HiHAT: A New Way Forward for Hierarchical Heterogeneous Asynchronous Tasking
A retargetable interface for tasking & language runtimes

CJ Newburn, Principal HPC Architect
Distributed and Heterogeneous Programming in C/C++ Workshop, Toronto, May 16, 2017
MOTIVATION FOR RETARGETABLE INFRASTRUCTURE

Build it right, for lasting impact

- “We haven’t agreed on a user-level interface for tasking”
  - It’s unlikely that we will anytime soon. But we can agree on infrastructure.
- “We’re done with science experiments and want something we can use”
  - Gather usage models and requirements → architect a durable, robust solution
- “We don’t want another academic endeavor”
  - Create something driven and supported by vendors
WHAT’S IN IT FOR THE COMMUNITY?

Seeking a win-win-win

- App developers
  - Common SW architecture across multiple targets, with multiple data layouts
- Runtime developers
  - Better performance and robustness, less effort
  - Tasking runtimes and language runtimes that don’t necessarily use tasking
- Vendors
  - Expose HW features to a larger market, i.e. SW that spans multiple targets
WHERE DO WE START?

Bottom up

• In order to make life easier for the largest set of people, start at the bottom
  • Extremely performant APIs that span targets, plus an easier-to-use set of APIs
  • Strive for inclusiveness and extensibility
• Progress from low-level plumbing to runtime building blocks
  • Building blocks or anything higher are useless until you have underlying plumbing layer
  • Foster collaboration once we have something to work off of
• Make it easy to create new or improved user interfaces
  • But don’t start by convincing anything to quit using their and use a new user interface
WHAT WOULD IT NEED FOR BROAD ADOPTION?

Top down and bottom up, like a hi-hat cymbal

- Tailor scope to cover what’s in common; keep where people innovate out of scope
- Has to meet all provisioning constraints - see list below
- Has to be performant and robust and extensible - see design below
- Has to be the easiest way to get what people want - incremental, meeting needs
- Has to be driven by vendors, who are incentivized to be successful
  - Interest from AMD, ARM, Cray, IBM, Intel, NVIDIA
  - Hosted by Wilf Pinfold, Modelado.org, a neutral party funded by vendors and others
IMPLEMENTATION LAYER INTEREST

Some part of each institution has expressed technical interest, not necessarily business commitment.

- Argobots: Halim Amer, ANL
- Qthreads, NoRMa: Stephen Olivier, Sandia
- UCX/UCS: Pasha Sharmis, ARM (remote)
- SYCL/ComputeCPP: Michael Wong, Codeplay, Khronos, HSA (remote)
LANGUAGE OR TASKING FRAMEWORKS

Some part of each institution has expressed technical interest, not necessarily business commitment.

- C++ (CodePlay, IBM) Michael Wong
- Charm++ (UIUC) Ronak Buch, (Charmworks) Phil Miller
- Darma (Sandia) Janine Bennett
- Exa-Tensor (ORNL) Wayne Joubert
- Fortran (IBM)
- Gridtools (CSCS, Titech) Mauro Bianco
- HAGGLE (PNNL/HIVE) Antonino Tomeo
- HPX (CSCS)
- Kokkos, Task-DAG (SNL) Carter Edwards
- Legion (Stanford/NV) Mike Bauer
- OmpSs (BSC) Jesus Labarta
- Realm (Stanford/NV) Sean Treichler
- OCR (Intel, Rice, GA Tech) Vincent Cave
- PaRSEC (UTK) George Bosilca
- Raja (LLNL) Rich Hornung
- Rambutan, UPC++ (LBL) Cy Chan
- R-Stream (Reservoir Labs) Rich Lethin
- SyCL (CodePlay) Michael Wong
- SWIFT (Durham) Matthieu Schaller
- TensorRT (NVIDIA) Dilip Sequeira
- VMD (UIUC) John Stone

Bold = had material for Mini-Summit
WHAT IS HiHAT?
4 faces

Community-wide requirements gathering effort
  • Leads to solid architecture that’s durable, extensible, robust

Architect user layer and common layer API and implementation

Implementation beneath user and common layers
  • Vendor-maintained and user-supplemented

Integrate with OSS project: pluggable, conformant building blocks
  • Built on user and common layers
  • Language and tasking runtimes are built out of these
HiHAT CLIENTS
Start incrementally, build from there

- HiHAT’s primary clients are **existing** language and tasking runtimes (e.g. C++, Kokkos)
  - Already have an interface to 1 or more targets, want a better interface/implementation
- HiHAT’s secondary clients are runtimes that are **being designed** (e.g. HIVE/HAGGLE)
  - Open to influencing their design to be amenable to integration with/building on HiHAT

- HiHAT provides a **target-neutral** interface, used **whole or in part** by clients
  - Identify what’s of greatest value, e.g. for future proofing, ease, robustness
  - **Incrementally adopt** those parts of HiHAT, and build up and out from there
- HiHAT does not have a near-term goal of providing a complete user-facing runtime
HiHAT’S OSS BUILDING BLOCKS

Accelerating communal progress

- The HiHAT interfaces will define types and a machine model
  - This HiHAT definition defines an architecture to which clients built on it conform
- Clients sharing a need for common functionality contribute/use building blocks
  - This would be an open source project
  - Examples: schedulers, cost models, visualization, dependence analysis, transformation
  - Suppose 4 orgs have needs in common; each can contribute a couple, consume others
  - Contributors can share tests (unit, functional, longevity)
  - Consumers can customize and contribute back, beef up testing, etc.
HiHAT’S IMPLEMENTATION LAYER
Vendor-driven performance and completeness

• HiHAT enables vendors/implementation providers to plug in functionality from below
  • Functionality behind the HiHAT APIs
  • Vendors may have the strongest incentive to provide access to their platform features
  • Others may offer alternate/improvements implementations
STATIC OR DYNAMIC
Both need a common infrastructure

- Commonalities between static and dynamic
  - Same actions: cost models, binding, ordering, allocation, data copies
  - Either can be greedy, look at a limited scope, or buffer to maximize the scope
- Similar principles, slightly different approach
  - Static vs. dynamic: make decisions, either record them for later or execute immediately
- The same (library) primitives are applicable to both
  - In order to be applicable to dynamic runtimes, can’t be only a compiler
  - But library interfaces need to be vetted to address compiler effectiveness and efficiency
- Inter-mix with compilation
Applications and frameworks: compilers, runtime libraries, ...

Services
- Monitoring
- Viz

Transformations
- Aggregate
- Decompose
- Specialize

Functional building blocks
- Comms costs
- Compute costs
- Sched

Applications and frameworks: compilers, runtime libraries, ...

Shared, contributed utilities
- Target agnostic
- Target specific

Open source
- Many frameworks

Layer

Target agnostic
- More portable, tunable
- Future proofing

Target specific
- Many hats
- Accelerate coding
- Share technology
- Increase robustness

Value

Targets
- HiHAT User Layer
- HiHAT Thin Common Layer
- Compute/Threading RTs
- Data Movt Planning
- Data Movt
- Sync
- Enum

https://wiki.modelado.org/HiHAT_SW_Stack
VALUE
Providing the easiest path toward what you already want

• Common interface to vendor-specific features
  • Modular design, separation of concerns
    • What’s above user/common layer can use target-agnostic heuristics on target-specific parameters
  • Future proofing
    • Retargetable across vendors, implementations, generations
    • Underlying implementations can chase changes and improvements

• Performance and robustness
  • Vendors are incentivized to provide 1st-class support; others can supplement
# STATUS

Gradual start, but on firm footing

- **Gather**
  - Usage models, applications, user requirements - modestly-broad participation, need more

- **Architect**
  - Design principles - good progress, much more to come; need more concrete requirements

- **Implement**
  - Implementation plan - POC this summer, anticipating partial implementation end of 2017

- **Integrate**
  - Proof of concept → early adopters → broaden

<table>
<thead>
<tr>
<th>Opt Timing</th>
<th>2016</th>
<th>2017</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>Community input</td>
<td>Community review</td>
<td>Community feedback</td>
</tr>
<tr>
<td>Architect</td>
<td>Design principles</td>
<td>API proposal</td>
<td>Refined API</td>
</tr>
<tr>
<td>Implement</td>
<td>Proof of concept</td>
<td>Initial subset</td>
<td>More complete</td>
</tr>
<tr>
<td>Integrate</td>
<td>Proof of concept</td>
<td>First/partial clients</td>
<td>Broader, more complete</td>
</tr>
</tbody>
</table>
MOMENTUM
Building interest, firming up investment

- Modelado.org - neutral zone, posting of usages, requirements, apps; monthly mtgs
- Active bottom-up discussions with vendors → initial POC with glue code
- Existence proofs and past learning: hetero streams, REALM, ~OCR, CodePlay
- ECP - ATDM funding, PathForward2 SW, CORAL/APEX/ECP app owners from ORNL, ANL, LBL, LANL
- PASC - interest from Platform for Advanced Scientific Computing, Switzerland
- Workshop on Exascale SW Technologies (WEST) - panelist, Feb. 22
- Workshop at GPU Tech Conference - May 9 am, share progress, deepen investment
- Talk @ IWOCL workshop, Distributed and Hetero Programming for C/C++17, May 16
- Performance portability workshop - August 21
SCOPE OF FUNCTIONALITY

- Cover key platform-specific actions and services
  - Data movement - target-optimized copies, DMA, networking
  - Data management - support many kinds and layers of memory, specialized pools
  - Coordination - completion events, locks, queues, collectives, iterative patterns
  - Compute - target-optimized tasks, including remote invocation
  - Enumeration - kinds and number of resources (compute, memory), topologies
  - Feedback - profiling, load
  - Tools - tracing, callbacks, pausing, ... {debugging}
Layer

Many frameworks

Shared, contributed utilities

Target agnostic
Target specific

Values

Many hats

Accelerate coding
Share technology
Increase robustness

Increase robustness
More portable, tunable

Future proofing

https://wiki.modelado.org/HiHAT_SW_Stack
## TABULATED RESULTS FROM MINI-SUMMIT

Strong interest, modestly amenable; focus on data first

<table>
<thead>
<tr>
<th>Type of functionality</th>
<th>Level of interest</th>
<th>Amenability to refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H  M  L</td>
<td>H  M  L</td>
</tr>
<tr>
<td>Data movement - target-optimized copies, DMA, networking</td>
<td>14 0 1</td>
<td>7 3 1</td>
</tr>
<tr>
<td>Data management - kinds and layers of memory, specialized pools</td>
<td>10 3 2</td>
<td>7 2 2</td>
</tr>
<tr>
<td>Coordination - completion events, locks, queues, collectives, iteration</td>
<td>8 7 0</td>
<td>5 4 1</td>
</tr>
<tr>
<td>Compute - local or remote invocation</td>
<td>7 1 4</td>
<td>4 4 3</td>
</tr>
<tr>
<td>Enumeration - kinds/# of resources, topologies</td>
<td>11 3 1</td>
<td>4 3 2</td>
</tr>
<tr>
<td>Feedback - profiling, utilization</td>
<td>6 5 2</td>
<td>4 5 1</td>
</tr>
<tr>
<td>Tools - tracing, callbacks, pausing, debugging</td>
<td>3 10 2</td>
<td>2 5 2</td>
</tr>
</tbody>
</table>
CALL TO ACTION
Join the momentum, keep us grounded in reality

• Join the community in providing input
  • Provisioning constraints, usage models, user stories @ hihat.modelado.org
  • Leverage real-world experience to influence API design
• Consider reviewing, contributing code
  • Implementation layer for new targets
  • Building blocks
• Detailed compare/contrast between HiHAT and OpenCL