HiHAT: A Way Forward to Perf Portability with Retargetable Infrastructure

CJ Newburn, Principal HPC Architect for Compute SW @ NVIDIA
OUTLINE

• Perspective: performance portability
• Challenges: More heterogeneity in HW platforms, SW interfaces
• Solutions: Common retargetable infrastructure - hierarchical hetero async tasking
HETEROGENEITY AND RETARGETABILITY

• Heterogeneity within a platform
  • Increasing specialization
  • Host, accelerators; kinds, layers and locations of memory; interconnect

• Retargetability across platforms
  • One software architecture, many targets
  • And of course we want...
PERFORMANCE PORTABILITY DEFINITION

• “Same code” + different architectures → efficient performance
PERFORMANCE PORTABILITY CONTRADICTIONS

• “Same code” + different architectures $\rightarrow$ efficient performance

• Contradictions - first set
  • But I like my language! The other guy’s language gives horrible performance!
  • But I need a special data layout for each target!
  • But I have a favorite user-level interface. Don’t take that away from me!
User interfaces

Target directives, languages, DSLs

HiHAT is at the boundary

target agnostic
target specific
PERFORMANCE PORTABILITY PARTIAL SOLUTIONS

• “Same code” + different architectures → efficient performance

• Potential solutions - first set
  • **Language**: Target-specific task implementations where needed
  • **Data layout**: Task implementations tailored for data layout, scheduler can choose to re-layout data off of the critical path
  • **User-level interface**: Layer client user-facing runtimes on top of retargetable interface
PORTABILITY IS IN THE EYE OF THE BEHOLDER

- Task: High-level language, with directives or DSL or even assembly instructions
PORTABILITY IS IN THE EYE OF THE BEHOLDER

- Pluggable implementations
  - Task: High-level language, with directives or DSL or even assembly instructions
  - Best way for a given platform: target-specific APIs and implementations
PORTABILITY IS IN THE EYE OF THE BEHOLDER

- Sequence of target-agnostic primitives
  - Invoke, manage data, move data, coordinate, enumerate
- Pluggable implementations
  - Task: High-level language, with directives or DSL or even assembly instructions
  - Best way for a given platform: target-specific APIs and implementations
PORTABILITY IS IN THE EYE OF THE BEHOLDER

• Scheduler - binding and ordering, based on cost model
  • Select target, implementation, layout, add actions as needed
  • Invoke primitives where and when most appropriate
• Sequence of target-agnostic primitives
  • Invoke, manage data, move data, coordinate, enumerate
• Pluggable implementations
  • Task: High-level language, with directives or DSL or even assembly instructions
  • Best way for a given platform: target-specific APIs and implementations
COMMON RETARGETABLE SW ARCHITECTURE

Implementation

Primitives

Open source building blocks

Glue code

Cost models

Scheduler

Target informed

Cross-platform target agnostic

Negligible overhead

Target-specific layout

Target-specific coding

Data management

Data movement

Task invocation

Target-specific dispatch

Enumeration

Data management

Data movement

Task invocation

Target-specific

Scheduler
MOTIVATIONS FOR A SCHEDULER

• Lack of predictability
  • Where data comes from, in memory hierarchy or across network
  • When computation will finish: complex algorithms, load imbalance, DVFS

• Growing complexity
  • Too many factors at play to settle on a single portable static scheduler
  • Too much diversity in increasingly-heterogeneous platforms

• Going asynchronous
  • Break out of bulk synchronous, move to point-point
  • Dynamic management of resources
PROVIDING ACCESS TO PERFORMANCE
Meeting our customers where they are, offering a path forward

- % lines of code gains, ROI

- Exposing maximal parallelism
- Extreme scaling
- Tuning for the target platform
- Tailored abstractions

- Limited effort
- Traditional language interfaces
Applications and frameworks: compilers, runtime libraries, ...

Common plumbing layer: HiHAT

Services
- Moni-
toring
- Viz

Transformations
- Aggre-
gate
- Decom-
pose
- Special-
ize

Functional building blocks
- Comms costs
- Compute costs
- Sched

App developers code

Tuners configure

Experts implement

Target 1

Target 2

Target 3

Target 4

https://wiki.modelado.org/Heterogeneous_Hierarchical_Asynchronous_Tasking
HIHAT: APIS FOR RETARGETABILITY

- Plug in target-specific implementations from below
- Implement data management, data movement, invocation, coordination, querying
- User: ease of use via abstraction
- Common: minimal overhead
LANGUAGE OR TASKING FRAMEWORKS

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

- C++ (CodePlay, IBM) Michael Wong
- Chapel (Cray), Brad Chamerlain
- Charm++ (UIUC) Ronak Buch, (Charmworks) Phil Miller
- Darma (Sandia) Janine Bennett
- Exa-Tensor (ORNL) Wayne Joubert
- Gridtools (CSCS, Titech) Mauro Bianco
- HAGGLE (PNNL/HIVE) Antonino Tomeo
- 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
- StarPU (INRIA) Samuel Thibault
- SyCL (CodePlay) Michael Wong
- SWIFT (Durham) Matthieu Schaller
- TensorRT (NVIDIA) Dilip Sequeira
- VMD (UIUC) John Stone

Bold = shared material on mapping to HiHAT
### TABULATED RESULTS

**Strong interest, modestly amenable; progress; next**

<table>
<thead>
<tr>
<th>Type of functionality</th>
<th>Level of interest</th>
<th>Amenability to refactoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data movement - target-optimized copies, DMA, networking</strong></td>
<td>H 1 1</td>
<td>H 7 5 1</td>
</tr>
<tr>
<td><strong>Data management - kinds and layers of memory, specialized pools</strong></td>
<td>11 4 2</td>
<td>7 4 2</td>
</tr>
<tr>
<td><strong>Coordination - completion events, locks, queues, collectives, iteration</strong></td>
<td>9 8 0</td>
<td>6 5 1</td>
</tr>
<tr>
<td><strong>Compute - local or remote invocation</strong></td>
<td>7 3 4</td>
<td>4 5 4</td>
</tr>
<tr>
<td><strong>Enumeration - kinds/# of resources, topologies</strong></td>
<td>11 5 1</td>
<td>4 4 3</td>
</tr>
<tr>
<td><strong>Feedback - profiling, utilization</strong></td>
<td>6 7 2</td>
<td>4 7 1</td>
</tr>
<tr>
<td><strong>Tools - tracing, callbacks, pausing, debugging</strong></td>
<td>3 12 2</td>
<td>2 7 2</td>
</tr>
</tbody>
</table>
ADOPTION

• Meet requirements
  • Provisioning: C ABI, library, interoperable, profiling
  • Performance: enables access to perf features, low overhead → supports fine granularity
  • Incremental, easy on ramp

• Open architecture
  • Be a provider for tasking and language runtimes and frameworks
  • Plug in implementations from below, from vendors or third parties
  • Share building blocks, e.g. cost models, schedulers

• Easiest and best solution
SO MANY FRAMEWORKS, SO LITTLE TIME

Design
Porting and performance tuning
Validation

Common plumbing layer: HiHAT

x86  POWER  Tegra ARM  GPUs

CoE Perf Portability Workshop 8/22/17
PROTOTYPE INFRASTRUCTURE CAPABILITIES

The basics are already working

• Current test platform: 2 CPU sockets + 2 GPUs in one node
• Data movement
  • User Layer: <dst, src, size> using logical handles for addressing
  • Common Layer: use specialized flavors
  • Set up comms, establish visibility as needed
• Data management
  • User Layer: Allocate or register, and create address-memory resource association
    • Also support tagging to clean up a set of allocations/wraps at once
  • Common Layer: No tagging
• Invocation
  • Register target-specific implementations, invocation with closure
• Microbenchmarks show overheads are within measurement noise

CoE Perf Portability Workshop 8/22/17
MOLECULAR ORBITALS (MO) APPLICATION

• Compute wavefunction amplitudes on a grid for visualization
  • Evaluate linear combination of Gaussian contractions (polynomials) at each grid point, function of distance from atoms
• Algorithm made arithmetic bound via fast on-chip memory systems
• Three different algorithms for different memory structures:
  • GPU constant memory
  • Shared memory tiling
  • L1 global memory cache
• Representative of a variety of other grid-oriented algorithms, stencils
• Use of special GPU hardware features, APIs helped drive completeness of HiHAT proof-of-concept implementation already at an early stage
MOLECULAR ORBITALS PERFORMANCE

- Performance of MO algorithm on HiHAT User Layer PoC implementation closely tracks CUDA performance.
- Spans x86, POWER and Tegra ARM CPUs

### HiHAT API gains for Molecular Orbitals application

<table>
<thead>
<tr>
<th>Molecular Orbital Algorithm, Mem Kind</th>
<th>Speedup vs. ShMem</th>
<th>HiHAT API gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 L1CachedGlbMem HiHAT ConstMem HiHAT</td>
<td>1.000x 1.088x 1.472x</td>
<td>1.028x 1.025x 1.031x</td>
</tr>
<tr>
<td>PWR L1CachedGlbMem HiHAT ConstMem HiHAT</td>
<td>1.000x 1.116x 1.534x</td>
<td>0.999x 1.001x 0.983x</td>
</tr>
<tr>
<td>ARM L1CachedGlbMem HiHAT ConstMem HiHAT NoPin-SharedMem HiHAT NoPin-L1CachedGlbMem HiHAT NoPin-ConstMem HiHAT</td>
<td>1.000x 1.094x 1.059x 2.349x 2.561x 2.562x</td>
<td>- - 0.995x 0.984x 0.998x</td>
</tr>
</tbody>
</table>
PORTABILITY ON MO
Mapping between CUDA and HiHAT

- Time to port MO: 90 minutes
- HiHAT has fewer unique APIS (6 vs. 10)
- HiHAT has fewer static API calls (30 vs. 38)
- Accelerate optimization space exploration
- Also enhance coding productivity
TAKE-AWAYS

- Portability comes at the scheduling layer, on top of target-agnostic primitives
- Dynamic scheduling may have the most promising path to portability and scaling
- Necessary conditions: meet requirements; be pluggable; open source approach; be the easiest path to performance, generality and robustness
- HiHAT prototype looks promising as a retargetable infrastructure