RECENT GPU PROGRAMMING IMPROVEMENTS IN CHAPEL

Engin Kayraklioglu, Andy Stone, Daniel Fedorin

CHIUW 2023 - June 2\textsuperscript{nd}, 2023
GPU PROGRAMMING IN CHAPEL
KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism:** What tasks should run simultaneously?
2. **locality:** Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory

Locale 0

Locale 1

Locale 2

Locale 3

- **CPU Core**
- **Memory**
KEY CONCERNS FOR SCALABLE PARALLEL COMPUTING

1. **parallelism**: What tasks should run simultaneously?

2. **locality**: Where should tasks run? Where should data be allocated?
   - complicating matters, compute nodes now often have GPUs with their own processors and memory
   - we represent these as *sub-locales* in Chapel
PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

locale 0

GPU 0

GPU 1

var x = 10;

on here.gpus[0] {
    var A = [1, 2, 3, 4, 5, ...];
    A += 1;
}

writeln(x);
PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

```javascript
var x = 10;

on here.gpus[0] {
    var A = [1, 2, 3, 4, 5, ...];
    A += 1;
}

writeln(x);
```
PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

```plaintext
var x = 10;

coforall g in here.gpus do on g {
    var A = [1, 2, 3, 4, 5, ...];
    A += 1;
}

writeln(x);
```
PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

```
var x = 10;
coforall l in locales do on l {
  coforall g in here.gpus do on g {
    var A = [1, 2, 3, 4, 5, ...];
    A += 1;
  }
}
writeln(x);
```
PARALLELISM AND LOCALITY IN THE CONTEXT OF GPUS

```plaintext
var x = 10;
coforall l in locales do on { l {
    cobegin {
        coforall g in here.gpus do on g {
            var A = [1, 2, 3, 4, 5, ...];
            A += 1;
        }
    }
}
writeln(x);
```
RECENT IMPROVEMENTS
FEATURES

CHIUW '22

- Fundamentals working

CHIUW '23

- GPU Module
  - assertOnGpu
  - setBlockSize
  - createSharedArray
  - atomic operations

- GpuDiagnostics Module
  - Count number of launches
  - Verbose kernel launches
  - Initial profiler support

Future

- Portable features for forall/foreach
- foreach intents
GPU MODULE

- Has basic introspection/debugging support
  - `assertOnGpu`
    - Compilation fails if the loop is not GPU-eligible, execution fails if it is not run on a GPU sublocale
  - `gpuWrite/gpuWriteIn`
  - `gpuClock/gpuClocksPerSec`
- A new standard module to support fundamental GPU operations
  - `setBlockSize`
  - `createSharedArray`
  - `syncThreads`

**Next steps:**
- We expect most of the functionality here to be implemented in existing Chapel features
  - i.e., `writeln` should replace `gpuWriteIn`
- Designing new features for `foreach/forall` loops for fundamental GPU operations in a portable way
  - i.e., `syncThreads` can be implemented with some form of a barrier
  - i.e., the user should be able to query the GPU thread ID or Chapel task ID from inside a `forall` loop
GPU DIAGNOSTICS MODULE

- Basic diagnostics support modeled after CommDiagnostics module
  - Currently only supports counting/reporting kernel launches

```chapel
startVerboseGpu(); // print a message out everytime a kernel is launched
startGpuDiagnostics(); // count kernel launches
on here.gpus[0] {
    foreach i in 1..10 do // gpu-eligible operations
}
stopGpuDiagnostics();
stopVerboseGpu();
writeln(getGpuDiagnostics());
```

Output:

```
0 (gpu 0): foo.chpl:4: kernel launch (block size: 512x1x1)  # via startVerboseGpuO
(kernel_launch: 1)  # via getGpuDiagnosticsO
```
PROFILER SUPPORT

Background:
• Debugging and profiling GPU kernels are typically more difficult than CPU applications
  – I/O support is typically poor, execution model is less intuitive, esoteric challenges
• NVIDIA has numerous profilers, where NSight Compute is used for profiling kernel performance
  – While using profilers for Chapel in general is not very straightforward, focusing on kernels is easier
• Out-of-the-box: NSight Compute was able to show line-by-line hardware counters when '-g' was used
  – However, '--fast -g' thwarted assembler optimizations → reduced kernel performance → less valuable profiling

This Effort:
• Added the '--gpu-ptxas-enforce-optimizations' flag to ensure that assembler optimizations are enabled

Impact:
• Significant help while trying to understand performance of compiler-generated kernels
  – Kernel performance is virtually unaffected
  – Profiler shows line-by-line information accurately
• Can compare performance behavior of a reference version against the Chapel version
PORTABILITY

CHIUW '22
- Monolithic runtime
  - CUDA Driver API wrapper

CHIUW '23
- Modular runtime
- AMD support
- "cpu-as-device" mode

Future
- Intel support
AMD SUPPORT

- AMD GPUs are now supported
  - 1.30: single-locale
  - 1.31 pre-release: multilocal

- Features/correctness
  - Mostly on-par with NVIDIA
  - Some 64-bit math functions aren't supported

- Performance
  - Only initial studies so far
  - (more on this later)
CPU-AS-DEVICE SUPPORT

- **Supports GPU locale model w/o GPUs**
  - GPU functionality is diverted to CPU

- **Loops are outlined for kernel generation**
  - however, they are not compiled/launched
  - loop always runs for equivalent correctness

- **Use-cases**
  - `assertOnGpu` to confirm GPU offload
  - `GpuDiagnostics` to count/report "launches"
• Intel GPU support is the next target
  • Modular runtime should help
  • Compilation is a bigger question
    – GPU transformations should be the same across vendors
    – LLVM generates the code for us;
    – but no Intel GPU target, yet
    – Intel's LLVM fork can target Intel GPUs
    – Potential solution: rely on system LLVM that's Intel-GPU-enabled

• Implementation has not started, yet
PERFORMANCE

CHIUW '22
- Initial studies

CHIUW '23
- Significant performance improvements
  - Faster kernel launch
  - Faster kernel execution
  - "array-on-device" mode
  - New benchmarks
  - Initial AMD experiments

Future
- "array-on-device"
- Compiler optimizations
GENERAL PERFORMANCE IMPROVEMENTS

- **Eager binary loading**
  - GPU binary is loaded at application launch
  - ~300x faster kernel launch times

- **Loop-invariant code motion before GPU pass**
  - Loops are optimized before turning into kernels
  - Faster kernel execution

- **Application-level optimizations with ChOp***

<table>
<thead>
<tr>
<th>N</th>
<th>Interop (s)</th>
<th>Native (s)</th>
<th>Off by</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.30</td>
<td>0.36</td>
<td>19%</td>
</tr>
<tr>
<td>16</td>
<td>1.79</td>
<td>2.06</td>
<td>15%</td>
</tr>
<tr>
<td>17</td>
<td>12.47</td>
<td>14.76</td>
<td>18%</td>
</tr>
<tr>
<td>18</td>
<td>94.94</td>
<td>110.98</td>
<td>17%</td>
</tr>
</tbody>
</table>

N-Queens Performance with ChOp

*(1x NVIDIA P100)*

*Tiago Carneiro, Nouredine Melab, Jan Gmys, Guillaume Helbecque, et al. — INRIA Lille, France; Imec, Belgium; University of Mons, Belgium; et al.*
**WORK IN PROGRESS: AMD PERFORMANCE**

**HPCC-Stream:** Similar to NVIDIA
- Maybe a little lower with smaller data
  - potentially due to higher kernel launch costs

**ChOp:** Performance drops with larger data
- Investigation is pending

### N-Queens Performance with ChOp
(1x AMD MI100)

<table>
<thead>
<tr>
<th>N</th>
<th>Interop (s)</th>
<th>Native (s)</th>
<th>Off by</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.40</td>
<td>0.55</td>
<td>36%</td>
</tr>
<tr>
<td>16</td>
<td>1.14</td>
<td>2.18</td>
<td>91%</td>
</tr>
<tr>
<td>17</td>
<td>6.36</td>
<td>13.28</td>
<td>209%</td>
</tr>
<tr>
<td>18</td>
<td>47.04</td>
<td>115.51</td>
<td>246%</td>
</tr>
</tbody>
</table>
FUTURE WORK

Time (s) (RTX A2000)

<table>
<thead>
<tr>
<th>Unified Memory</th>
<th>Array on Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12</td>
<td>18.16</td>
</tr>
<tr>
<td>0.038</td>
<td>0.018</td>
</tr>
</tbody>
</table>

Faster initialization on GPU

How memory is allocated

<table>
<thead>
<tr>
<th>Unified Memory</th>
<th>Array on Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>host</td>
</tr>
<tr>
<td>data</td>
<td>host</td>
</tr>
</tbody>
</table>

Faster data movement

```plaintext
var CpuArr: [1..n] int;

on here.gpus[0] {
    var GpuArr: [1..n] int;
    GpuArr = CpuArr;
    CpuArr = GpuArr;
}
```
SUMMARY

Status so far:
• Can target NVIDIA and AMD GPUs in single- and multilocal
• The performance has been significantly improved, but there's more room
• Fundamental GPU operations are supported via a standard module
• Diagnostics and introspection tools can help performance analysis and optimization

Next steps:
• Target Intel GPUs
• More performance improvements
• New features to support portable programming between GPU- and vector-based execution
THANK YOU

engin@hpe.com