Heterogeneous Streaming

Chris J. Newburn, Gaurav Bansal, Michael Wood, Luis Crivelli, Judit Planas, Alejandro Duran
Paulo Souza, Leonardo Borges, Piotr Luszczek, Stanimire Tomov, Jack Dongarra, Hartwig Anzt,
Mark Gates, Azzam Haidar, Yulu Jia, Khairul Kabir, Ichitaro Yamazaki, Jesus Labarta

Monday May 23, 2016
IPDPS/AsHES, Chicago

What do programmers want for a heterogeneous environment?

- **Separation of concerns → suitable for long life time**
  - Application developer does not have to become a computer scientist or technologist
  - Tuner has freedom to adapt to new platforms, with easy-to-use building blocks
- **Sequential semantics → tractable, debuggable**
- **Task concurrency → among and within computing elements**
- **Pipeline parallelism → hide communication latency**
- **Unified interface to heterogeneous platforms → ease of retargetability**

hStreams delivers these features

What is hStreams?

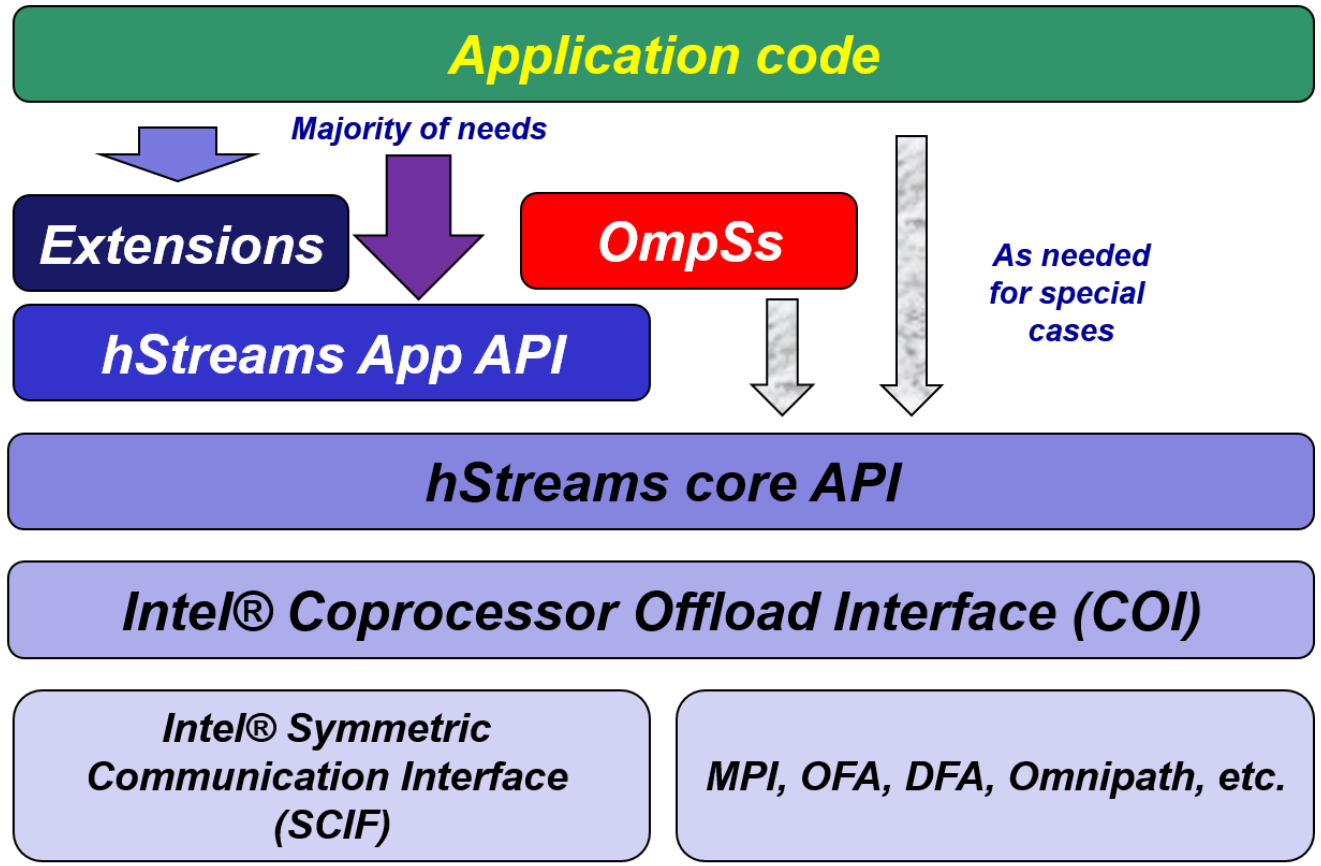
- **Library with a C ABI→ fit customer deployment needs**
  - Opened sourced: 01.org/hetero-streams, also lotsofcores.com/hstreams
- **Streaming abstraction**
  - FIFO semantics, out of order execution
  - Streams are bound to resources; compute, data transfer and sync actions occur in that context
- **Memory buffer abstraction**
  - Unified address space
  - Tuner can manage instances independently, e.g. in each card or node
  - Buffers can have properties, like memory kind
- **Easy retargeting to different platforms**
- **Dependences among actions**
  - Inferred from order in which library calls are made
  - Managed at the buffer granularity
- **Easy on ramp, pay as you go scheme**

Current deployments with hStreams

- **Production**
  - Simulia Abaqus Standard, v2016.1
  - Siemens PLM NX Nastran, v11
  - MSC Nastran, v2016
- **Academic and pre-production**
  - Petrobras HLIB – Oil and gas, 3D stencil
  - OmpSs from Barcelona Supercomputing Center

  - …more on the way

API layering



- Application frameworks can be layered on top of hStreams
- hStreams adds streaming, memory management on top of offload plumbing
- Possible targets include localhost, PCI devices, nodes over fabric, FPGA,s SoCs

hStreams Hello World

source

```
// Main header for app API (source)
#include <hStreams_app_api.h>
int main() {
        uint64_t arg = 3735928559;
        // Create domains and streams
        hStreams_app_init(1,1);
        // Enqueue a computation in stream 0
        hStreams_app_invoke(0, "hello_world",
                1, 0, &arg, NULL, NULL, 0);
        // Finalize the library. Implicitly
        // waits for the completion of
        // enqueued actions
        hStreams_app_fini();
        return 0;
}
```

1 other node,
1 stream

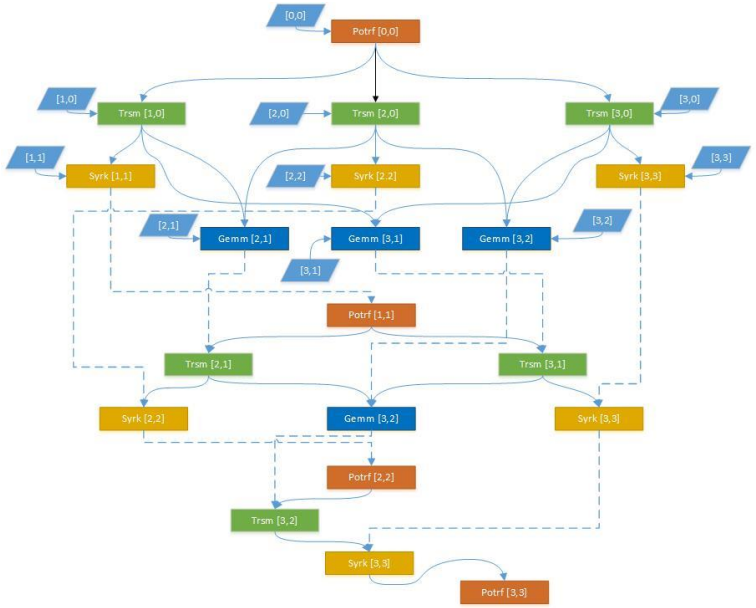
In stream 0,
1 argument

sink

```
// Main header for sink API
#include <hStreams_sink.h>
// for printf()
#include <stdio.h>
// Ensure proper name mangling and symbol
// visibility of the user function to be
// invoked on the sink.
HSTREAMS_EXPORT
void hello_world(uint64_t arg)
{
        // This printf will be visible
        // on the host. arg will have
        // the value assigned on the source
        printf("Hello world, %x\n", arg);
}
```

Consider a Cholesky factorization, e.g. for Simulia Abaqus

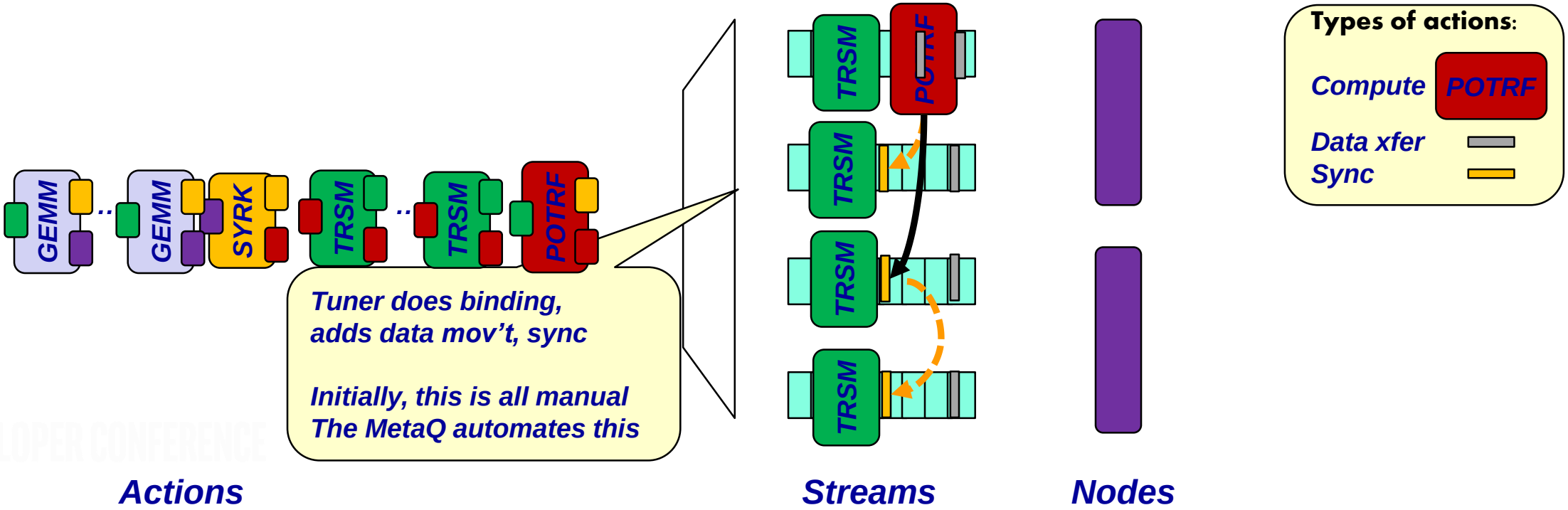
```
void tiled_cholesky(double **A)
{
  int k, m, n;
  for (k = 0; k < T, k++) {
    A[k][k] = DPOTRF(A[k][k]);
    for (m = k+1; m < T; m++) {
      A[m][k] = DTRSM(A[k][k], A[m][k]);
    }
    for (n = k+1; n < T; n++) {
      A[n][n] = DSYRK(A[n][k], A[n][n]);
      for (m = n+1; m < T; m++) {
        A[m][n] = DGEMM(A[m][k], A[n][k], A[m][n]);
      }
    }
  }
```



It looks like there's opportunity for concurrency
But do you want to create an explicit task graph for each of these?

So what's a good abstraction? How about streams?

- **A sequence of library calls induces a set of dependences among tasks**
  - The dependence graph is never materialized
  - A tuner or runtime can bind and reorder tasks for concurrent execution and pipelining



- **Manual (now): individual streams – bound to subsets of threads**
  - Tuner does the compute binding, data movement, synchronization
- **MetaQ (future version) – spans all resources**
  - Pluggable runtime does compute binding, data movement, synchronization

Responsibility

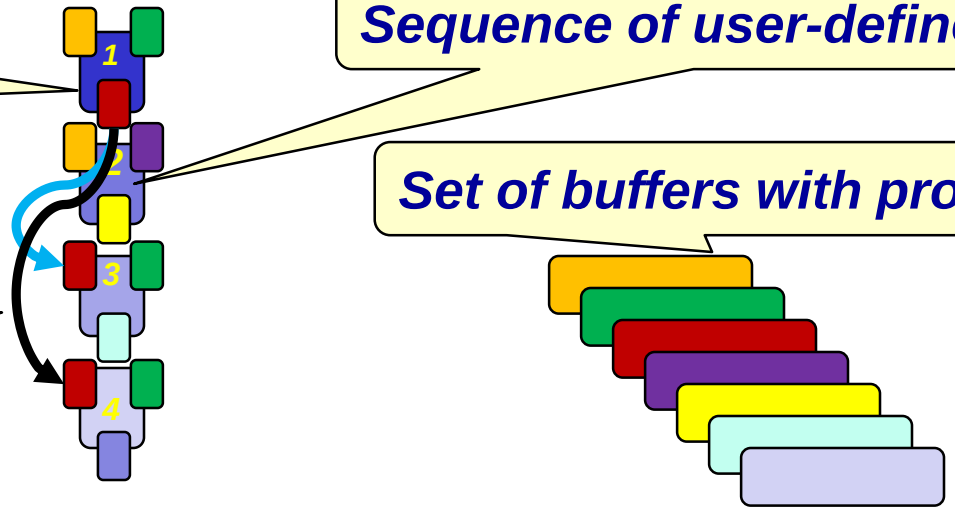
App developer

Input and output operands

Sequence of user-defined tasks

Set of buffers with properties

Induced dependences

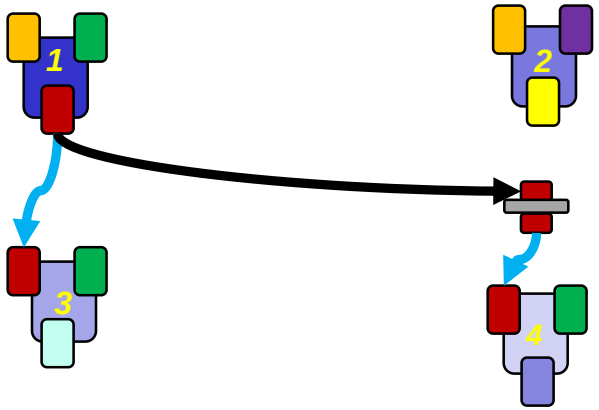


Ordering
Distribution
Association

Tuner

Stream 0

Stream 1



Sync action inserted
Induces dependences only on "red"
Non-dependent tasks could pass

FIFO semantic, OOO execution

Favorable competitive comparison

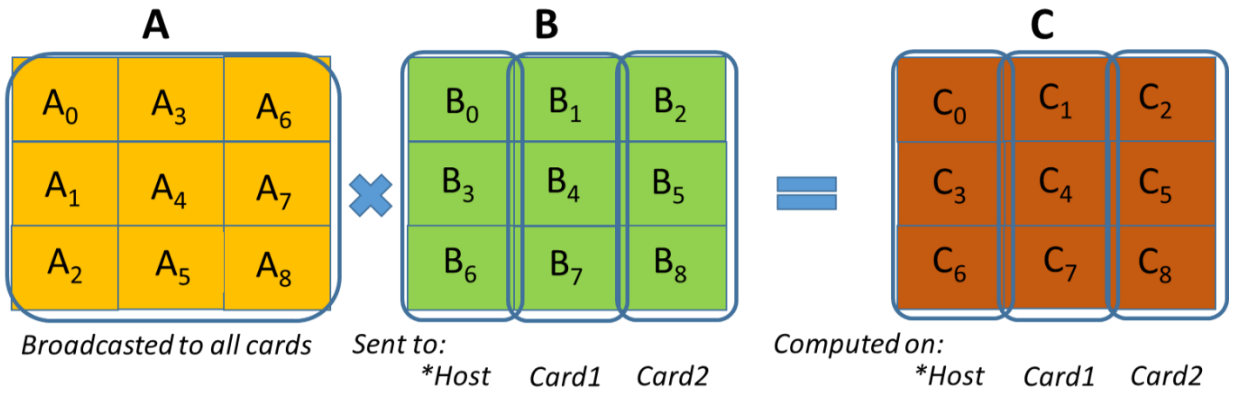
- **Similar approaches**
  - CUDA Streams
  - OpenCL (OCL)
  - OmpSs
  - OpenMP offload
- **Also at Intel**
  - Compiler Offload Streams
  - LIBXSTREAM
- **Fewer lines of extra code**
  - 2x CUDA Streams, 1.65x OCL
- **Fewer unique APIs**
  - 2.25x CUDA Streams, 2x OCL
- **Fewer API calls**
  - 1.9x CUDA Streams, 1.75x OCL

# additional source code lines vs. basic tiled version						
Descr.	hStreams	CUDA	OMP 4.0	OMP 4.5	OmpSs	OpenCL
Initialization	2	9	0	0	0	8
Data alloc	3	6	0	3	0	6
Data transfers	7	7	0	7	0	7
Computation	0	2	1	1	3	0
Synchronization	1	1	0	1	1	1
Data transfers	2	2	0	2	0	2
Data dealloc	3	6	0	3	0	6
Finalization	2	7	0	0	0	3
Total	20	40	1	17	4	33

# support variables	
hStreams	CUDA
1 matrix[M][N][L] (events)	1 matrix [M][N] (streams) 1 matrix [M][N][L] (events) 1 opaque pointer (CUBLAS) 1 matrix [M][L] (dev. addr.), 1 matrix [L][N] (dev. addr.), 1 matrix [M][N] (dev. addr.)

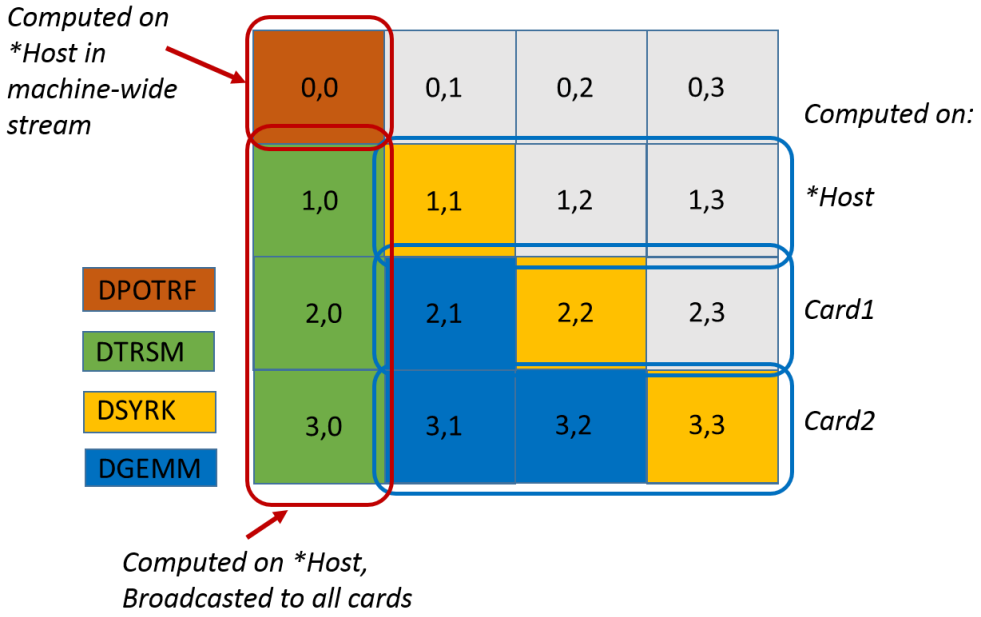
Metric	hStreams	CUDA	OMP 4.0	OMP 4.5	OmpSs	OpenCL
Unique APIs	8	18	1	5	5	16
Total APIs used	16	31	1	14	9	28
GFl/s,$(10K)^2$	916	N/A	460, 180	N/A	762	35

Tiling and scheduling



Tiling and binding for matrix multiply

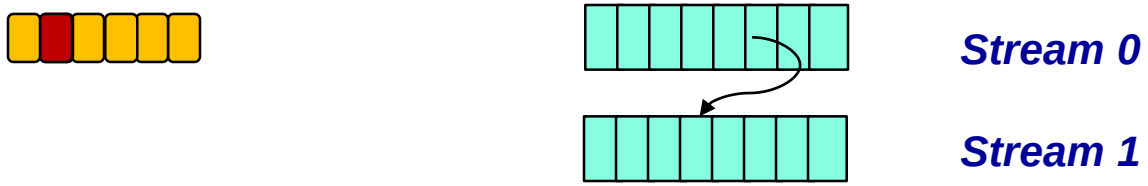
- Matrices are tiled
- Work for each tile bound to stream
- Streams bound to a subset of resources on a given host or MIC
- hStreams manages the dependences, remote invocation, data transfer implementation, sync



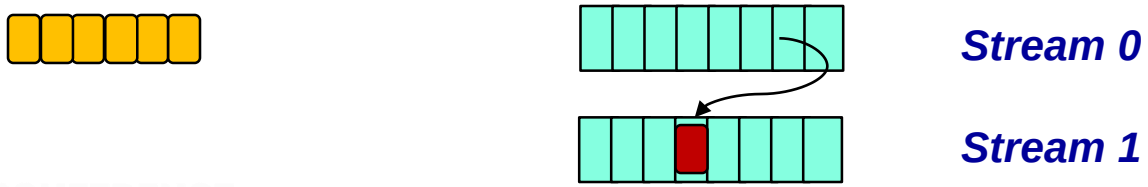
Tiling and binding for Cholesky

Benefits of synchronization in streams

- **Synchronization outside of streams – OmpSs on CUDA Streams**
  - OmpSs checks if cross-streams dependences satisfied
  - Host works around blocking by doing more work

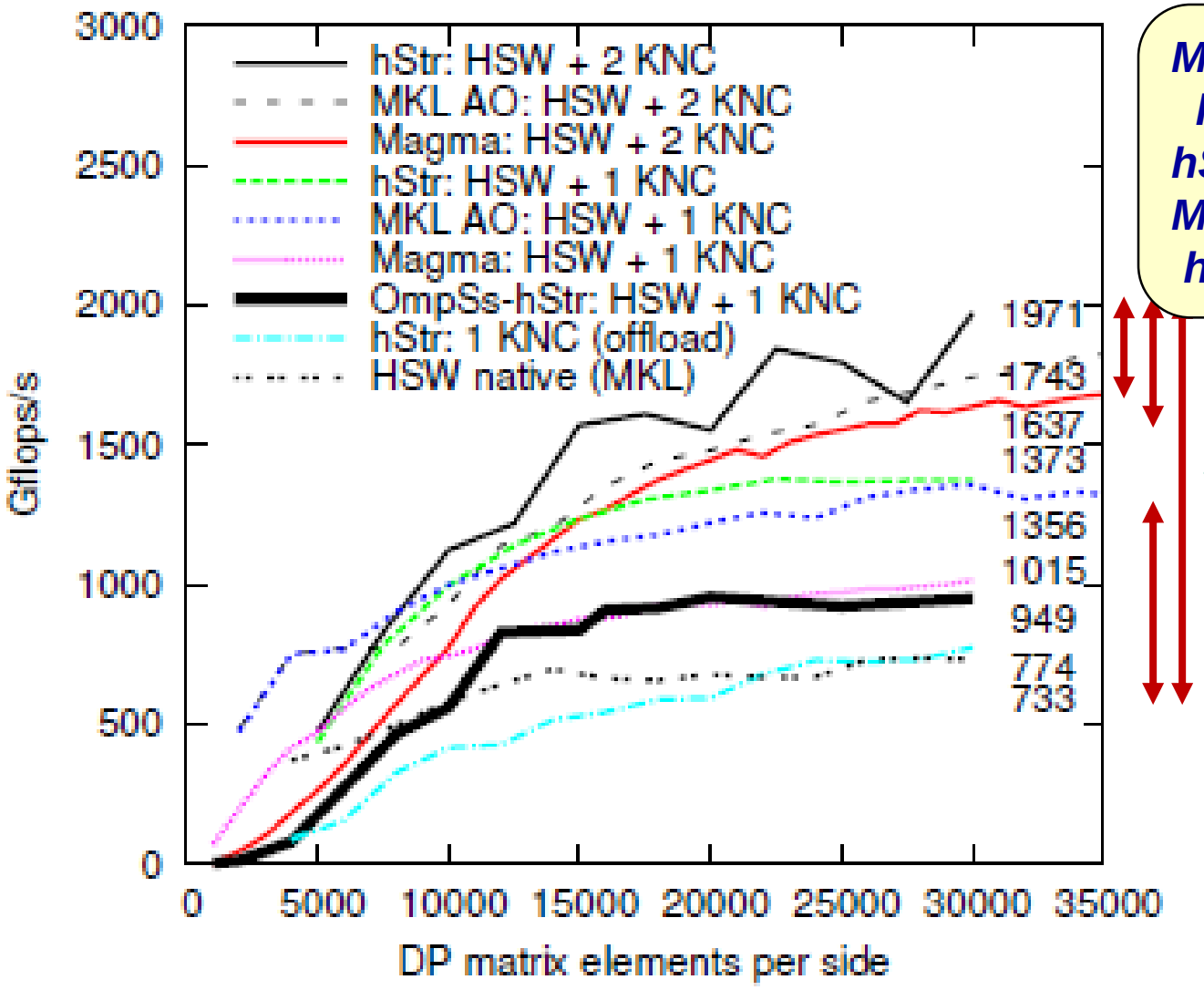


- **Synchronization inside streams – OmpSs on hStreams**
  - Cross-stream sync action enqueued within stream



- **Performance impact**
  - For a 4Kx4K matrix multiply, the host was the bottleneck
  - Avoiding the checks for cross-stream dependences yielded a 1.45x perf improvement

Tiled Cholesky – MAGMA, MKL AO



MAGMA* uses host only for panel on diagonal, hStreams balances load to host more fully
hStreams optimizes offload more aggressively
MAGMA tunes block size and algo for smoothness
hStreams is jagged since block size is less tuned

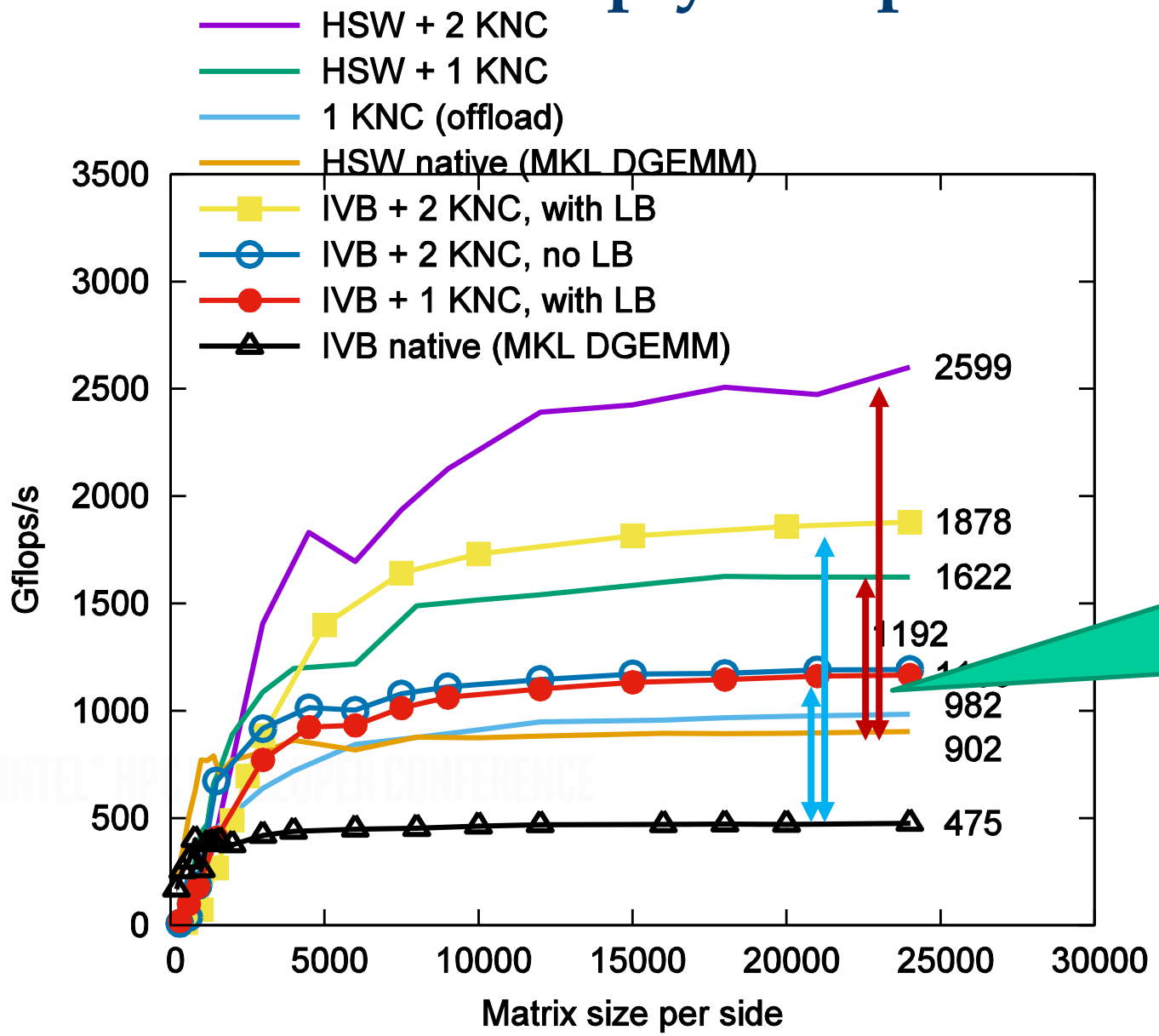
HSW:
2 cards + host vs. host only: 2.7x
1 card + host vs. host only: 1.8x

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

System info:
Host: E5-2697v3 (Haswell) @ 2.6GHz, 2 sockets
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; Intel compiler v16/MKL 11.3, Linux
Average of 4 runs after discarding the first run

MAGMA MIC 1.4..0 data measured by Piotr Luszczek of U Tenn at Knoxville

Tiled matrix multiply – impact of load balancing

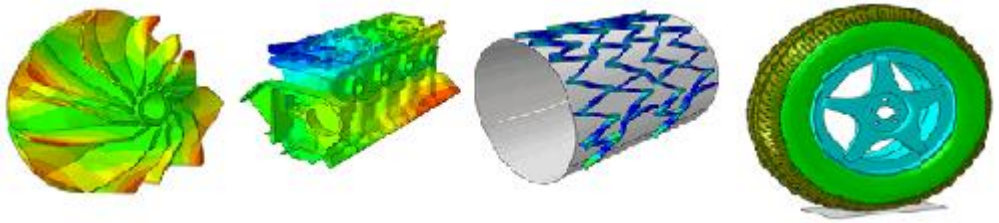


Good scaling across host, cards
Load balancing (LB) matters more for asymmetric perf capabilities (IVB vs. KNC)

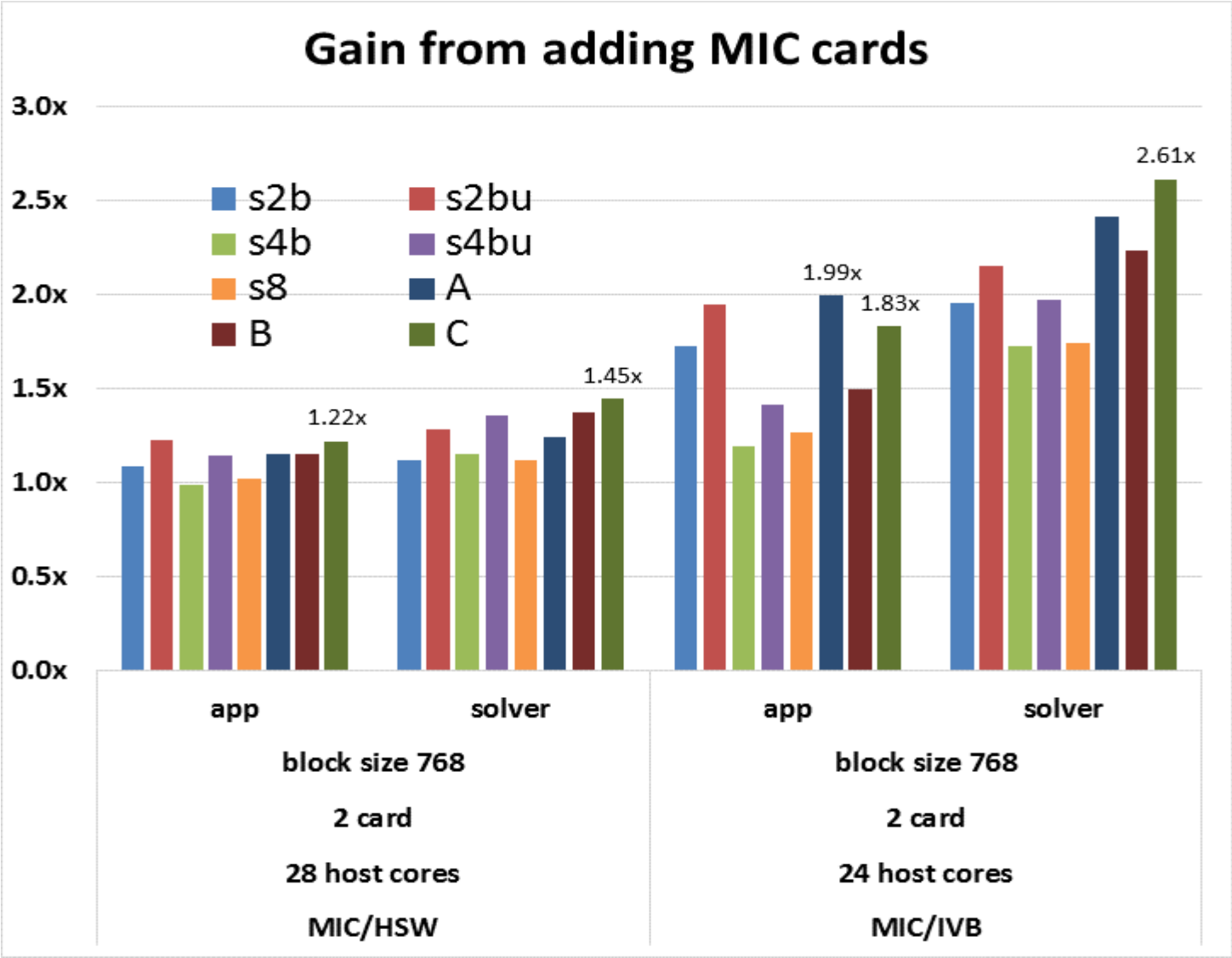
HSW:
2 cards + host vs. host only: 2.89x
1 card + host vs. host only: 1.80x
IVB:
2 cards + host vs. host only: 3.95x
1 card + host vs. host only: 2.45x

System info:
Host: E5-2697v3 (Haswell) @ 2.6GHz, v2 (Ivy Bridge) @ 2.7GHz,
Both 2 sockets, 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; Intel compiler v16/MKL 11.3, Linux
Average of 4 runs after discarding the first run

Simulia Abaqus Standard*



- Offload to two cards, from IVB or more-capable 28-core HSW
- Showing modest gains from using 2 cards in addition to host on more-capable HSW
- Up to 2x at app level on less-capable 24-core IVB



System info:
Host: E5-2697v3 (Haswell) @ 2.6GHz, v2 (Ivy Bridge) @ 2.7GHz,
Both 2 sockets, 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; Intel compiler v16/MKL 11.3, Linux
Average of 4 runs after discarding the first run

Simula Abaqus Standard preproduction v2016 results measured by Michael Wood of Simulia
There are no guarantees that the formal release will have the same performance or functionality

Conclusion: results delivered by hStreams

- **Support for heterogeneity**
  - Offload to multiple cards, localhost
  - Portability, retargetability
  - Effective layering above and below hStreams
- **Ease of use**
  - ~2x fewer lines of code, fewer API calls, fewer unique APIs, less variable allocation
  - 1.4x lower overheads for cross-stream coordination
  - Ease of design exploration: target affinity, degree of tiling, number of streams
  - Ease of porting and future proofing through separation of concerns
- **Performance**
  - Outperformed MAGMA and MKL Automatic Offload by 10%
  - Perfect scaling using MPI and multiple cards on Petrobras
  - Boosted Petrobras HLIB by 10% by overlapping communication with computation
  - 2^{+}x of just host by adding 2 cards

Related work

- **Offload libraries: CUDA Streams, OpenCL**
  - Explicit event creation, can only wait on 1 event at a time
  - Fewer unique APIs, fewer extra lines of code
  - Also: Qualcomm MARE, StarPU, TBB Flow Graph
- **OmpSs**
  - Uses dynamic scheduling, data movement, sync on top of hStreams or CUDA Streams
  - Offload to 1 card only; does not target localhost
  - Source-source compiler
- **Compiler based**
  - Intel Offload Streams: offload only; does not target localhost
  - OpenMP 4.x
    - Task scheduling within a single domain, offload to other domains
    - Does not support stream abstraction, dependences enforced only at the same nesting level
- **Does not depend on C++**
  - SyCL, Phalanx, CnC, UPC++, CHARM++, TBB FG, Kokkos, Legion, Chapel

What would you like to ask or discuss?

hStreams vs. CUDA Streams performance

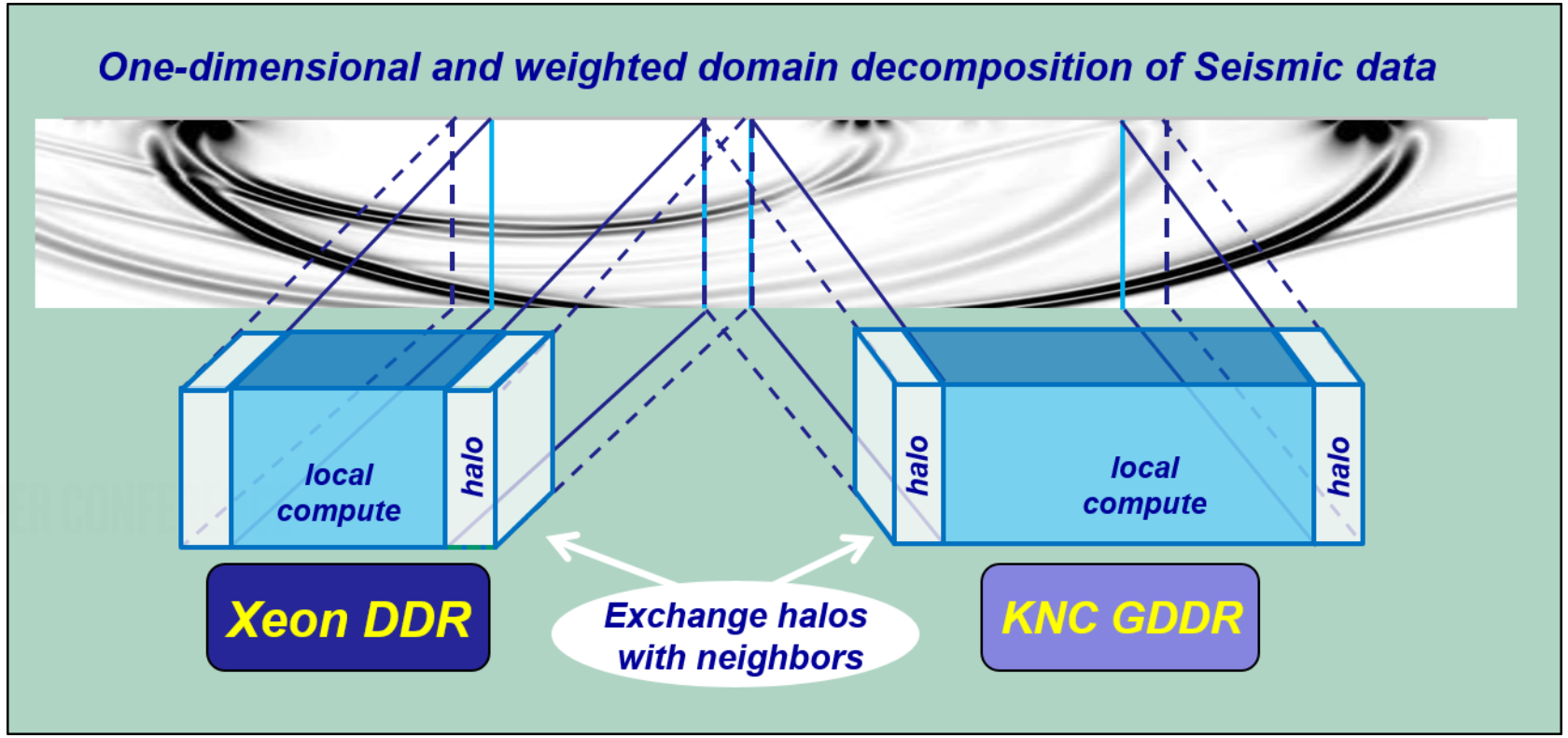
- **Setup: factorize a supernode with a Simulia standalone harness**
  - Compare offload to K40x and KNC, with the same host (IVB)
- **Upper bound: (total solver time) – (on-card time) to factor out HW diffs**
  - Measurement methodology may under-count K40x non-kernel time
- **Normalized: (ratio of solver time)(ratio of native kernel time)**
  - Lower hStreams overheads balance higher KNC card-side times
- **hStreams shows lower overhead than CUDA Streams**

	S4b	S8	"A"
CUDA Streams non-kernel	9.8s	7.6s	3.1s
hStreams non-kernel	5.2s	4.4s	2.7s
hStreams advantage vs. CUDA Streams, upper bound	1.89x	1.71x	1.12x
hStreams advantage vs. CUDA Streams, normalized	1.28x	1.24x	1.03x

Actual hStreams advantage in between these

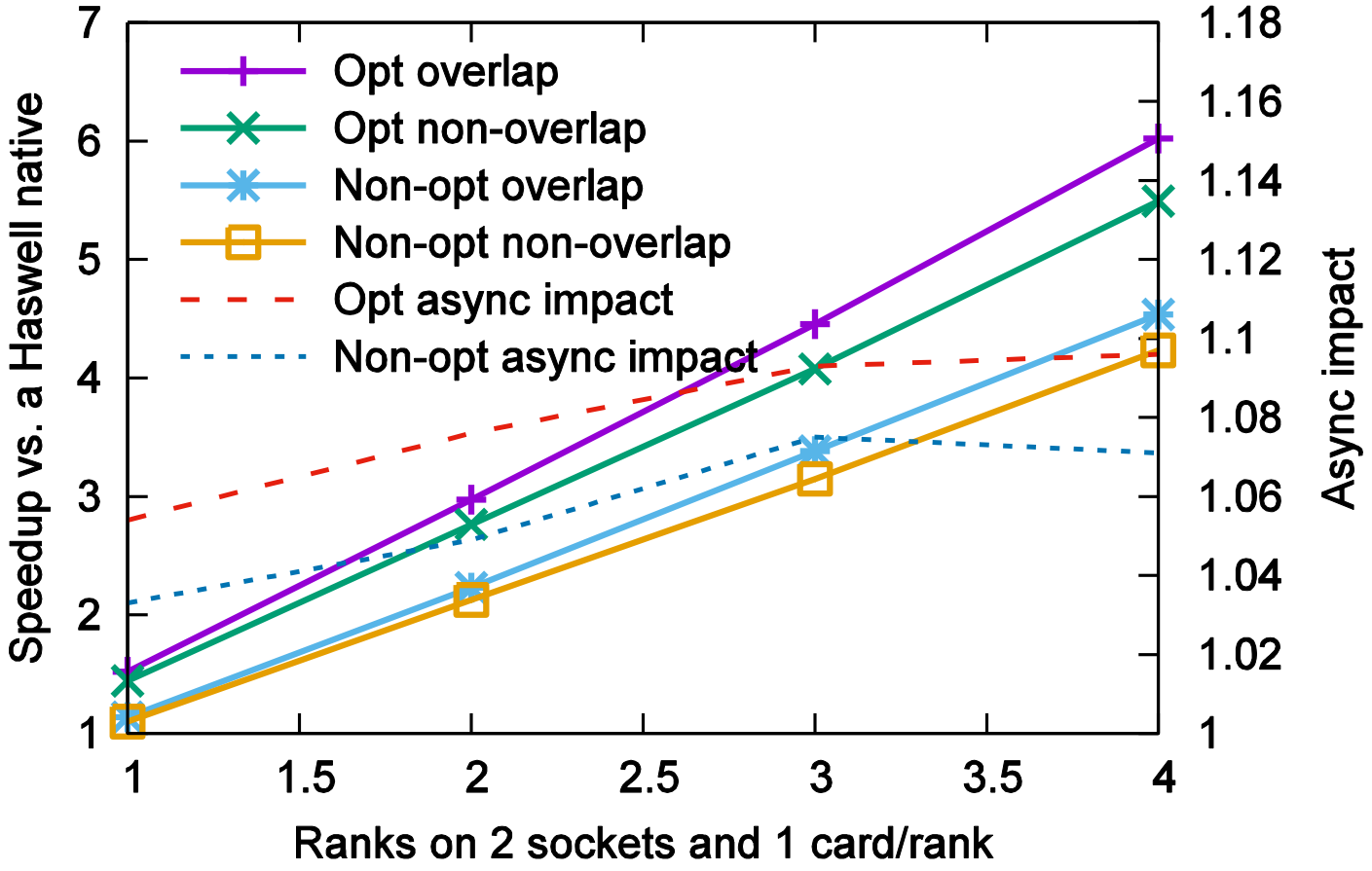
Petrobras application

- Oil and gas application, performs reverse time migration
- A high-level Fortran90 library called HLIB abstracts CUDA, OpenCL and CPU
- hStreams support was added by Paulo Souza of Petrobras



Petrobras* HLIB (Heterogeneous library)

- Petrobras's current code executes one task at a time, across a whole card, and doesn't yet use host
- 1.10x benefit from using asynchronous pipelining for optimized (shorter) code, 1.07x for unoptimized
- Benefit (not shown) from 1 MIC is 1.5x, 4 MICs is 6.0x
- Submitted to IPDPS15



System info:
Host: E5-2697v3 (Haswell) @ 2.6GHz, 2 sockets
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; Intel compiler v16/MKL 11.3, Linux
Average of 4 runs after discarding the first run

Petrobras data from preproduction HLIB code measured by Paulo Souza of Perobras
There are no guarantees that the formal release will have the same performance or functionality

Configurations

Specification	Intel Xeon Processor E5-2697v2 (IVB) and E5-2697v3 (HSW)	Intel Xeon Phi Coprocessor C0-7120A (KNC)	NVidia K40x
Skt,Core/Skt,Thr/Core	2S,12C(v2),14C(v3),2T	1S, 61C, 4T	1S, 15C, 256T
SP, DP width, FMA	{8,4,N (v2) 8,4,Y (v3)}	16,8,Y	192, 64, Y
Clock (GHz)	2.7(v2) 2.6(v3)	1.33 (turbo)	0.875 (turbo)
RAM (GB)	64 DDR3-1.6GHz	16 GDDR5	12 GDDR5
L1 data, instr (KB)	32,32	32,32	64
L2 Cache (KB)	256	512	roughly 200
L3 Cache (KB)	32K(v2),35K(v3) (sh)	-	-
OS, Compiler	{RHEL 6.4, Intel 16.0}	{Linux, Intel 16.0}	-
Middleware	MPSS 3.6	MPSS 3.6	CUDA 7.5

Simulia harness standalone performance

- Setup: factorize a supernode with a Simulia standalone harness
- Standalone data not available for K40x; Intel had only a MIC version
- HSW is a bit faster than KNC; IVB is much slower

KNC offload	HSW host as target	IVB host as target
2.35s	2.24s	4.27s
4 60-thread streams	3 9-thread streams	3 7-thread streams