MiniAero

Wonchan Lee
MiniAero

- Fluid dynamics mini-app that uses the Runge-Kutta forth-order time marching scheme
- Ported to both Legion C++ API (Sandia) and Regent (Stanford)
- Initial versions do not scale up well:

  Each node is becoming less efficient as the node count is growing

Weak-scaling graph with 256K cells per node

http://legion.stanford.edu
Sources of Inefficiency

- Having a **single control task** launch tasks on all nodes
  - Adds delay between tasks being launched

![Diagram showing sources of inefficiency]

- Analysis cost
- Launch delay

**node 0**

**node 1**

**node 2**

---

December 7, 2015

http://legion.stanford.edu
Sources of Inefficiency

- Having a **single control task** launch tasks on all nodes
  - Adds delay between tasks being launched
Sources of Inefficiency

Using different partitions of the same region can effectively serialize tasks

```plaintext
var r = region(...)
var pD = partition(disjoint,r,...)
var pA = partition(aliased,r,...)

for i = 0,2: F(pD[i]) -- writes pD
for i = 0,2: G(pA[i]) -- reads pA
```
Sources of Inefficiency

- Using different partitions of the same region can effectively serialize tasks

```plaintext
var r = region(...)  
var pD = partition(disjoint,r, ...)  
var pA = partition(aliased,r, ...)

for i = 0,2: F(pD[i]) -- writes pD  
for i = 0,2: G(pA[i]) -- reads pA
```

- To start G, runtime waits for all updates of F on r to be visible to pA[i]
- The runtime can minimize the underlying data movement between instances but cannot avoid the serialization
Sources of Inefficiency

Using different partitions of the same region can effectively serialize tasks

- Tasks in MiniAero read cells from other blocks on the border

4x2x1 mesh

<table>
<thead>
<tr>
<th>cell</th>
<th>cell</th>
<th>cell</th>
<th>cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>cell</td>
<td>cell</td>
<td>cell</td>
</tr>
</tbody>
</table>

Disjoint partition for private cells

<table>
<thead>
<tr>
<th>cell</th>
<th>cell</th>
<th>cell</th>
<th>cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>cell</td>
<td>cell</td>
<td>cell</td>
</tr>
</tbody>
</table>

Aliased partition for ghost cells

<table>
<thead>
<tr>
<th>cell</th>
<th>cell</th>
<th>cell</th>
<th>cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>cell</td>
<td>cell</td>
<td>cell</td>
</tr>
</tbody>
</table>

http://legion.stanford.edu
Sources of Inefficiency

- Using different partitions of the same region can effectively serialize tasks
  - Tasks in MiniAero read cells from other blocks on the border
  - Updating private cells makes the next task accessing ghost cells wait

...  
```
var cells = region(...)  
var pcells = partition(disjoint, cells, ...)  
var pghost = partition(aliased, cells, ...)  
...  
for i = 0,4:  
  compute_limiter(pcells[i], pghost[i], pfaces[i])  -- writes pcells[i]
for i = 0,4:  
  compute_face_flux(pcells[i], pghost[i], pfaces[i])  -- reads pghosts[i]
...  ```
Sources of Inefficiency

- Using different partitions of the same region can effectively serialize tasks
  - Tasks in MiniAero read cells from other blocks on the border
  - Updating private cells makes the next task accessing ghost cells wait

Legion MiniAero on one node

the last **computer_limiter** task blocked the first **compute_face_flux** task
Solution: SPMD, Legion Style

- Have **multiple control tasks** launch tasks on their own node
  - Lower latency from analysis cost

```
node 0
node 1
node 2
```

```
app thread  
runtime thread 
app thread  
runtime thread 
app thread  
runtime thread 
app thread  
runtime thread 
```

**smaller analysis cost**
Solution: SPMD, Legion Style

- Have **multiple control tasks** launch tasks on their own node
- Lower latency from analysis cost

SPMD-ified Legion MiniAero on two nodes

runtime barely runs analysis once control tasks launch all tasks
Solution: SPMD, Legion Style

- Have multiple control tasks launch tasks on their own node
- Have tasks locally **share their updates** with each other

Slower tasks no longer stop the other tasks from communicating with each other.
Solution: SPMD, Legion Style

- Have multiple control tasks launch tasks on their own node
- Have tasks locally share their updates with each other
- Not necessarily manual
  - Planned automatic SPMD transformation in the Regent compiler
  - Planned automatic SPMD optimization in the Legion runtime
Solution: SPMD, Legion Style

- Have multiple control tasks launch tasks on their own node
- Have tasks locally share their updates with each other
- Not necessarily manual
  - Planned automatic SPMD transformation in the Regent compiler
  - Planned automatic SPMD optimization in the Legion runtime
- Manual SPMD-ification is always an option
  - Can be done relatively easily for simple cases
  - Regent provides a cleaner syntax for hand-written SPMD-style code
  - Good exercise to understand transformations that the future compiler and runtime will provide
    ➔ Let’s talk about how I’ve transformed MiniAero
MiniAero in Legion’s SPMD Style

Makes **ghost regions** be their own **root regions**

aliased partition for ghost cells

```
  cell  cell  cell  cell
  cell  cell  cell  cell
```

explicit regions for ghost cells

```
  ghost region 0
  cell          cell
  cell          cell
  cell          cell

  ghost region 1
  cell          cell          cell
  cell          cell          cell
  cell          cell          cell

  ghost region 2
  cell          cell
  cell          cell
  cell          cell

  ghost region 3
  cell          cell
  cell          cell
  cell          cell
```
MiniAero in Legion’s SPMD Style

- Makes ghost regions be separate regions explicitly
- Tell runtime to run **simultaneously** a list of control tasks

```latex
\textbf{must\_epoch}
\textbf{spmd\_control}(p\text{cells}[0], r\text{ghost0}, \ldots)
\textbf{spmd\_control}(p\text{cells}[1], r\text{ghost1}, \ldots)
\textbf{spmd\_control}(p\text{cells}[2], r\text{ghost2}, \ldots)
\textbf{spmd\_control}(p\text{cells}[3], r\text{ghost3}, \ldots)
\textbf{end}
```

explicit ghost regions

- owned cells
- ghost cells
MiniAero in Legion’s SPMD Style

- Makes ghost regions be separate regions explicitly
- Tell runtime to run simultaneously a list of control tasks
- Control tasks should copy changes from their owned cells to ghost regions
MiniAero in Legion’s SPMD Style

- Makes ghost regions be separate regions explicitly
- Tell runtime to run simultaneously a list of control tasks
- Control tasks should copy changes from their owned cells to ghost regions

```plaintext
must_epoch
spmd_control(pcells[0], rghost0, rghost1, rghost2, ...)
spm_control(pcells[1], rghost1, rghost0, rghost3, ...)
spm_control(pcells[2], rghost2, rghost0, rghost3, ...)
spm_control(pcells[3], rghost3, rghost1, rghost2, ...)
end
```

should see one instance of the same region
⇒ simultaneous coherence!
MiniAero in Legion’s SPMD Style

- Makes ghost regions be separate regions explicitly
- Tell runtime to run simultaneously a list of control tasks
- Control tasks should copy changes from their owned cells to ghost regions

```plaintext
task spmd_control(rcells  : region(...), rghost  : region(...),
  rneighbor1 : region(...), rneighbor2 : region(...),
  ...)
  where reads exclusive(cells), reads simultaneous(rghost)
  reads writes simultaneous(rneighbor1, rneighbor2)
  ...
end
```
MiniAero in Legion’s SPMD Style

- Makes ghost regions be separate regions explicitly
- Tell runtime to run simultaneously a list of control tasks
- Control tasks should copy changes from their owned cells to ghost regions

```plaintext
task spmd_control(rcells : region(...), rghost : region(...),
                   rneighbor1 : region(...), rneighbor2 : region(...),
                   ...)
where reads exclusive(cells), reads simultaneous(rghost)
reads writes simultaneous(rneighbor1, rneighbor2)
...
end
```

tell runtime to map these regions simultaneously
Pushing Updates to Ghost Regions

Tasks and copies must be \textit{synchronized} to avoid races

```plaintext

task spmd_control(...) where ...
doi ...
compute_limiter(rcells, rghost, rfaces)
copy(rcells, rneighbor1) -- data race!

doi ...
end
```

- node 0
  - compute_limiter
  - ghost 1
  - rneighbor1 (= ghost 2) still being used \(\rightarrow\) data race!

- node 1
  - ghost 2
  - compute_limiter

[Diagram showing task and data flow with nodes and ghost regions]
Phase Barriers for Synchronization

- Legion provides phase barriers, a light-weight mechanism to synchronize between operations
  - Phase barriers are not a global barrier, unlike MPI barriers
  - Each barrier can make progress at a different rate
Synchronizing Tasks and Copies

Each control task is responsible for synchronizing its subtasks

```plaintext
\text{task} \ \text{spmd\_control}(\ldots) \ \text{where} \ \ldots \\
do \\
\ldots \\
\text{compute\_limiter}(r\text{cells}, \ r\text{ghost}, \ r\text{faces}) \ \text{arrives}(\text{pb\_g\_free}) \\
\text{copy}(r\text{cells}, \ r\text{neighbor}1) \ \text{awaits}(\text{pb\_n1\_free}) \\
\ldots \ \text{end}
```

![Diagram illustrating the synchronization process]

- **Await rneighbor1 (= ghost 2) becoming free**
- **Copy**
- **Arrive on the barrier to trigger the copy**
Synchronizing Tasks and Copies

Each control task is responsible for synchronizing its subtasks

```lua
  task spmd_control(...) where ...
  do
    ...
    compute_limiter(rcells, rghost, rfaces) arrives(pb_g_free)
    copy(rcells, rneighbor1) awaits(pb_n1_free) arrives(pb_n1_ready)
    compute_face_flux(rcells, rghost, rfaces) awaits(pb_g_ready)
  ...
end
```

![Diagram showing synchronization and copies between nodes and threads.](http://legion.stanford.edu)
Relaxing Simultaneous Constraints

Simultaneous coherence enforces that all tasks use the same region instance.

- Acquire and release operations relax that constraint.
  - Useful when the task needs to copy the instance somewhere else (e.g., GPU framebuffer memory).

Graphical representation:
- Node 0
- Acquire
- Copy
- Compute_face_flux
- Release
- Ghost 1

Legend:
- App thread on CPU
- App thread on GPU
- Runtime thread
- Region instances on system memory
- Region instances on framebuffer

- Once ghost 1 is acquired, runtime can make copies.
- Back to the simultaneous instance once ghost 1 is released.

December 7, 2015
http://legion.stanford.edu
Programming Experience

- Started with the initial C++ port
  - Regent support for SPMD-style programs wasn’t ready yet
  - First correct version in 2 weeks
  - A few more weeks to optimize and tune
  - Would have been quicker with Regent

- Legion Spy was helpful in tracking down synchronization bugs
  - Currently, this is the price of managing tasks manually
  - Event graphs show which tasks are depending on which phases of barriers
  - Physical dependence analysis shows some missing dependencies if tasks are synchronized incorrectly
Preliminary Performance Study

- Weak scaling experiments
  - 256K cells per node
  - Certainty Cluster

- Two target versions
  - Initial version without SPMD
  - Manually SPMD-ified version (one control task per processor)
Weak Scaling Graph

- SPMD version scales much better than the original

- Dropped by 97%
- Dropped by 30%
Measuring Runtime Overhead

- Commenting out task bodies
  - Runtime still issues all tasks with necessary copies
  - SPMD-style version has stable overhead (0.6s – 0.8s per timestep)
Plans

- SPMD-ification in Regent
  - Will be faster due to better leaf tasks in Regent
  - Manual SPMD-ification support is now available
  - Automatic SPMD-ification will become available soon

Comparing between various SPMD configurations

- We can have M control tasks each of which manages N processors
  - More control tasks better amortizes analysis cost but has more overhead due to partitioning
  - Fewer control tasks can reduce communication overhead but be less adaptable to load imbalance
- We’ll explore with Regent’s automatic SPMD-ification
Concluding Remarks

- Legion’s SPMD-style is a practical way to achieve high scalability
  - MiniAero shows steady weak-scaling performance up to 32 nodes

- SPMD style is not too difficult
  - Requires only the control task to be rewritten
  - Does require explicit programmer-managed synchronization between control tasks

- SPMD-style programming will become easier
  - Cleaner syntax for in Regent
  - Planned automatic SPMD transformation in the Regent compiler
  - Planned automatic SPMD optimization in the Legion runtime