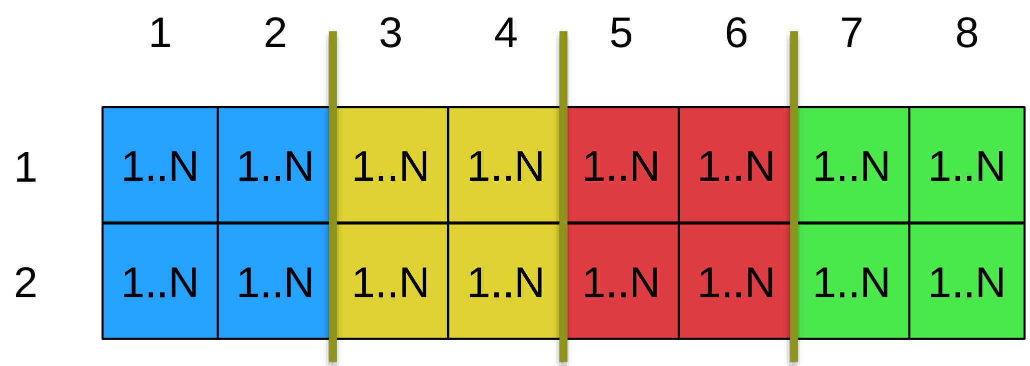
```
coforall subloc in 0..#here.getChildCount() {   // 0..3
  local on here.getChild(subloc) {   // 1 task runs on each sublocale
    forall i in {1..2, subloc*2+1..#2, 1..N} { … }
  }
}
```

NUMA: Background (DefRect Array Locality)

- **Goal: array memory locality matches index partitioning**
  - locality was set implicitly, via first-touch

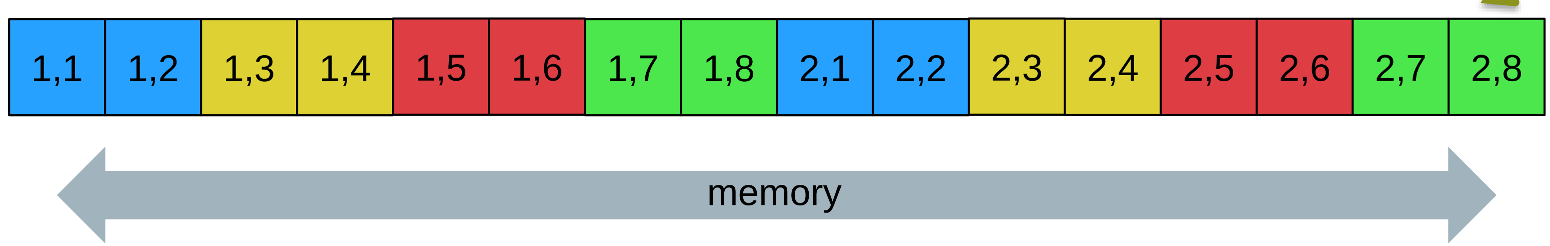
example:
{1..2, 1..8, 1..N}

on 4 sublocales:



memory locality follows domain partitioning

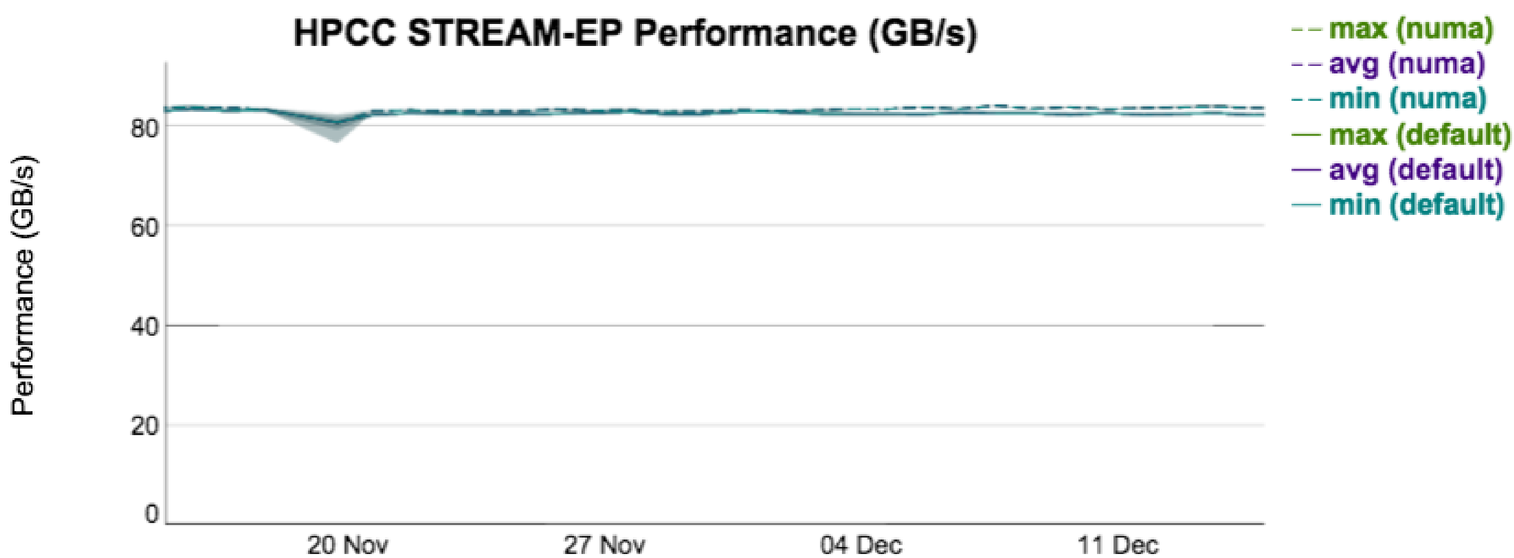
1..N array
elements
in each
box



NUMA: Background (DefRect Array Locality)

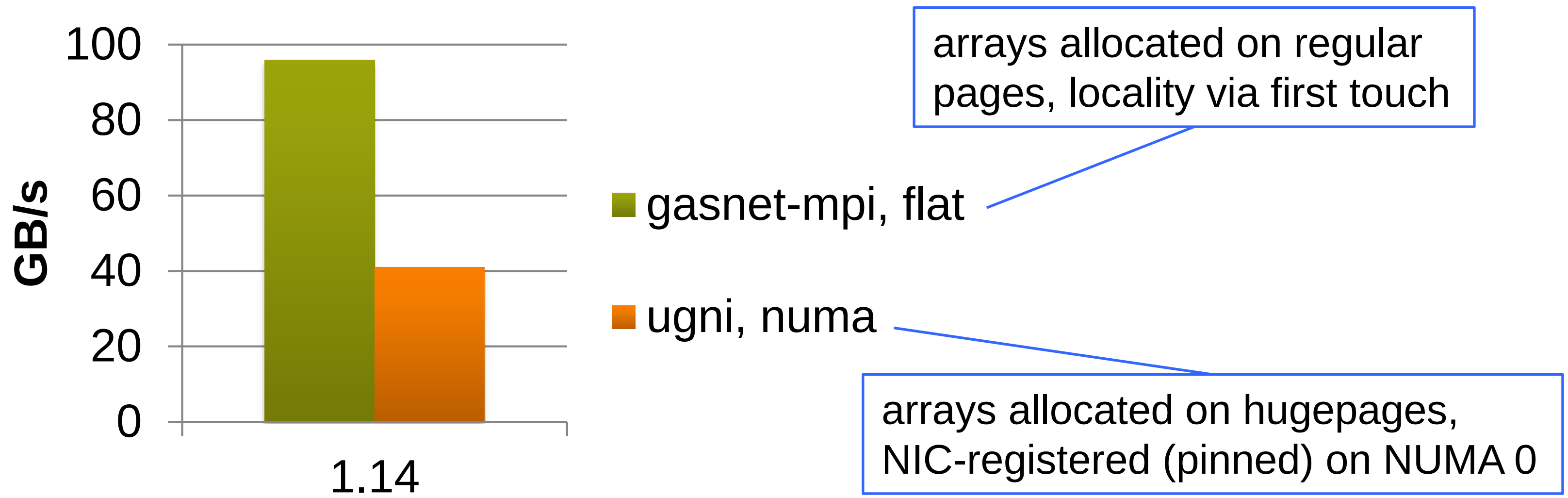
- **In many cases this technique resulted in good affinity**
  - unsurprising, since this is effectively what OpenMP does for NUMA

stream-ep on single-locale linux64:



NUMA: Background (DefRect Array Locality)

- **But pre-existing locality can thwart first-touch**
  - stream-ep on multi-locale Cray XC:

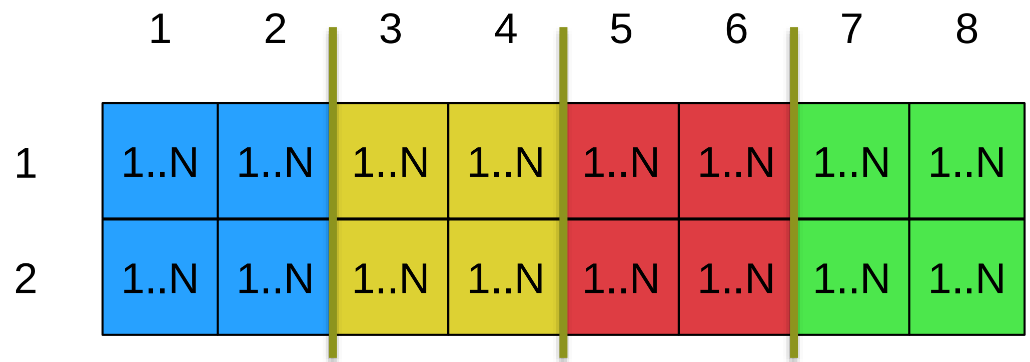


  - NIC registration of comm=ugni heap pins mem, setting NUMA locality
    - changing NUMA locality would mean re-registration, broadcast, etc.
  - any reused memory will also have pre-existing locality
    - changing NUMA locality will be expensive due to page migration

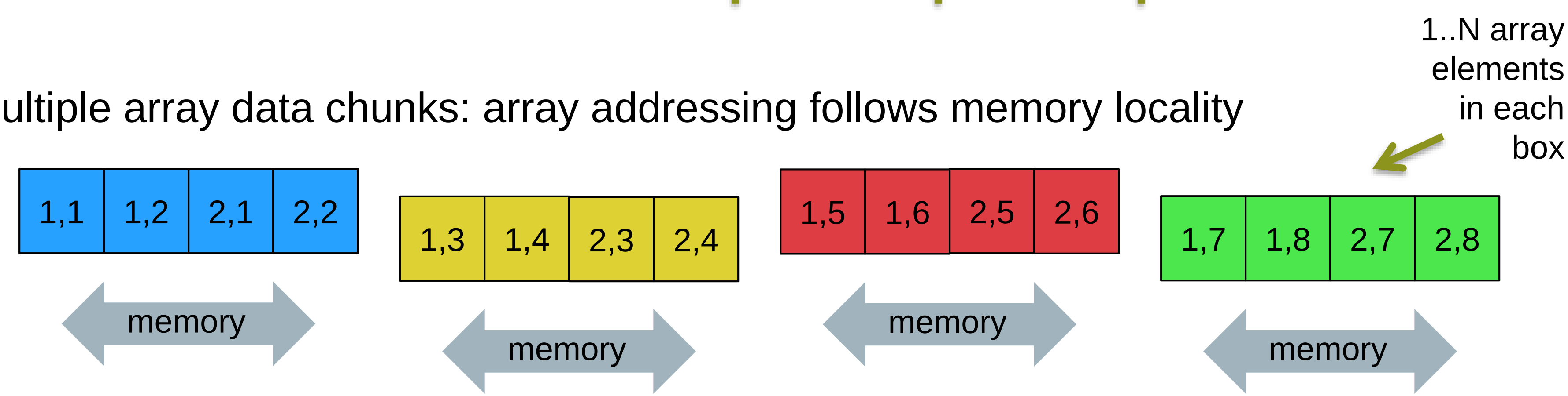
NUMA: This Effort

- **Split array data just like DefRect iterator splits domain**
- **Set NUMA locality of each array data chunk separately**

example:
{1..2, 1..8, 1..N}
on 4 sublocales



multiple array data chunks: array addressing follows memory locality



NUMA: This Effort (Multi-chunk Array Alloc)

- **Multi-chunk array allocation is also known as *multi-ddata***
- **Partition array data as DefRect iterator partitions domain**
  - Currently only done for arrays >= 1 MiB
- **Reworked array implementation extensively**
  - rewrote array addressing and iterators
  - updated other code that assumed a single array data block

NUMA: This Effort (NUMA-Aware Mem Alloc)

- **In some cases, just force locality after allocation**
  - configurations: single-locale, any memory layer
  - only for array data allocation
- **In other cases, do real NUMA-aware allocation**
  - configuration: Cray XC, comm=ugni, registered heap, mem=jemalloc
  - affects all memory allocation, not just array data
  - implemented using new comm/mem layer cooperation:

comm:	allocate heap space on hugepages
mem:	logically partition heap into #sublocs blocks; set each block's locality to corresponding NUMA sublocale
comm:	register heap with the NIC
mem:	manage each block as a separate heap: allocation by task on *subloc* always comes from *subloc*'s heap

NUMA: Impact

- **Performance depends on style of array access**

- **Array iteration, implicit element access:**

```
forall (a,b,c) in zip(A,B,C) do a = b + alpha * c;
```

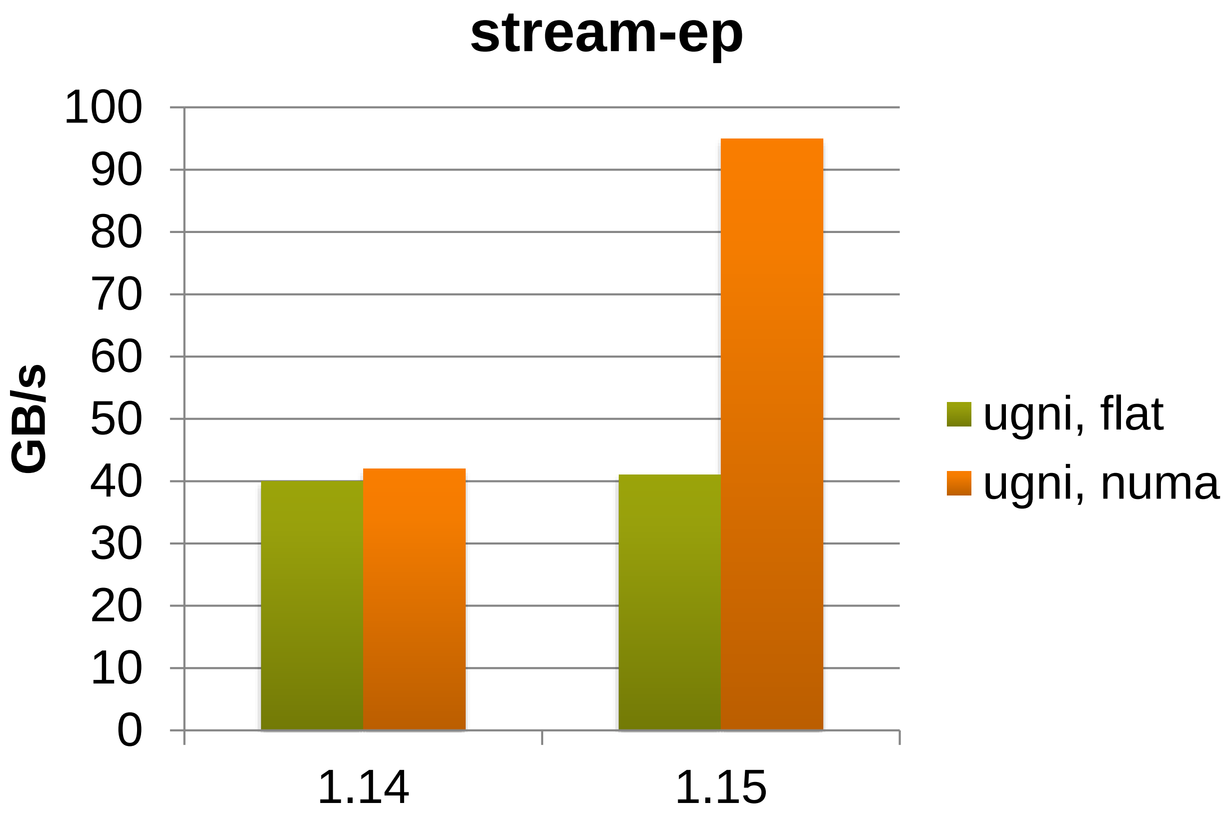
  - single-node: performance largely unchanged
    - locality was good based on first-touch, though now is more principled
    - a memory reuse test case should show improvement, but is difficult to code
  - multi-node with registered heap: performance better, often much better

- **Domain/range iteration, explicit element access:**

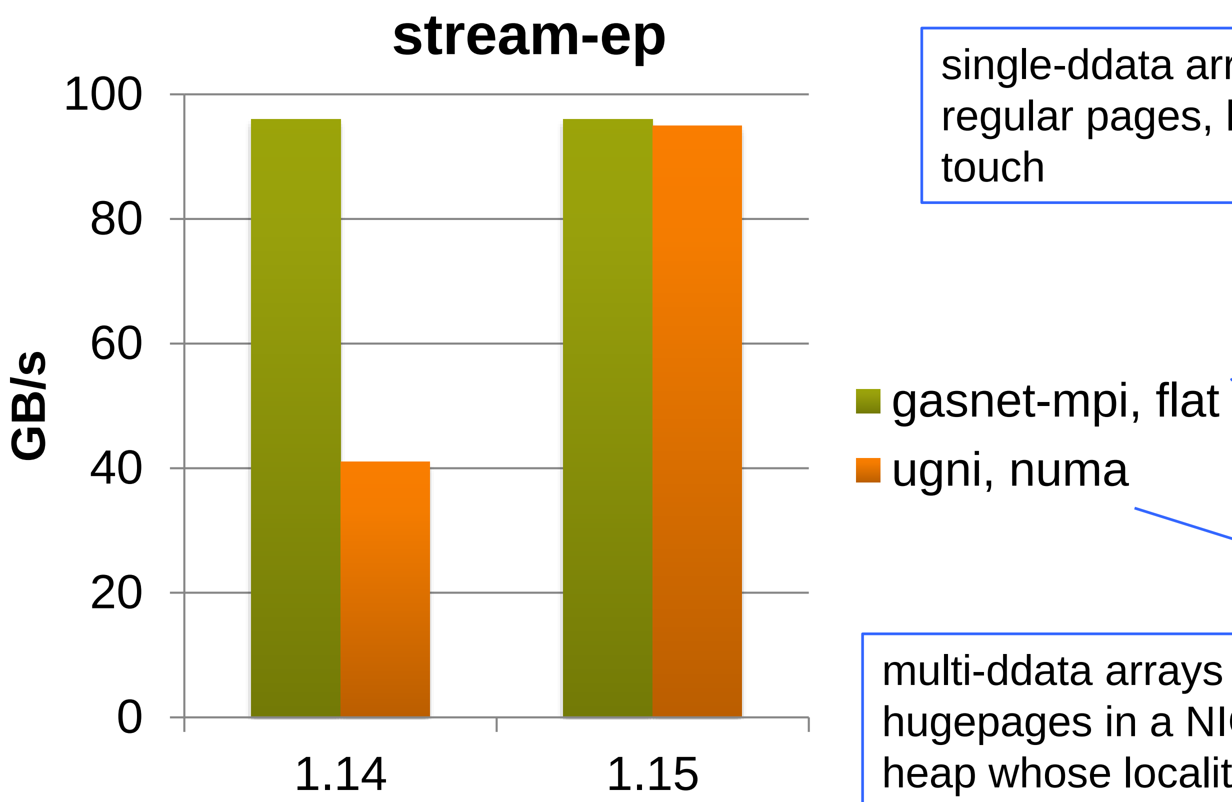
```
forall i in A.domain do A[i] = B[i] + alpha * C[i];
```

  - performance worse, often much worse

NUMA: Impact (2-node Cray XC w/ ugni)



NUMA: Impact (2-node Cray XC ugni vs. gasnet)

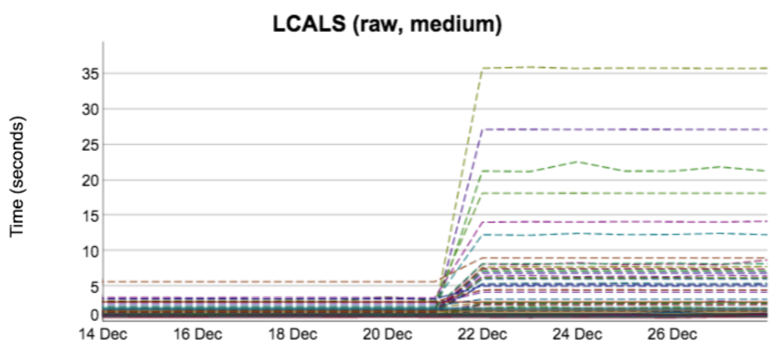


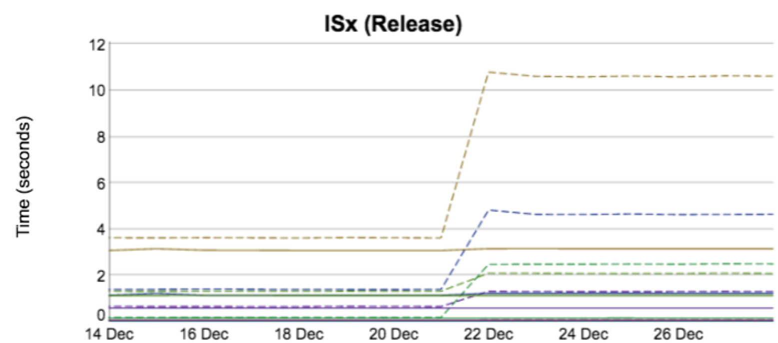
single-ddata arrays allocated on regular pages, locality via first touch

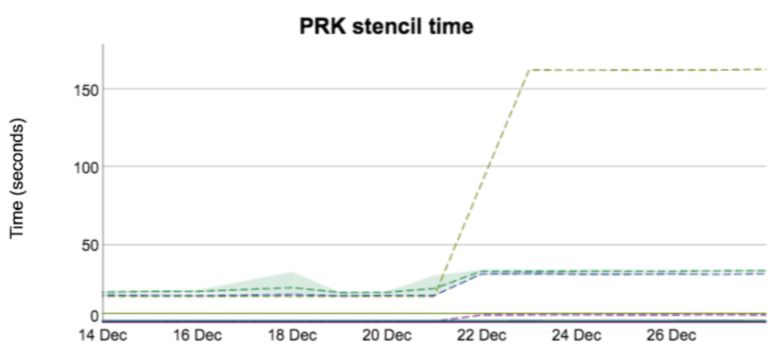
multi-ddata arrays allocated on hugepages in a NIC-registered heap whose locality is blocked across NUMA sublocales

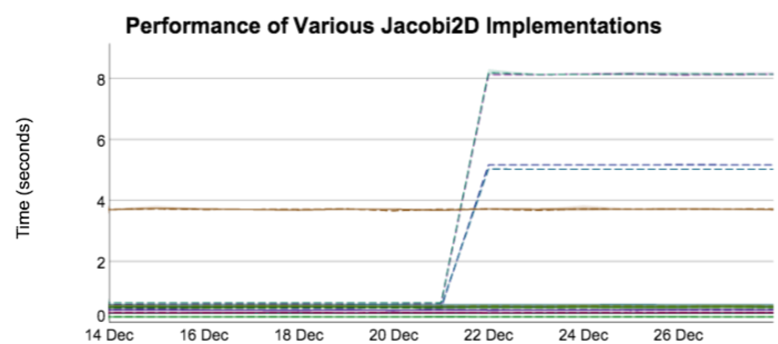
NUMA: Impact (Performance Regressions)

- **Big locModel=numa performance losses for many tests**
  - array access is much slower when iterating over domain or range
  - due to added overhead of index calculation for multi-chunk arrays





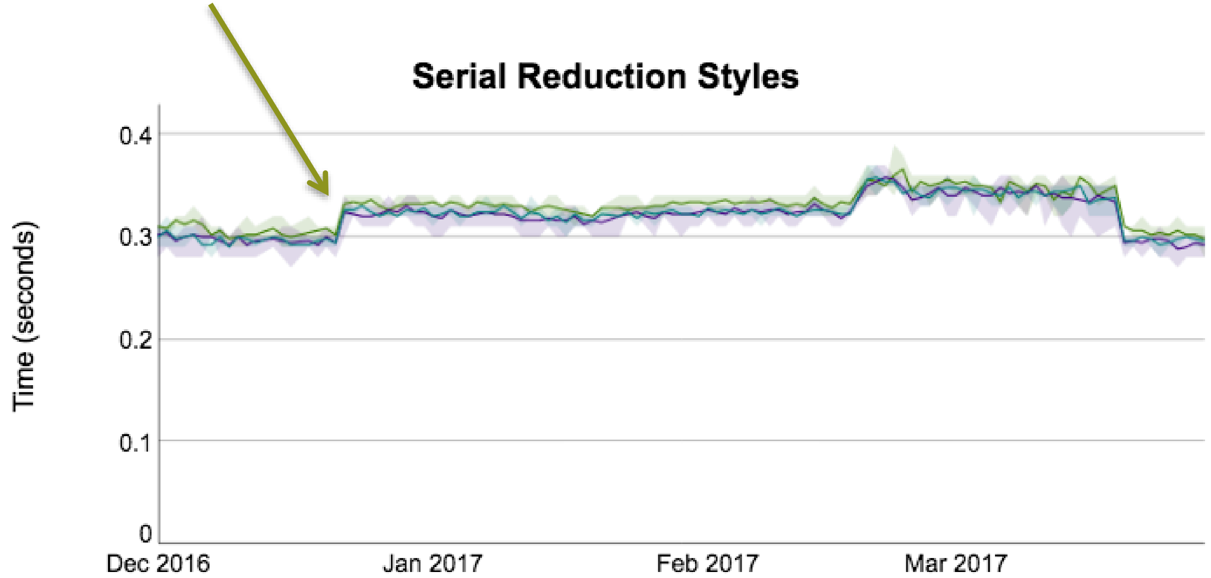




NUMA: Impact ('flat' Perf Regressions)

- **New array metadata fields even impacted locModel=flat**
  - they weren't used, but initializing them wasn't free
  - and mere presence changed cache line sharing in array descriptor

added metadata fields in array
descriptor reduce performance

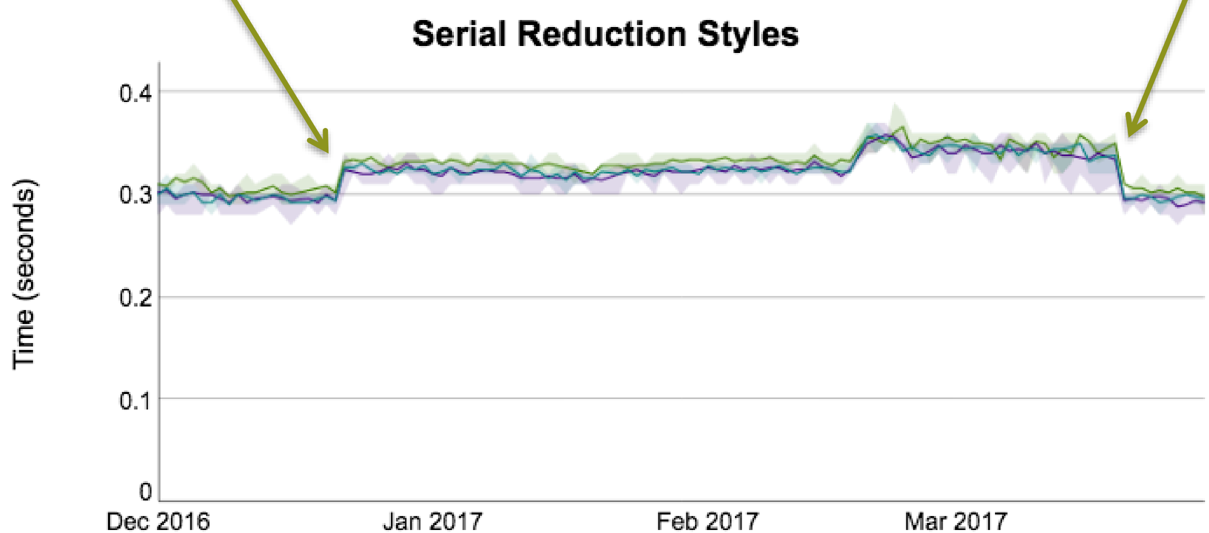


NUMA: Impact ('flat' Perf Regressions)

- **New array metadata fields even impacted locModel=flat**
  - they weren't used, but initializing them wasn't free
  - and mere presence changed cache line sharing in array descriptor

added metadata fields in array descriptor reduce performance

new 'void' type allows effectively removing numa-oriented array descriptor fields for locModel=flat



NUMA: Next Steps

- **Major 1.16 decision: stay with multi-ddata?**
- **If retaining multi-ddata:**
  - reduce array access overhead; how much can we achieve?
  - provide for programmer specification re: single- vs. multi-ddata
- **If reverting to single-ddata:**
  - how (whether) to handle pre-existing locality?
  - how to handle multi-node NIC-registered heaps?
- **Improve NUMA-aware memory allocation**
  - do more sublocale-aware allocation (task stacks, etc.)
  - reduce migration by allocating memory with proper locality

Chapel on Intel Xeon Phi “Knights Landing” (KNL)

Chapel on KNL: Background (KNL)

- **KNL is a many-core platform (60+ cores)**

- **Cores can access two kinds of memory**
  - external memory (DDR)
  - on-package high-bandwidth multichannel DRAM (MCDRAM)

- **Different from other processors supported by Chapel**
  - helps us think forward to more complex emerging architectures
  - leads us to exercise and expand NUMA support
  - will also benefit from future vectorization efforts

Chapel on KNL: Background (KNL Configs)

- **KNL can be used in several different configurations**
  - cluster modes
    - cores can appear as grouped into one, two, or four NUMA nodes
  - memory modes
    - MCDRAM used as memory
    - MCDRAM used as direct-mapped level-3 cache
    - a combination of the two
  - configuration is controlled by BIOS
    - a change requires ~15 minutes to reboot the affected processor

- **MCDRAM is seen as core-less NUMA nodes**
  - when accessed via the Linux NUMA interface

Chapel on KNL: This Effort

- **Create 'knl' locale model**
  - NUMA-oriented, based on 'numa' locale model with modifications
  - benefits from the multi-ddata improvements

- **Provide a mechanism to use MCDRAM**
  - in a portable way

- **Use a newer release of hwloc**
  - required updates to handle core-less NUMA nodes (MCDRAM)

Chapel on KNL: This Effort

- **New methods on 'locale'**

```
locale.highBandwidthMemory()  // MCDRAM if 'locale' is KNL in mem. mode
locale.lowLatencyMemory()     // DDR on KNL
locale.largeMemory()          // DDR on KNL
locale.defaultMemory()        // DDR on KNL
```

  - implemented across all locale models for portability
  - use the Intel memkind library when CHPL_LOCALE_MODEL=knl
  - yield regular memory, otherwise

- **A basic structure for accessing different kinds of memory**
  - long term, possibly could be used by domain map authors

Chapel on KNL: Impact

- **Chapel programs can target KNL's MCDRAM**

```
on here.highBandwidthMemory() {
  x = new myClass();        // placed in MCDRAM
  ...
  on here.defaultMemory() {
    y = new myClass();      // placed in DDR
    ...
  }
}

on y.locale.highBandwidthMemory() {
  z = new myClass();        // same locale as y, but using MCDRAM
  ...
}
```

Chapel on KNL: Impact

- **Programs work whether MCDRAM is available or not**
  - uses default memory if MCDRAM is unavailable

- **Each locale figures this out for itself**
  - can use heterogeneous configuration of KNL processors
  - avoids the need to reboot if configuration choice is not critical

Chapel on KNL: Status and Next Steps

Status:

- 'knl' locale model and locale methods are available in Chapel 1.15
  - included in Chapel module on Crays

Next Steps:

- Experiment with using MCDRAM in various benchmarks
- Work on vectorization improvements/optimizations
  - also explore potential KNL-specific optimizations
- Enhance the interface to specify multiple conditions
  - "Give me high bandwidth memory, but only if at least 1GB exists"
- Architecture queries
  - "Is high bandwidth memory available on this locale?"

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