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 $\rightarrow$ 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

HiHAT APIs

Data management

Data movement

Task invocation

Target-specific dispatch

Scheduler

Cost models

Target informed

Cross-platform target agnostic

Negligible overhead

Target-specific layout

Target-specific coding

Cost models

Scheduler

Enumeration

Target informed

Cross-platform target agnostic

Negligible overhead

Target-specific layout

Target-specific coding
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

New/revised code

Old code

CoE Perf Portability Workshop 8/22/17
Applications and frameworks: compilers, runtime libraries, ...

Open source:
- Services
  - Monitoring
  - Viz
- Transformations
  - Aggregate
  - Decompose
  - Specialize
- Functional building blocks
  - Comms costs
  - Compute costs
  - Sched

Common plumbing layer: HiHAT

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></td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>Data movement - target-optimized copies, DMA, networking</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>Data management - kinds and layers of memory, specialized pools</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>Coordination - completion events, locks, queues, collectives, iteration</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Compute - local or remote invocation</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Enumeration - kinds/# of resources, topologies</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Feedback - profiling, utilization</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Tools - tracing, callbacks, pausing, debugging</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>
ADOPTION

• Meet requirements
  • Provisioning: C ABI, library, interoperable, profiling
  • Performance: enables access to perf features, low overhead $\rightarrow$ supports fine granularity
  • Productivity: 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
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
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 + SharedMem HiHAT</td>
<td>1.000x</td>
<td>1.028x</td>
</tr>
<tr>
<td>x86 + L1CachedGblMem HiHAT</td>
<td>1.088x</td>
<td>1.025x</td>
</tr>
<tr>
<td>x86 + ConstMem HiHAT</td>
<td>1.472x</td>
<td>1.031x</td>
</tr>
<tr>
<td>PWR + SharedMem HiHAT</td>
<td>1.000x</td>
<td>0.999x</td>
</tr>
<tr>
<td>PWR + L1CachedGblMem HiHAT</td>
<td>1.116x</td>
<td>1.001x</td>
</tr>
<tr>
<td>PWR + ConstMem HiHAT</td>
<td>1.534x</td>
<td>0.983x</td>
</tr>
<tr>
<td>ARM + SharedMem HiHAT</td>
<td>1.000x</td>
<td>-</td>
</tr>
<tr>
<td>ARM + L1CachedGblMem HiHAT</td>
<td>1.094x</td>
<td>-</td>
</tr>
<tr>
<td>ARM + ConstMem HiHAT</td>
<td>1.059x</td>
<td>-</td>
</tr>
<tr>
<td>ARM + NoPin-SharedMem HiHAT</td>
<td>2.349x</td>
<td>0.995x</td>
</tr>
<tr>
<td>ARM + NoPin-L1CachedGblMem HiHAT</td>
<td>2.561x</td>
<td>0.984x</td>
</tr>
<tr>
<td>ARM + NoPin-ConstMem HiHAT</td>
<td>2.562x</td>
<td>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

<table>
<thead>
<tr>
<th>Category</th>
<th>Original CUDA</th>
<th>Ported to HiHAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke</td>
<td>&lt;&lt;&lt;&gt;&gt;&gt;</td>
<td>hhuInvoke()</td>
</tr>
<tr>
<td>Data mvt</td>
<td>cudaMemcpy()</td>
<td>hhuCopy()</td>
</tr>
<tr>
<td></td>
<td>cudaMemcpyToSymbol()</td>
<td>hhuCopy()</td>
</tr>
<tr>
<td>Configuration</td>
<td>cudaSetDeviceFlags()</td>
<td>(config)</td>
</tr>
<tr>
<td></td>
<td>cudaFuncSetCacheConfig()</td>
<td>(config)</td>
</tr>
<tr>
<td>Data mgt, minimal</td>
<td>cudaMalloc()</td>
<td>hhuAlloc()</td>
</tr>
<tr>
<td></td>
<td>cudaMallocHost()</td>
<td>hhuAlloc()</td>
</tr>
<tr>
<td></td>
<td>cudaHostAlloc()</td>
<td>hhuAlloc()</td>
</tr>
<tr>
<td></td>
<td>[free]</td>
<td>hhuClean()</td>
</tr>
<tr>
<td></td>
<td>[symbols]</td>
<td>hhuRegMem()</td>
</tr>
<tr>
<td>Data mgt, eliminatable</td>
<td>cudaFree()</td>
<td>hhuFree()</td>
</tr>
<tr>
<td></td>
<td>cudaFreeHost()</td>
<td>hhuFree()</td>
</tr>
<tr>
<td></td>
<td>[symbols]</td>
<td>hhuDeregMem()</td>
</tr>
<tr>
<td>Coordination</td>
<td>-</td>
<td>hhuSyncAll()</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>static</td>
<td>14+3+3+9+9+0</td>
<td>9+3+0+16+16+1</td>
</tr>
<tr>
<td>static min’l</td>
<td>14+3+3+9+9+0</td>
<td>9+3+0+17+0+1</td>
</tr>
<tr>
<td>unique</td>
<td>2+1+2+5+0+0</td>
<td>1+1+0+2+2+1</td>
</tr>
<tr>
<td>unique min’l</td>
<td>2+1+2+5+0+0</td>
<td>1+1+0+3+0+1</td>
</tr>
</tbody>
</table>

CoE Perf Portability Workshop 8/22/17
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