HSTREAMS: FINDING THE RIGHT ABSTRACTION FOR THE WAY FORWARD

Argonne Leadership Computing Facility
June 1, 2016
Objectives of this presentation

- Introduce a heterogeneous streaming model, hStreams
- Suggest ways that we can make it easier to use platform features
- Explore your system design ideas and usage models
- Solicit your feedback and engagement on relevant codes
hStreams: a streaming task/data programming model

Before hStreams: Some ISVs already refactored to use target-agnostic Streaming API to reach GPUs

- We met and surpassed the competition
- Enable ISVs on our HW with relatively low enabling cost vs. complete rewrite
- Expected to work with self-boot and offload over fabric – Xeon or KNL
Summary of what hStreams offers

- Task concurrency ↷ path to parallelism
  - Especially for a modest number of small tasks
- Intuitive, uniform interface ↷ make hetero platforms accessible
  - Applicable to offload, local host, clusters
- Sequential semantics ↷ incremental, debuggable
- Out of order execution ↷ aggressive runtime
- Pay as you go ↷ easy to get started, powerful controls
- Suitable for deployment ↷ C ABI, not dependent on updating compiler
- Composable, interoperable ↷ Fits real-world contexts
Current deployments with hStreams

- **Open sourced:** [01.org/hetero-streams](01.org/hetero-streams)
  - Look for a tutorial there, e.g. see how to convert dusty-deck loop nests to concurrent tasks

- **Also see** [lotsofcores.com/hstreams](lotsofcores.com/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, SoCs
What’s the next big thing?
What does it take to meet the developers where they are?

- Excited about platform
- Overwhelmed with the complexity of platform features
  - Memory kinds, policies; computing kinds, configurations
- Make platform features accessible, intuitive to use

- Excited about parallelism
- *Maybe* moderately successful in threading, vectorization, memory tuning
- Discouraged by having to become an expert in languages, tools, APIs
- Overwhelmed by complexity of scheduling tasks in unpredictable environment
- Offer intuitive task interface, extensible for dynamic scheduling
Outline

- Heterogeneous platforms
- Exposing parallelism
- Separation of concerns
- Mapping across the gap: MetaQ

- Code
  - Needs assessment
  - Performance
  - Buffer properties

- Call to action
Execution on heterogeneous targets

Async invocation
Memory management
Data movement

Target self vs. other

IA or not

Over there

PCIe device
Wide CPU
- local host
- other node
Accelerator

KEY
here now in hStreams
here soon in hStreams
maybe someday in hStreams
Scientists want platform richness...

- **Heterogeneous compute resources**
  - Xeon, Xeon Phi, FPGAs, GPUs
  - Sub-NUMA cluster or any meaningful domain

- **Heterogeneous memory**
  - DDR, HBM, 3D XPoint; RDMA to near or far over OmniPath
  - Affinitized to a sub-NUMA cluster

- **Unpredictable latencies**
  - PCIe, fabric
  - Various topologies

- ** Stuff that wasn't invented yet when the scientist coded the algorithm**
... but don’t want to become a computer scientist to get it

- **Scientist – Application Developer**
  - Not a computer scientist
  - Exposes parallelism
  - Wants to use features, but ignorant of how

- **Tuner**
  - Might be a computer scientist
  - Maps parallel work to target
  - Wants to use features with minimal effort and complexity

- **Infrastructure**
  - Written by ninja computer scientists
  - Harvests parallelism
  - Does the work to use features

*This is our job*

*We need to make it easy for devs to do these*

*Make it possible for the skeptical, hands-on dev have visibility and control*
Exposing parallelism

- **Distribution**
  - MPI

- **Threading**
  - OpenMP, TBB

- **Vectorization**
  - CilkPlus
Exposing parallelism - augmented

- Distribution
  - MPI
- Coordination, task scheduling
  - CnC, OpenMP tasks, streaming models
- Threading
  - OpenMP, TBB
- Vectorization
  - CilkPlus $\rightarrow$ OpenMP, ISPC

Josh Strodtbeck, Convergent Science:
“Scheduling dynamic tasks on very-parallel machines with dynamic dependences is the next big thing I need for my app. But I don’t want to become a computer scientist to accomplish that. I want the ninjas at HW vendors to make that easy for me.”
Task concurrency

- **Natural granularity for coarser-grained parallelism is a task**
  - Tasks can be executed on any resource - suitable for remote execution
  - Suitable for dynamic scheduling and dependence analysis

- **Start with straight-line code, end up with a sequence of task invocations**
  - Loop nests are encapsulated as tasks
  - Variables referenced are summarized in task arguments
  - Operands and their properties may be described
Consider a Cholesky factorization, e.g. for Simulia Abaqus

```c
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?
Consider a Cholesky factorization, e.g. for Simulia Abaqus

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

*Task function is named vs. called*

*Offer a valid schedule – sequential semantics*

*Pay as you go – the more you describe operands, the more the opportunity for concurrency*

*Either declare as a buffer…*

*Or describe it as an operand*
Dynamic scheduling, for portability and unpredictability

- Coming up with a good static schedule of tasks is becoming tougher
- Unpredictability
  - latency of data movement from memory and across network
- Portability
  - capabilities for computes and fabric vary
- Requires
  - Asynchronous execution
  - Dynamic scheduling
  - Integrated dependence mgt
  - Platform tuning
Making concurrency on a hetero platform easier

- **Natural granularity is a task**
  - A function with arguments, where heap arguments are in a **unified address space**
  - This does NOT cover all usage models, e.g. loop-level parallelism like SWARM

- **Performance comes from task concurrency**
  - To the scientist, tasks can execute anywhere
    - Task code runs on any HW for which a compiler and transport plumbing is available
  - Tuner’s job to decide where and when
  - Across any kind of resource
  - Across any fraction of a resource
  - Pipelining computation and communication
  - Uniform interface, no special handling

- **What would make this harder**
  - Non-uniform interface – OpenMP doesn't yet offer a uniform hetero interface
  - Complicated addressing for distributed memory
Do I need this?
   Maybe or maybe not

“As simple as possible but no simpler”
   — Albert Einstein
Task scheduling is trivial if

- Each task is so parallel it fills a whole resource

- There are so many tasks that each can execute on one thread

- Tasks execution drowns out communication latency

But is there data locality among subsets of these tasks?
Additional reasons to use fewer tasks

- There may be enough task parallelism to have one task per HW thread

- If memory requirements scale with tasks
  - you may run out of memory to hold your entire working set, e.g. in high-bandwidth memory
  - Examples: QMCPACK, ATLAS

- If there’s some per-task overhead
  - fewer tasks means better total utilization
  - if that overhead is threadable, it also helps latency
  - Examples: QMCPACK

- If there’s more opportunity for locality within a task than across tasks
  - let threads within a task be bound to computing resources which share caches or memory controllers (for sub-NUMA clustering)
Task scheduling is more interesting when

- There are several concurrent tasks, but they have limited parallelism

- Tasks have dependences among them, but the latency of execution and/or communication is unpredictable

- It becomes important to pipeline communication and computation

- The target platform is heterogeneous

See backup for data on efficiency vs. size and width
So what should the application developer reveal?

- **A sequence** of library calls induces a set of dependences among tasks
  - The dependence graph is never materialized

- **Optional:** properties of objects in memory
  - Size and shape of variables
  - Access characteristics, e.g. read only, latency sensitive, bandwidth sensitive

**Actions**

- Data operands only – no transfers
  - Sequence and operands imply dependences – no synchronization

**Types of actions:**

- Compute
- Data xfer
- Sync
Sequence of user-defined tasks

Set of buffers with properties

Input and output operands

Induced dependences

Ordering Distribution Association

FIFO semantic, OOO execution

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

Comput 0

Compute 1

Responsibility
App developer

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

- A sequence of library calls induces a set of dependences among tasks
  - 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
hStreams Hello World

#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;
} // source

#include <hStreams_sink.h>
#include <stdio.h>
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);
} // sink
Sample sequence

Initialize

\[ \text{hStreams\_app\_init} (Streams\_Per\_Domain=4, \ldots) \]

Allocate buffers

\[ \text{hStreams\_app\_create\_buf} (\text{Host\_Proxy\_Adr}=\text{ArrayA}, \text{Num\_Bytes}=4096*1024) \]

Transfer memory there

\[ \text{hStreams\_app\_xfer\_memory} (\text{Log\_Str\_ID}=3, \text{Src\_Adr} = A[5], \text{Dest\_Adr} = B[3], \text{Num\_Bytes}=4096, \text{Xfer\_Direction}=HSTR\_SRC\_TO\_SINK, \text{Event} = \&\text{Event1}) \]

Remote invocation

\[ \text{hStreams\_app\_invoke} (\text{Log\_Str\_ID}=3, "myfunc", \text{Scalar\_Args}=2, \text{Heap\_Args}=3, \text{Arg\_Array}=\text{Args}, \text{Return\_Val}=\text{NULL}, \text{RetVal\_Size}=0, \text{Event} = \&\text{Event2}) \]

\[ \text{hStreams\_app\_dgemm} (\text{Order}, \text{TransA}, \text{TransB}, M, N, K, \ldots) \]

Transfer memory back

\[ \text{hStreams\_app\_xfer\_memory} (\text{Log\_Str\_ID}=3, \text{Src\_Adr} = C[7], \text{Dest\_Adr} = C[7], \text{Num\_Bytes}=4096, \text{Xfer\_Direction}=HSTR\_SINK\_TO\_SRC, \text{Event} = \&\text{Event3}) \]

Synchronize

\[ \text{hStreams\_app\_thread\_sync} () \]

\[ \text{hStreams\_app\_stream\_sync} (\text{Log\_Str\_ID}=3) \]

Finalize

\[ \text{hStreams\_app\_fini} () \]
Sample sequence - future

**Initialize**
```
hStreams_app_init(StreamsPerDomain=4,...)
```

**Allocate buffers - implicit**
```
hStreams_app_create_buf(HostProxyAdr=ArrayA, NumBytes=4096*1024)
```

**Transfer memory there - implicit**
```
hStreams_app_xfer_memory(LogStrID=3,SrcAdr=A[5],DestAdr=B[3], NumBytes=4096,XferDirection=HSTR_SRC_TO_SINK, Event=&Event1)
```

**Remote invocation**
```
hStreams_MetaQ(“myfunc”,arg1, arg2, ...)
hStreams_app_invoke(LogStrID=3,”myfunc”,ScalarArgs=2,HeapArgs=3, ArgArray=Args,ReturnVal=NULL,RetValSize=0, Event=&Event2)
hStreams_app_dgemm(Order,TransA,TransB,M,N,K,...)
```

**Transfer memory back - implicit**
```
hStreams_app_xfer_memory(LogStrID=3,SrcAdr=C[7],DestAdr=C[7], NumBytes=4096,XferDirection=HSTR_SINK_TO_SRC, Event=&Event3)
```

**Synchronize - implicit**
```
hStreams_app_thread_sync()
hStreams_app_stream_sync(LogStrID=3)
```

**Finalize**
```
hStreams_app_fini()
```
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
Tiling and scheduling

- 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
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
Progression

- **Single function offload to MIC, e.g. DGEMM**
  - MSC Nastran, Siemens PLM Nastran, Petrobras

- **Higher-function targets MIC and host, e.g. Cholesky**
  - Simulia Abaqus, Petrobras iso3DFD, OmpSs
  - Tuner binds tasks to streams, managed data movement and sync

- **MetaQ with dynamic scheduling**
  - Heuristic engine available to bind, manage data and sync; pluggable interface
  - Two-layer scheduling – distinguish between waiting and data ready

- **Integration of IO with dependence system**
  - MPI receive and IO and async data movement all tied directly into dynamic scheduler
HPC use cases

- **Manufacturing**
  - Why: Biggest exposure to CUDA Streams/K40x, which does offload
  - What: Use multiple KNCs plus host, up to 2.6x
  - Metrics: Comparable performance, narrowed HW performance gap
  - What's next
    - Use remaining resources on host, as though it were offloaded
    - More tasks overlapped between sync points; expand to use dynamic scheduling

- **Oil and gas**
  - Why: Another opportunity to implement hStreams under a target-agnostic API (NV, AMD)
  - What: Use up to 4 KNCs (4x), overlapped communication and computation (1.1x)

- **Machine learning**
  - Why: big competitive exposure, enable KNM
  - What: standalone on KNM, KNB

- **Life sciences**
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
Needs assessment

- Separation of concerns
- Sequential semantics
- Task concurrency
- Dynamic scheduling
- Uniform interface to heterogeneous computing elements and memory kinds
- Declarative interface for memory and operand properties
Separation of concerns, easy on ramp

Problem

- app developers don't want to become computer scientists and low-level experts
- codes outlive the developers' involvement, run on platforms that weren't invented
- frameworks left unused if too hard to get started on, esp. with huge codes
- tuners may want total control over performance

Solution

- minimize app dev effort through abstraction: task parallelism and optionally memory reference properties
- tuners have simple interfaces for mapping work and memory to platform
- technology ninjas provide easy to use building blocks and optimized implementations; early measurements show overheads to be ~1%.
- tuners have control over partitioning, placement, ordering, resource management
- there's an easy on ramp, where users can use hStreams interfaces incrementally, on top of (large) codes they already have

Comparison

- OpenMP is quite prescriptive and requires users to understand OMP intimately
- TBB doesn't enable control over placement, ordering; little resource management
- Some schemes require different files, or require that all code be ported into a new framework, just to get started
Sequential semantics

Problem
- Difficult to think in parallel
- Cumbersome to specify task dependences as nodes and edges

Solution
- Specify task dependences through a sequential invocation of library calls that enqueue tasks, where operands can optionally be described in more detail
- Sequential semantics enable debuggability
- Underlying runtime's dependence analysis enables out of order, concurrent execution

Comparison
- OCR and TBB flow graph require specifying the task graph explicitly
- OpenMP task dependences are similar: they are inferred from a valid ordering
Task concurrency

Problem

- Modern architectures are very parallel, may have multiple hetero computing nodes/resources, and each node/resource may support more threads than are available in each low-level task
- Computing and memory components may be distributed enough that data movement latencies may become a bottleneck

Solution

- Support task concurrency across nodes and within nodes
- Enable overlapping of communication and computation
- Platform-tuned primitives offer best performance

Comparison

- OpenMP: While task concurrency within a node is possible, it requires expertise
- OpenMP doesn't yet support offloading to multiple node types
- OpenMP 4.5 enables async tasks and data movement
Dynamic scheduling

Problem
- It's increasingly difficult to create a portable and effective static schedule, given variances and uncertainties from DVFS, network congestion, multiple memory kinds, unpredictable load imbalance, and the need for fault tolerance.

Solution
- Dynamically schedule tasks and data movement. Data movement among MPI ranks and to/from various memory kinds is all integrated into a unified framework.
- Localize the scope of dependence analysis and scheduling to make it efficient.
- Can use abstract interface or exert control. Open-sourced code with pluggable architecture enables customization.
- An optional hierarchy for the scope of binding and scheduling, that can help facilitate localization and fault tolerance.
Uniform hetero interface

Problem
- App devs don't care about where execution happens; don't force them to do the binding of work to platform resources
- Heterogeneous platforms may have many computing and memory kinds

Solution
- App devs map work and data movement to an abstract queue; a tuner has the freedom to bind/retarget that to any subset of resources for the sake of performance

Comparison
- OpenMP 4.5’s auto-scheduler does not span multiple target domains; either users must manually bind tasks to a given target, or offloading must happen at a lower level of nesting, where actions are no longer truly asynchronous, because dependences are enforced only at the same nesting level
Declarative memory interface

Problem

- The number of kinds of memory are growing: DDR, HBM, 3D XPoint, etc.
- Users may need to learn many technologies and APIs in order to perform enumeration, allocation and access
- There are many opportunities for optimization, including (sub-NUMA-cluster) affinity, placement, layout, but there are few ways to manage these now
- Some scheme are not interoperable, requiring app devs to allocate memory using special allocators

Solution

- Provide a declarative interface that's associated with caller or callee operands and/or memory buffers
- Let tuners easily modify these to explore a design space
- Let more properties be added over time in ways that don’t affect the developer's semantics
- Users may use their own allocators for “host” memory, and that can get wrapped with “buffers” to manage properties and to manage allocation of instances in other domains. This facilitates interoperability with MPI, IO and complex user memory management schemes
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

*Trademarks may be claimed as the property of others
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
Making memory management easy

- Natural abstractions for programmers
  - Single unified address space
  - Refer to contiguous blocks or multi-dimensional tiles

- Infrastructure enables ease of use
  - Translates addresses across domains at invocation for heap arguments
  - Wrap memory that user allocates – enables interoperability for IO, MPI
    - Could do this implicitly upon first use, if necessary
  - Instantiate in selected domains – enables efficiency memory usage
  - Manage affinitization, if the user requests it – efficiency on sockets, sub-NUMA clusters

- What would make this harder
  - Requiring that the framework owns all memory allocation
  - Complicated addressing and translation, different types for each domain
Buffer and operand properties

- Let application developers communicate what they know
  - Size, shape, stride, etc. of operands
  - Latency and bandwidth sensitivity
  - Access patterns: read only, write combining partial results, atomic
  - Persistent storage

- Let tuners augment that collection of properties
  - Use a particular kind of memory, e.g. MCDRAM
  - Data layout

- Easy on ramp
  - Don’t have to specify much to get started; pay as you go scheme

- Give anyone visibility and control
  - See what happens semi-automatically, fix it if you don’t like it
Ease of use: Declarative approach to allocation

- Where – instances
- What – memory kind
- Which – affinity
- Whether – pinned

Alternatives to our approach

- Learn new low-level APIs
- Remote invocation of specialized actions
- Complicated set up
- Ramp on target-specific details
Ease of use: Background data transfers

- What to move specified with a descriptor
- Dependences for data xfer implied by operands
- Data xfer depends on producers, triggers consumers
- Can be to remote node or to self, using DMA

Program order

\[ x = f(A) \]
\[ z = f(B) \]
\[ DDR(x) \rightarrow MCDRAM(y) \]
\[ w = f(y) \]

Execution order

\[ x = f(A) \]
\[ DDR(x) \rightarrow MCDRAM(y) \]
\[ z = f(B) \]
\[ w = f(y) \]

High bandwidth

MCDRAM

Large capacity

DDR
Will it scale efficiently?

- We’ll find out!

- Separation of enqueuing and execution
  - Perform dependence analysis, binding, scheduling ahead of time, on distinct resources

- Decentralized
  - Each MPI rank manages its own work
  - Distribute work to local queues, so dependence enforcement scope is limited

- Other sources of efficiency
  - All actions (compute, data transfer, sync) are asynchronous
  - Transitive dependences are avoided – potential OpenMP problem
  - Dependence enforcement can be point to point or among groups
  - Address translation happens at enqueuing time, not at execution
Call to action

- Consider whether other approaches are easy enough
  - Expose platform features using natural interfaces
  - Abstract the complexity, while retaining full transparency and control

- Try out hStreams on your codes
  - See whether MPI + hStreams is a natural fit
Supplemental material

- Additional performance evaluation
- Other features
- Task efficiency
- 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

Simulia 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
HPE Apollo 6000 Benchmark Systems

Dave Mullally, HPE
Collected by Sharon Shaw, HPE

High Density
HPE Apollo 6000 Power Shelf

Pooled Power Efficiency
HPE Apollo 6000 Compute Nodes
- Node: XL250a Gen9
- Processor Family: Intel® Xeon® Processor E5 v3
- Processor Speed: 2.5 GHz
- Processor Model: E5-2680 v3
- Proc/Node: 2
- Cores/Node: 24
- Memory: 128 GB DDR4 2133 1DPC
- L3 Cache: 30 MB shared between 12 cores
- Operating System: RHEL 6.6
- Coprocessors: Xeon Phi™ 7120p Coprocessor
- Number of Nodes: 4
- Interconnect: InfiniBand FDR

Front Serviceability
HPE Apollo a6000 Chassis

Rack Level Manageability
HPE Advanced Power Manager

Argonne Leadership Computing Facility, June 1, 2016
Intel © 2016
Performance Benefit of HPE XL250a using Xeon Phi

Benefit of Xeon Phi coprocessors with Abaqus/Standard Benchmarks

### Relative Performance

- **1 Node**:
  - S4B 5.2M DOF: 1.01x
  - S4B 5.2M DOF mp_host_split=2: 1.10x
  - S2B* 1.8M DOF: 1.34x
  - S2B* 1.8M DOF mp_host_split=2: 1.32x

- **4 Nodes**: 1.11x

### Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XL250a: E5-2680 v3 (2.5 GHz, 24 Cores)</td>
<td>2 XEON PHI 7120P</td>
</tr>
</tbody>
</table>
Performance/Token Benefit of HPE XL250a using Xeon Phi

Benefit of Xeon Phi coprocessors/Token with Abaqus/Standard Benchmarks

<table>
<thead>
<tr>
<th>Node</th>
<th>1 Node</th>
<th>4 Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg Performance Improvement</td>
<td>17.7%</td>
</tr>
<tr>
<td></td>
<td>Avg Performance Improvement/Token</td>
<td>15.7%</td>
</tr>
<tr>
<td></td>
<td>Relative Performance/Token</td>
<td>1.08x</td>
</tr>
<tr>
<td>1</td>
<td>0.98x</td>
<td>1</td>
</tr>
<tr>
<td>182 Tokens</td>
<td>188 Tokens</td>
<td>182 Tokens</td>
</tr>
<tr>
<td>S4B 5.2M DOF</td>
<td>S4B 5.2M DOF</td>
<td>S2B* 1.8M DOF</td>
</tr>
<tr>
<td>mp_host_split=2</td>
<td>mp_host_split=2</td>
<td>mp_host_split=2</td>
</tr>
<tr>
<td>182 Tokens</td>
<td>188 Tokens</td>
<td>182 Tokens</td>
</tr>
<tr>
<td>S4E* 16.7M DOF</td>
<td>S4E* 16.7M DOF</td>
<td></td>
</tr>
</tbody>
</table>

*Proposed Benchmarks

**XL250a:** 2 E5-2680 v3 (2.5 GHz, 24 Cores)

**XL250a:** 2 Xeon Phi 7120P (2.5 GHz 24 Cores)
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

*Trademarks may be claimed as the property of others
Make managing dependences among tasks easy

- **Natural abstraction is a heap operand**
  - Single unified address space for distributed computation
  - Allow references at different granularities

- **Infrastructure enables ease of use**
  - Track dependences at the granularity of compute or transfer operands
    - Dependences incrementally maintained as granularities change – looks like ~O(1)
  - Dependence graph never materialized, dependences based on objects not producers

- **What would make it harder**
  - Require users to convert all code into a graph representation to get started
  - Tracking dependences only at the allocation granularity
Make language interfaces and provisioning easy

- **Natural language and provisioning**
  - Some shops prohibit use of C++
  - Some shops allow use of only a single compiler that changes every 2 years

- **Infrastructure matches those user expectations**
  - C ABI, on which Fortran (Siemens PLM Nastran) and Python (PyMIC) have been layered
  - Allow users to provide C++ template-based interfaces, e.g. for remote invocation
  - Dynamically-linked library interface

- **What would make it harder**
  - Require use of C++ templates – TBB
  - Require users to get latest compiler – standard language interfaces, e.g. OpenMP 4
  - Recompile your app against the library - Legion
## Concurrent small tasks on partitions improve efficiency

*Increasing concurrency within domain*

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

Credit for data collection: Efe Guney