# OVERVIEW OF HSTREAMS :

## A STREAMING LIBRARY FOR HETEROGENEOUS PLATFORMS

February, 2017

Hetero Streams Library 1.2

# OUTLINE

- **Goals and objectives**
- **Design and usage**
- **Performance**
- **Interfaces**
- **Outlook**
- **Backup**
  - Terminology
  - Dependence cases

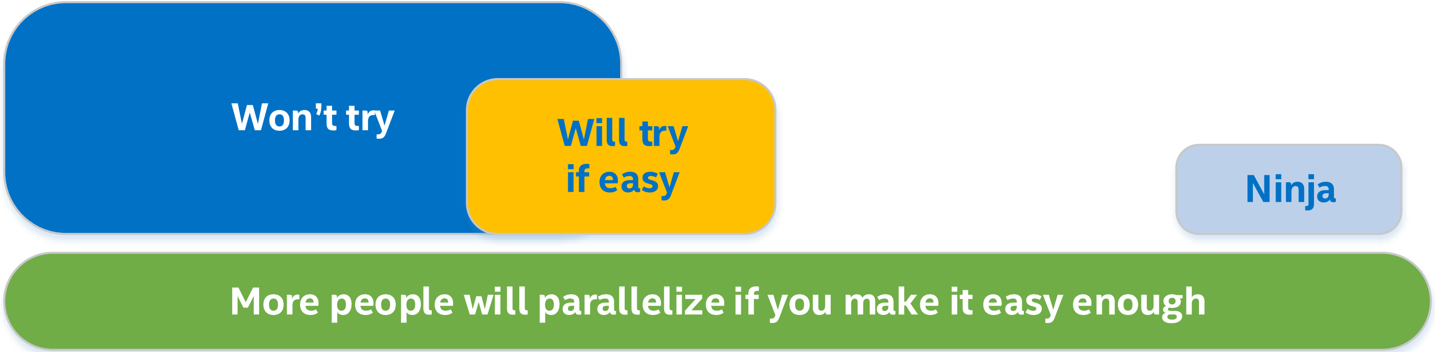
# Related documents

- **hStreams_Release_Notes.pdf**
  - Release notes
- **hStreams_Tutorial.pdf**
  - Description of the concepts, APIs, reference codes, operation and usage
- **hStreams_Reference.pdf**
  - Programming Guide and API Reference
- **hStreams_Porting_Guide.pdf**
  - How to port from Hetero Streams Library 1.1 to 1.2

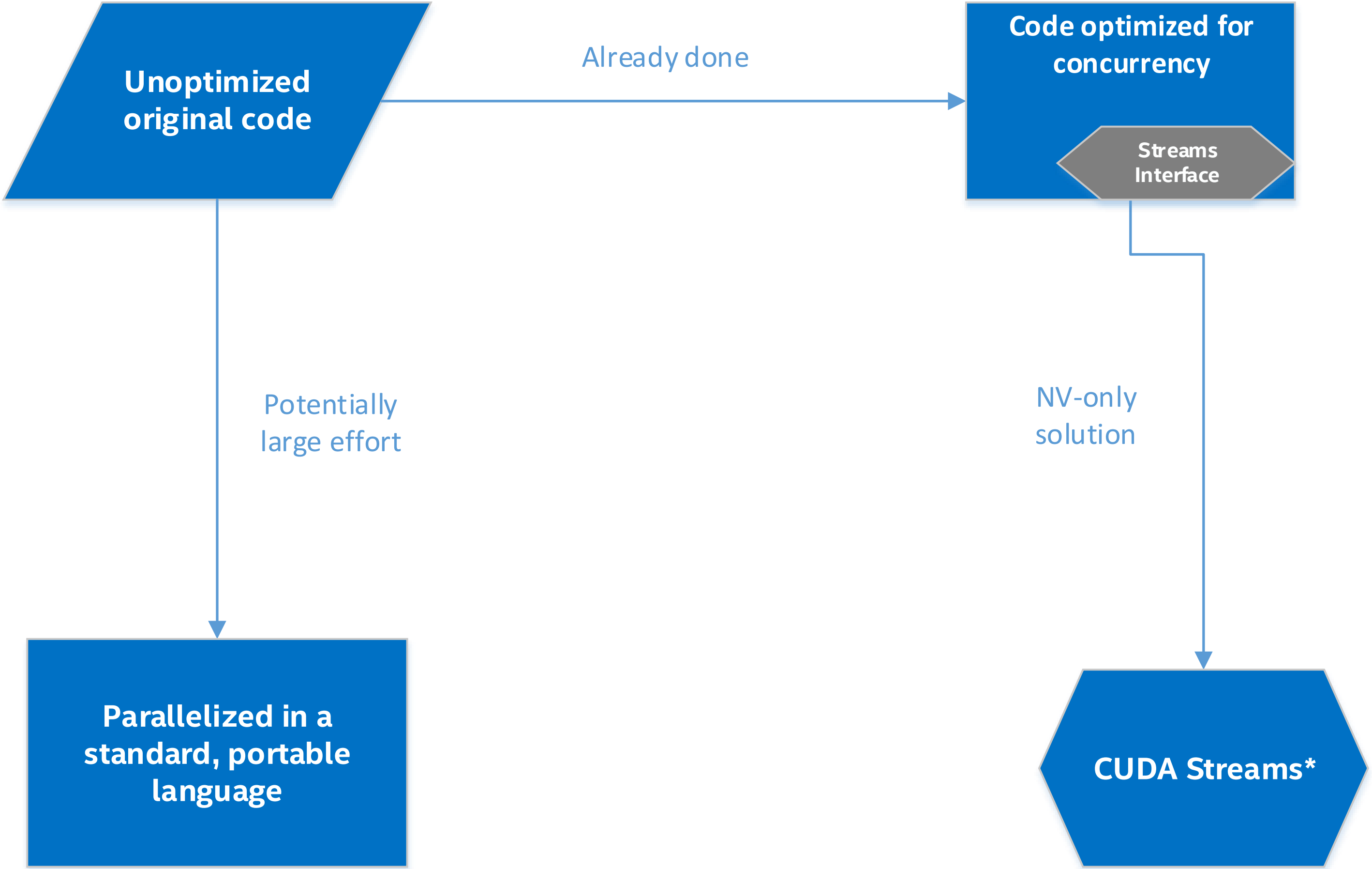
# GOALS AND OBJECTIVES

# Technical gaps being filled.

- **Concurrency**
  - Between host and cards
  - Multiple tasks within a card (domain)
  - Overlap computation and communication
- **Ease of use**
  - “Don’t force me to over-specify controls” - easy defaults
  - “Make the abstractions really simple” - FIFO stream

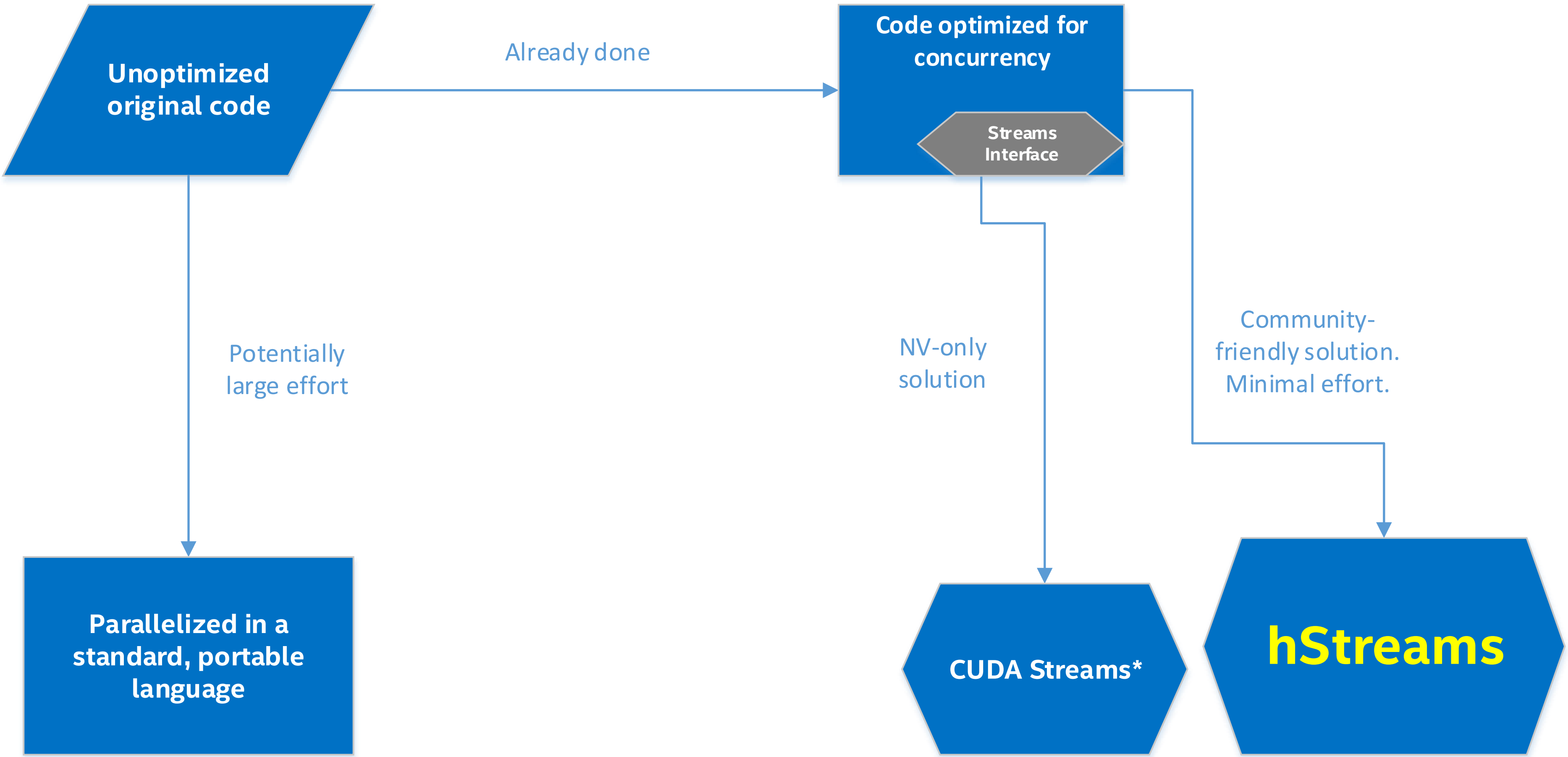


# Fitting the current reality


Unoptimized original code
Already done
Code optimized for concurrency
Streams Interface
Potentially large effort
NV-only solution
Parallelized in a standard, portable language
CUDA Streams*

- **From scratch? Intel® recommends MPI + OpenMP 4+**
- **Sometimes code is only enabled for streams**
- **Some names and brands may be claimed as the property of others*

# Fitting the current reality


Unoptimized original code
Already done
Code optimized for concurrency
Streams Interface
Potentially large effort
NV-only solution
Community-friendly solution. Minimal effort.
Parallelized in a standard, portable language
CUDA Streams*
hStreams

- **This is our compatibility offering**
- **Some names and brands may be claimed as the property of others*

# Goals and objectives

- **Abstraction of memory, data transfers, invocation**
  - Heterogeneous: local or remote or a mix of CPU, accelerators, …
- **Library, not a language or compiler feature**
  - Because many customers do not keep up with our compilers
  - Fortran is important, use C API
  - There is a similar feature available in the Intel® compiler
- **Plumbing vs. a new programming model**
  - Not necessarily suggesting that people rewrite code in this
  - Continue to make use of OpenMP, TBB, MPI, etc.
  - Users provide their own functionality
- **Offers simple interface, opt-in control**
  - Minimum work required is low, available control is high

# Making concurrency easier

- **Compatible enabling path**
  - Similar to CUDA* Streams, OpenCL*
  - Users can provide their own functionality
- **Simplicity**
  - FIFO abstraction is about as simple as it gets
  - Map FIFOs to streams
- **Abstract heterogeneous resources**
  - Map work onto streams (OpenMP places), let it expand to fill them
  - Deferred binding and logical abstraction enables portability
  - Optimize away redundant actions

**Some names and brands may be claimed as the property of others.*

# Plumbing vs. a programming model

- **Programming models**
  - MPI*
    - *Alternative to streams, for static distribution, ~same work per rank*
  - OpenMP* or TBB*
    - *Users expose and manage parallelism within a “place”  (OpenMP4)*
- **Plumbing**
  - hStreams
    - *Heterogeneous – can map to any set of “places” (OpenMP4)*
    - *Streams – simple FIFO semantics, mapped onto a “place”*
    - *Abstracts memory management, distribution, remote invocation*
  - Higher layer does decomposition, partitioning, cross-stream sync

**Some names and brands may be claimed as the property of others.*

# Concurrent small tasks on partitions improve efficiency

| Parallel DGEMMs: | 1 | 2 | 3 | 4 | 5 | 6 | 10 | 12 | 15 | 20 | 30 | 60 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Core Count: | 60 | 30 | 20 | 15 | 12 | 10 | 6 | 5 | 4 | 3 | 2 | 1 |
| Size | Overall Efficiency - Intel® Xeon Phi™ Coprocessor 7120P | | | | | | | | | | | |
| 128 | 2% | 4% | 6% | 9% | 11% | 14% | 15% | 22% | 28% | 27% | 33% | 17% |
| 256 | 12% | 17% | 17% | 26% | 29% | 21% | 35% | 40% | 49% | 44% | 44% | 32% |
| 512 | 34% | 43% | 49% | 49% | 48% | 52% | 58% | 57% | 59% | 60% | 58% | 59% |
| 768 | 50% | 57% | 57% | 62% | 65% | 64% | 69% | 60% | 63% | 66% | 66% | 68% |
| 960 | 58% | 65% | 69% | 65% | 65% | 68% | 70% | 71% | 74% | 71% | 73% | 72% |
| 1024 | 54% | 61% | 62% | 61% | 64% | 63% | 69% | 67% | 70% | 70% | 71% | 70% |
| 1536 | 64% | 66% | 70% | 70% | 73% | 72% | 75% | 73% | 76% | 76% | 76% | 74% |
| 1920 | 72% | 75% | 76% | 76% | 77% | 76% | 77% | 78% | 78% | 78% | 78% | 76% |
| 3840 | 78% | 79% | 80% | 80% | 81% | 80% | 81% | 81% | 81% | 81% | 81% | NA |

***Some results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.***

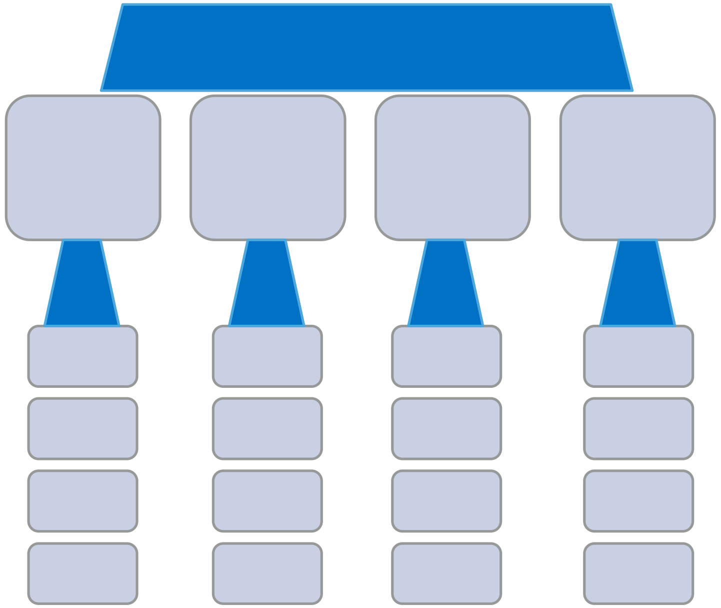
*Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Source: Intel measured or estimated as of November 2014.*

***For more information go to http://www.intel.com/performance***

# OpenMP width

- **Intel® MKL (Intel® Math Kernel Library) currently supports two degenerate cases**
  - Single thread – no OpenMP
  - All threads – OpenMP's notion of the whole machine
- **Middle ground**
  - The challenge is that OpenMP doesn't have a solid notion of multiple narrower logical machines within a wider process
    - *OMP 4.0 defines a notion of places within a wider process*
    - *But it doesn't offer good affinity control over them, and OMP_NUM_THREADS isn't managed well per place*
    - *OpenMP may be enhanced to support this better, but we'll still need a mechanism, like hStreams, to do the partitioning*
  - If we can create a framework, a hierarchy, in which we can have multiple partial-width places, that's something new

# The nested OpenMP problem



4-wide
Whole machine

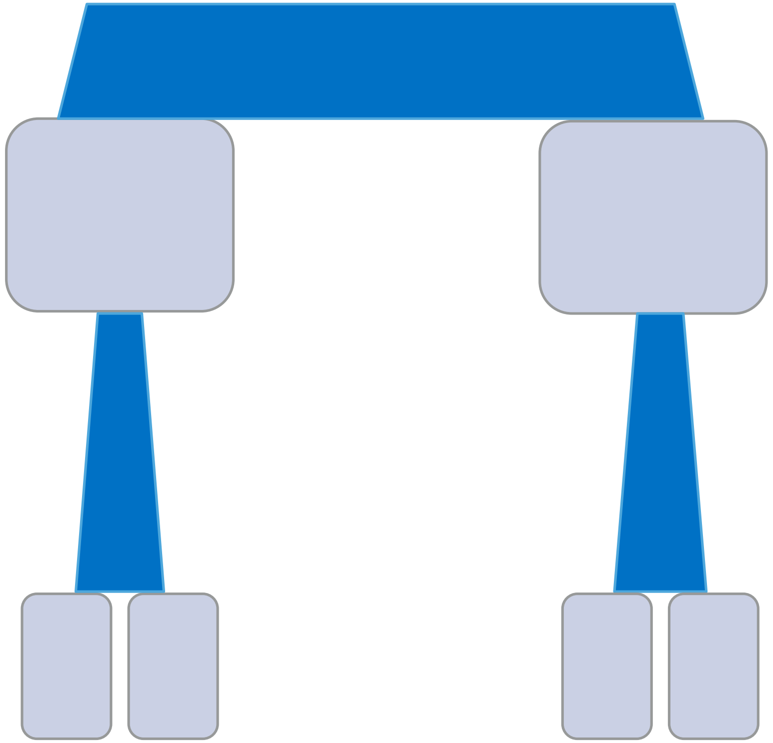
Nested
Oversubscribed

```
main() {
  omp parallel
  function()
  …
}

function() {
   omp parallel for
   …
}
```

- **Nested but context oblivious**
- **Each layer thinks it owns the world (e.g. 4 wide)**

# Avoid OpenMP nesting with a hierarchy



- **OpenMP isn't what does the nesting**
- **Hierarchy established outside of OpenMP context**

```
main() {
  establish_partition(P)
  distribute_work(P)
  sync()
}

do_work(P) {
   omp parallel for
      work(P)
}

work(P) {
  i = omp_thread_num()
  // do ith part of P
```

# hStreams Alpha (MPSS 3.5) features

- **Targets**
  - Windows* or Linux*
  - Captive MIC cards
- **Resource management**
  - One or more cards, any subset of cores on each card
    - *Avoids reserved threads, via static selection*
  - Can query what's used so far
- **Invocation**
  - General, user-provided functions
    - *uint64_t args copied by value*
    - *Heap args are matched up with COIBuffers for dependences*
    - *Up to a total of ~20 arguments, easy to add more*
    - *User explicitly specifies # of scalar & heap, creates & passes an array*
- **Buffers**
  - Per-stream FIFO dependences are enforced at buffer granularity
  - Allocation, granularity and persistence controlled by users

**Some names and brands may be claimed as the property of others.*

# hStreams Beta (MPSS 3.6) features

- **Targets**
  - Experimental feature: stream to host using core APIs (app APIs not yet supported for host)
- **Resource management**
  - Buffers may now be selectively instantiated, using hStreams_Alloc1DEx()
  - Some of the buffer properties may be controlled, e.g. pinning, aliasing, incremental addition
- **Data transfers**
  - May be among sink domains, thereby enabling card-card transfers
- **Thread safety**
  - APIs which may normally be called between init and fini are now thread safe with respect to internal state. Some examples of activities that may safely be done concurrently in different source threads:
    - *Creating buffers*
    - *Actions in different streams*
    - *Adding or removing domains or straems*
  - But users are responsible for resolving race conditions between multiple host threads, e.g.
    - *Actions in the same stream*

# You may not need hStreams if...

- **You can use the latest Intel compiler**
  - The Intel® Composer XE offload compiler supports offload streams, though it supports a subset of hStreams functionality
- **You don't need multiple streams within a host or device**
  - No concurrency: single thread or OpenMP can work fine
- **You start with an MPI + OpenMP app**
  - MPI gives you concurrent places
  - OpenMP gives you "width" within each process (not thread)
  - *However, hStreams allows for the sharing or memory across streams in the same domain, saving a copy relative to a send/recv through shmem*

# DESIGN AND USAGE

# Design and usage

- **Layering**
- **Usage overview**
- **What we use from COI**

- **See hStreams_Tutorial.pdf for**
  - Invocation example
  - Building your application
  - API descriptions

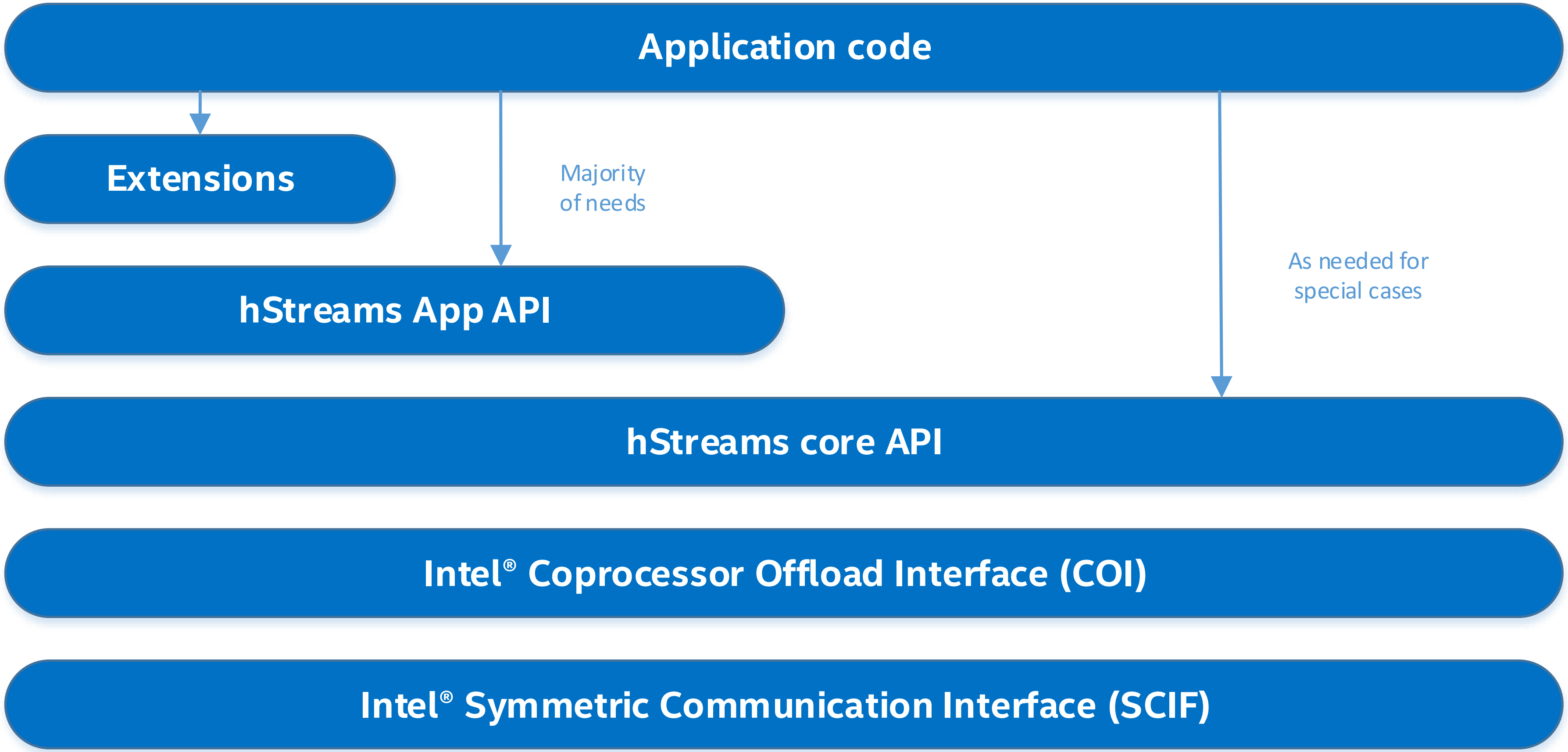
# Keep it simple but flexible

- **Most users won't need full richness of hStreams**
- **So keep things simple with an "app API" layer**
  - Self-contained for most uses, but may be mixed with core APIs
  - Fewest-possible arguments
  - Calls lower "hStreams core" layer
  - Precompiled into packaged hStreams library for ease of use
- **But flexible**
  - App API source code is available
  - Can copy it, rename functions, specialize, recompile
  - Build your own library, with incremental changes
  - Completely encapsulates lower layers
- **For more detail on APIs, see**
  - hStreams_Usage.pdf – tutorial overview of APIs and reference code
  - hStreams_Reference.pdf – Programming Guide and API Reference

# Layering

- **Logical streams**
  - Abstraction belonging to user API, and at hStreams app API
  - As many as you want
  - Mapped to hStreams core by a shim layer
- **Physical streams**
  - Explicitly mapped to one domain and an arbitrary subset of threads
  - FIFO semantics with both logical streams and physical streams
  - Can offload compute and offload movement of data
  - Associated with a COIPipeline in the implementation
  - All physical streams mapped to same domain (card) can share data
- **Completely encapsulated underlying implementation**
  - COI – Intel® Coprocessor Offload Interface
  - SCIF – Intel® Symmetric Communications InterFace

# Layering



Application code

Extensions

Majority of needs

hStreams App API

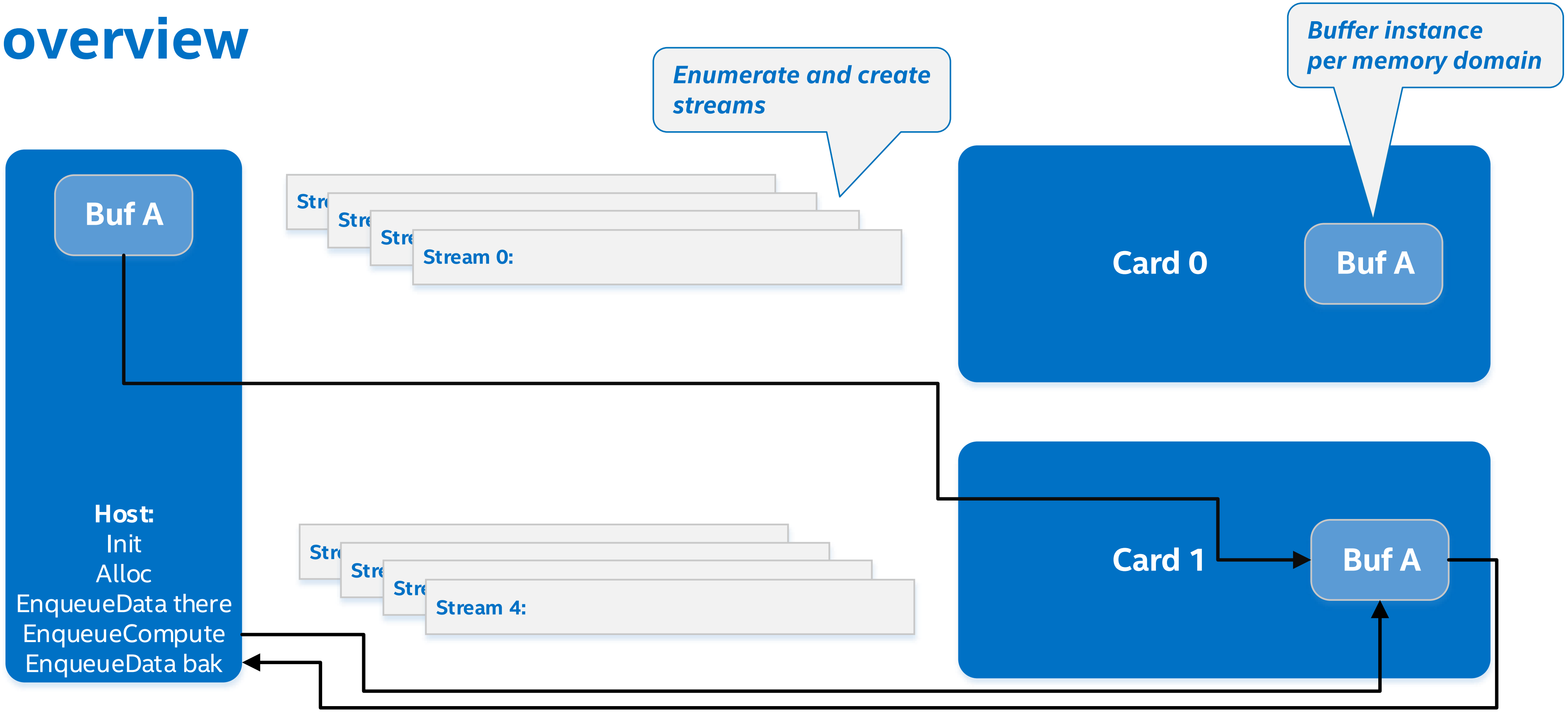
As needed for special cases

hStreams core API

Intel® Coprocessor Offload Interface (COI)

Intel® Symmetric Communication Interface (SCIF)

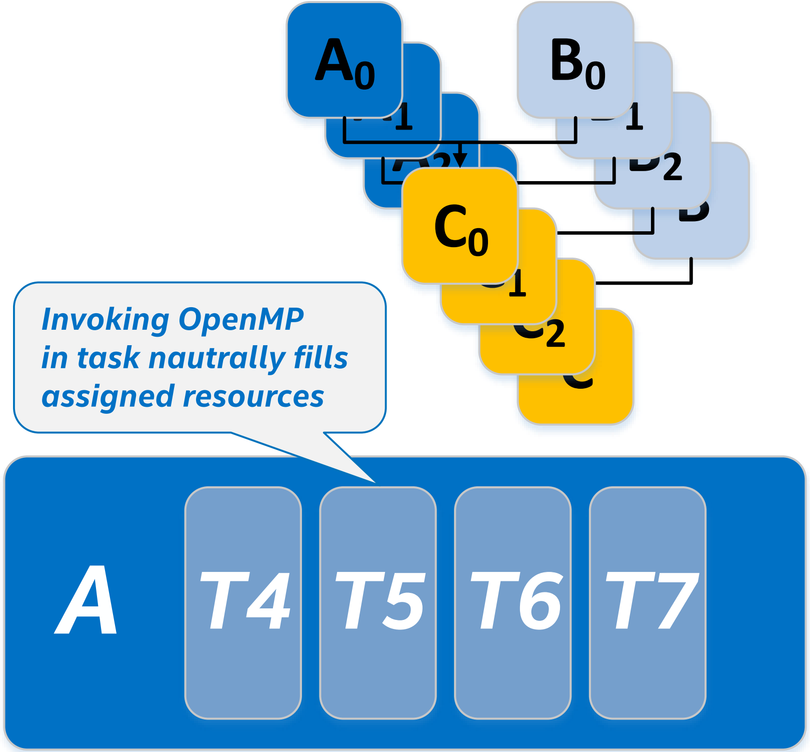
# Usage overview



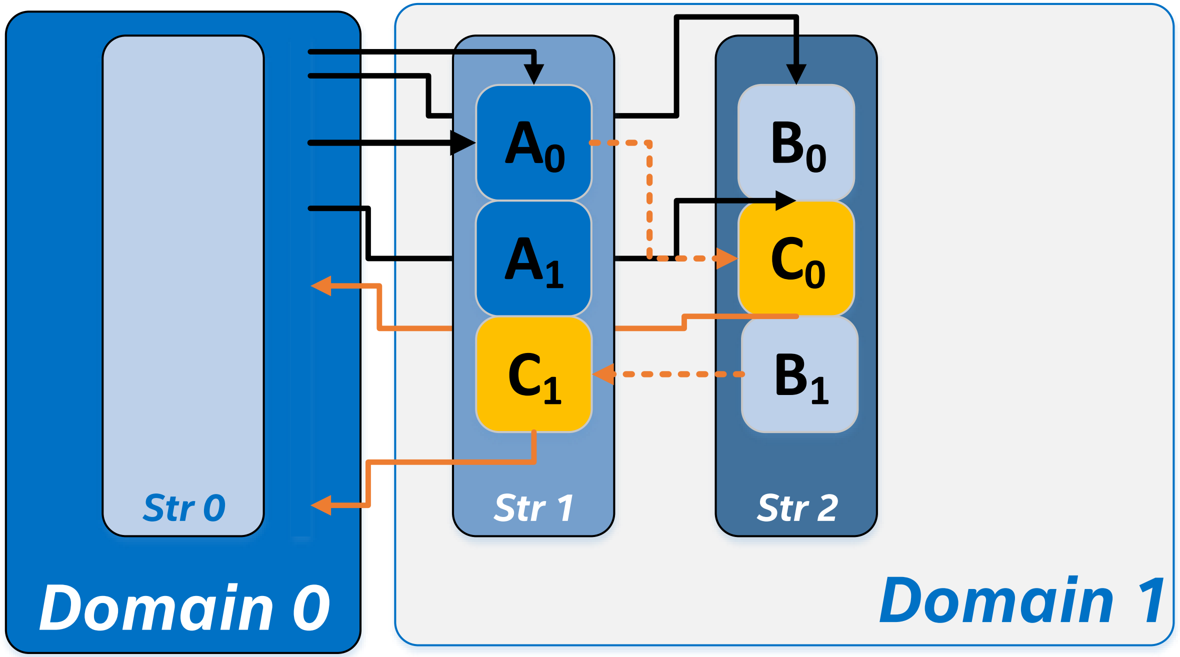
- **Streams are FIFOs, mapped to arbitrary subsets of cards; subsets can exactly match to oversubscribe**
- **All Enqueue* is always async**

# Illustration

- **Decompose work into tasks**
  - User-defined, unlike MKL; can use OpenMP or TBB
- **Streams are associated with compute resources**
  - OpenMP or TBB expand to fill stream
  - Abstraction of resource binding enables portability
- **Explicitly move data into domains**
- **Assign tasks to streams**
  - Let FIFO handle dependences within streams
  - Explicitly code cross-stream dependences



*Invoking OpenMP in task nautrally fills assigned resources*



*EnqueueData, str 1, $A_0$*
*EnqueueData, str 2, $B_0$*
*EnqueueCompute, str 1, funcA*
*EnqueueCompute, str 2, funcB*
*EventWait, str2, [$A_0$]*
*EnqueueCompute, str 2, funcC*
*EnqueueData, str 2, $C_0$*

# What we use from COI

- **COI daemon, vs. setting up server**
- **COIProcess, to enable sharing of data among logical streams and with (enabled) offload clients**
- **COIPipeline, to manage cpuset**
- **COIEvents**
  - To manage dependences within and across COIPipelines
- **COIBuffers**
  - To manage efficient data movement across PCIe
  - But not
    - *To span multiple memory domains*
    - *To manage dependences with COIPipelineRunfunctions*
    - *To manage eviction - defeated by not making RunFunction args; requires use of SetState to “unpersist”*

# Performance goals

- **Depend on COI perf features and maturity**
- **Data transfer only**
    - To and from card hit PCIe bandwidth limit
    - Limited overhead for dividing up transfers across streams
    - Eventual benefit from overlapping to and from in diff streams
- **With compute**
    - Streams help overlap compute and data transfers
    - Streams help cover memory latency with very large working sets

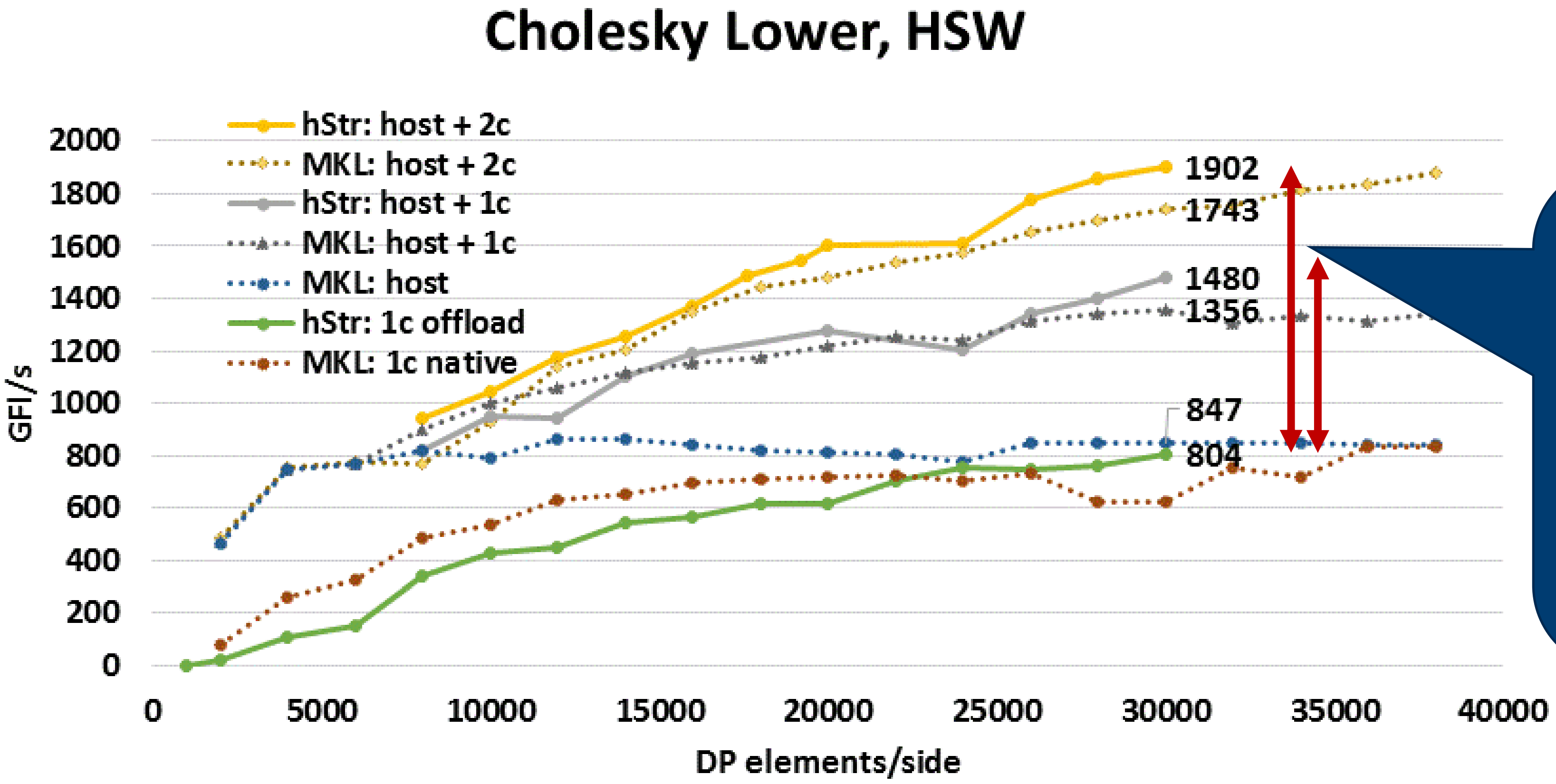
# Async Block Matrix Multiply (BMM) vs. Sync Full Matrix Multiply

- **[C] = [A]*[B]  (M x K)*(KxN) = (MxN)  (row x col)**
- **Async BMM uses the new hStreams with app api**
  - 4 places and 4 Logical Streams
  - 4x4 blocking used for comparable matrix size
- **Sync'd Full Matrix Multiply uses #pragma offload.  Can time only the offload dgemm.**
- **Did 5 runs w/Block MatMul, reported average of 4 after discarding the 1st**
- **Timed #pragma offload_transfer's for Effective Gigaflops on same given matrix sizes.  Includes time to send A & B to card, mat mul, and get the C matrix back**

# Invocation

- **Parameters of target function**
  - Scalar args, then heap args
  - **All** must be 64 bits.  Otherwise bits will get incorrectly mapped into arguments
  - Caller and callee outside of hStreams are responsible for type casting
  - Return value can have variable size
- **Be careful to type cast arguments carefully**
- **A string format convenience wrapper is possible**

# TILED CHOLESKY – NEW MPSS 3.6 VERSION OF HSTREAMS



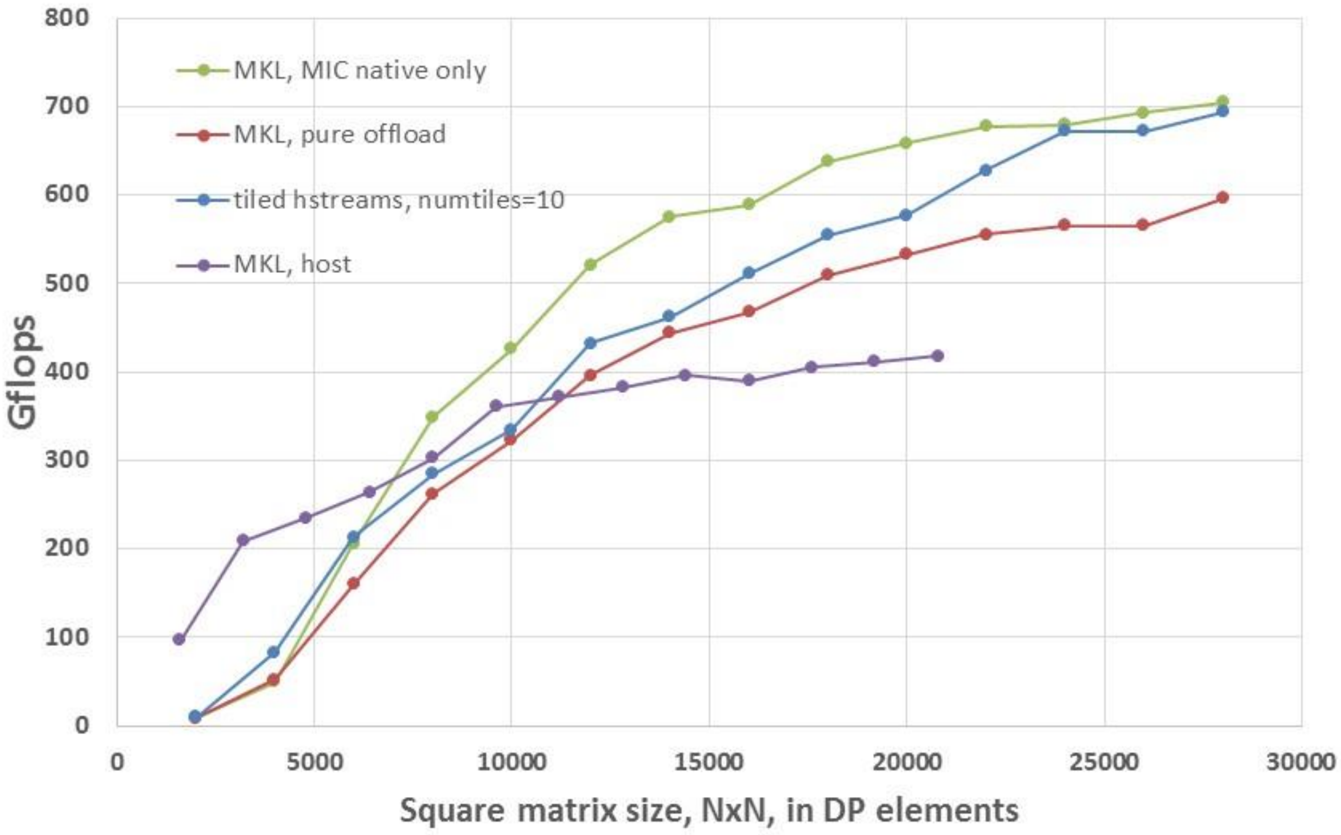
2 cards + host vs. host only: 2.25x

1 card + host vs. host only: 1.75x

Compared favorably with MKL automatic offload, after only 4 days' effort

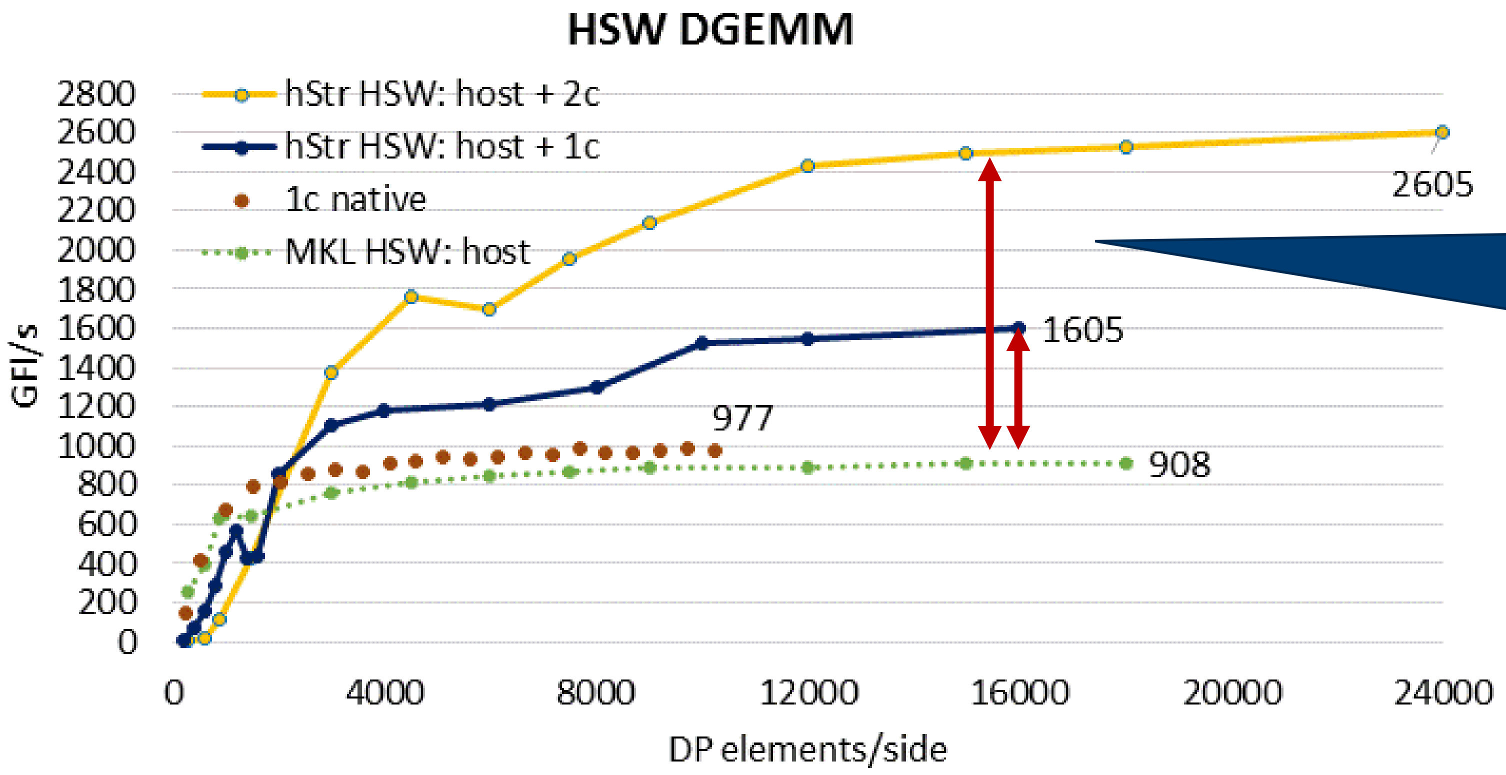
System info:
Host: E5-2697v3 (Haswell), 64GB 1600 MHz; SATA HD; Linux 2.6.32-358.el6.x86_64; MPSS 3.5.2, hStreams for 3.6
Coprocessor: KNC 7120a FL 2.1.02.0390; uOS 2.6.38.3; 15.1.133 compiler
Intel compiler v16/MKL 11.3, Linux

# TILED CHOLESKY – OLD MPSS 3.5 VERSION OF HSTREAMS



System info:
Host: E5-2697v2, 64GB 1600 MHz; SATA HD; Linux 2.6.32-358.el6.x86_64; MPSS 3.5 RC1
Coprocessor: KNC 7120a FL 2.1.02.0390; uOS 2.6.38.3; 15.1.133 compiler

# TILED MATRIX MULTIPLY – NEW MPSS 3.6 VERSION OF HSTREAMS



2 cards + host vs. host only: 2.74x

1 card + host vs. host only: 1.77x

System info:

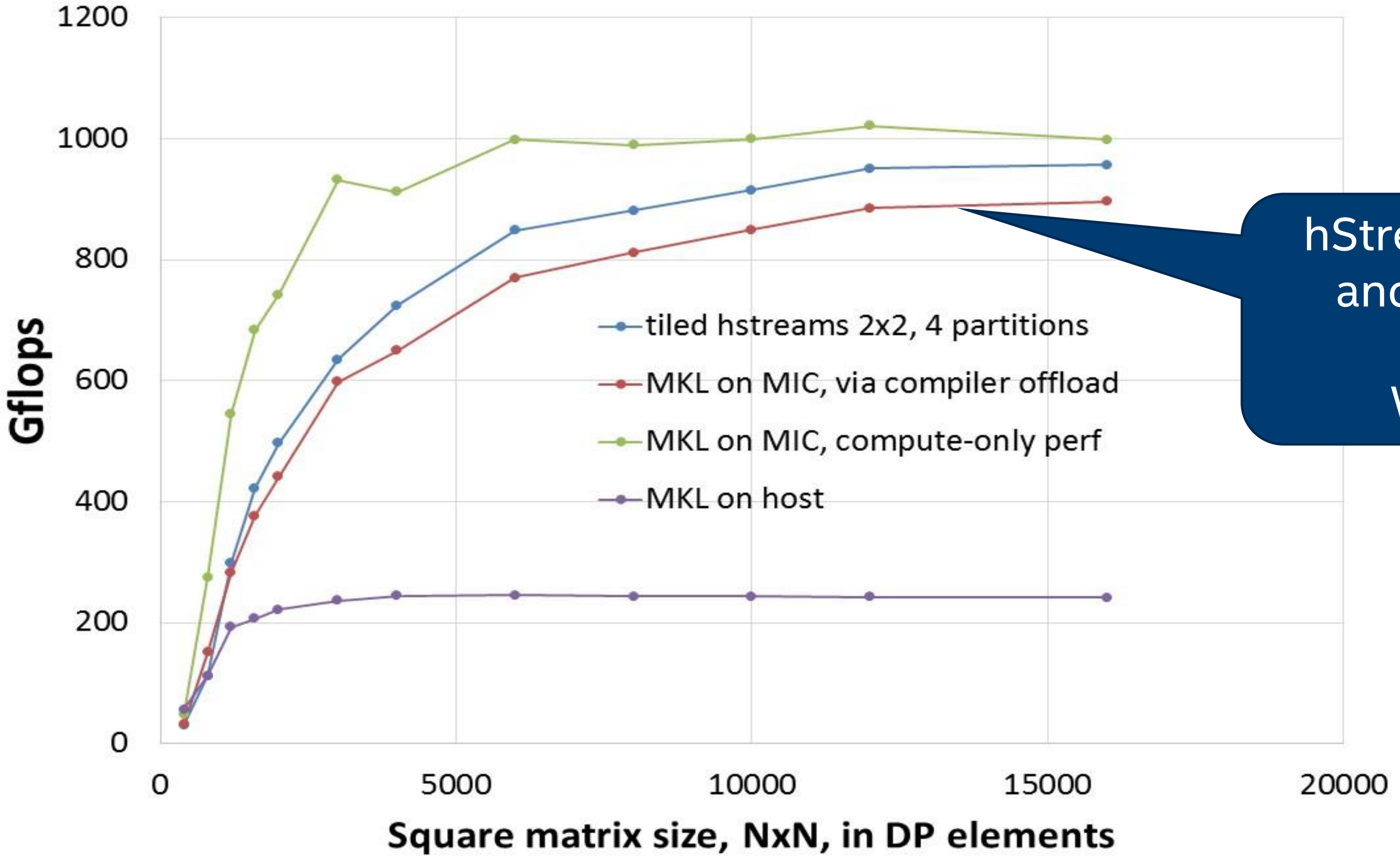
Host: E5-2697v3 (Haswell), 64GB 1600 MHz; SATA HD; Linux 2.6.32-358.el6.x86_64; MPSS 3.5.2, hStreams for 3.6

Coprocessor: KNC 7120a FL 2.1.02.0390; uOS 2.6.38.3; 15.1.133 compiler

Intel compiler v16/MKL 11.3, Linux

Average of 4 runs after discarding the one run

# TILED MATRIX MULTIPLY – OLD MPSS 3.5 VERSION OF HSTREAMS



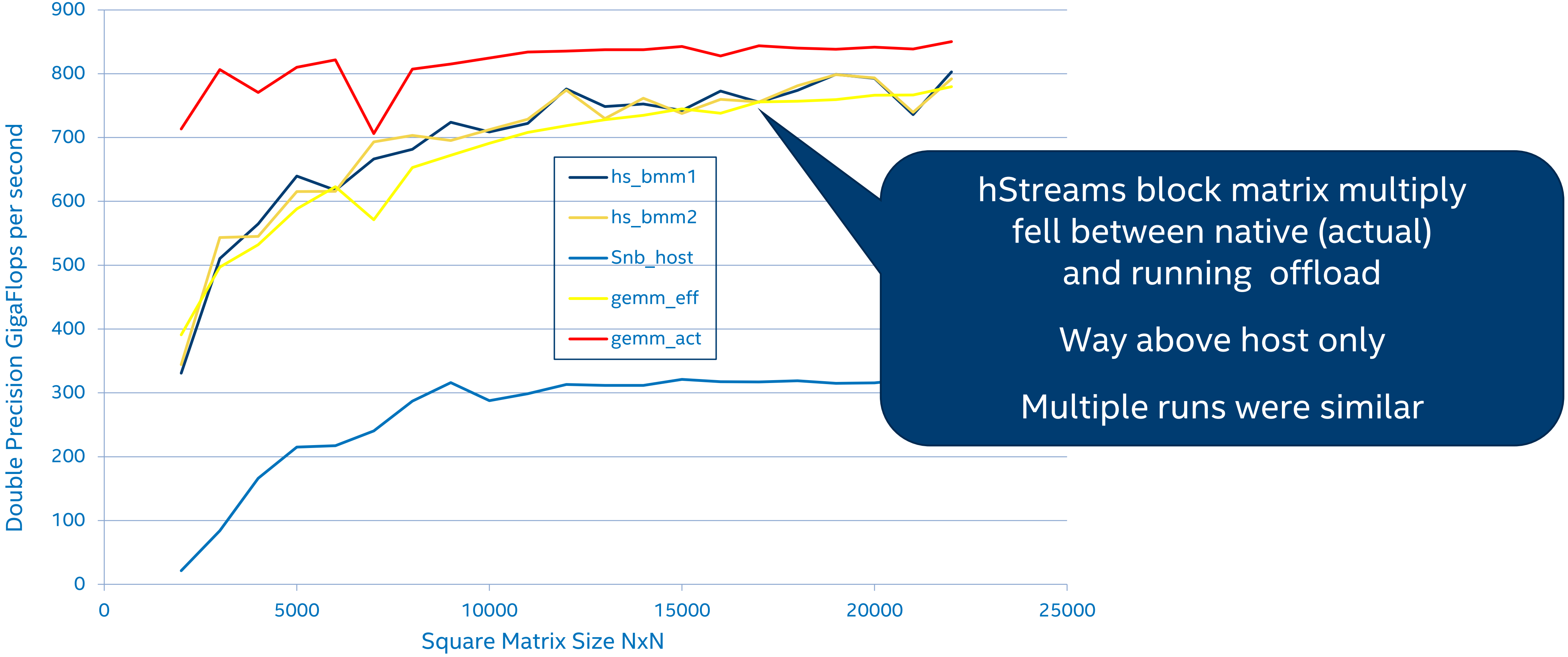
hStreams fell between native and sync compiler offload

Way above host only

System info:

Host: E5-2697v2, 64GB 1600 MHz; SATA HD; Linux 2.6.32-358.el6.x86_64; MPSS 3.5 RC1
Coprocessor: KNC 7120a FL 2.1.02.0390; uOS 2.6.38.3; 15.1.133 compiler

# TILED MATRIX MULTIPLY – OLD MPSS 3.4 VERSION OF HSTREAMS



hStreams block matrix multiply fell between native (actual) and running offload

Way above host only

Multiple runs were similar

System info:

Host: E5-2670, 64GB 1600 MHz; SATA HD; Linux 2.6.32-279.el6.x86_64; MPSS 3.4 RC1  
Coprocessor: KNC 7120a FL 2.1.02.0390; uOS 2.6.38.3; 14.0.1.106 compiler

# Data transfers

- Data transfer rates have been tuned, and can be further improved
- Performance approaches the expected PCIe bandwidth limits
- Just as for benchmarks like SHOC, buffers must be of substantial size, on the order of 1MB, before the overhead of DMA invocation is amortized away

This slide MUST be used with any slides with performance data removed from this presentation

# Optimization Notice

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# Notices and disclaimers I

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more complete information about performance and benchmark results, visit Performance Test Disclosure

This document contains information on products in the design phase of development.

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

# Notices and disclaimers II

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein.  You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm