Integrating External Resources into Legion

Zhihao Jia
Motivation

- Legion assumes a “closed world”

But, applications need to interact with external data
  - e.g., files, checkpoints, databases

Challenge
  - external interactions are expensive: the data is often huge
Motivation

Original solution

- Ask users to manage external I/O at application level
- Access external data within Legion tasks

Performance issues

- Block computing threads, hard to hide I/O latency, hard to control resource utilization

Correctness issues

- Manually control external data consistency at application level
Approach

Define semantics for external resources in Legion

- Correctness: Legion guarantees consistency and preservation of dependencies
- Performance: runtime automatically performs external I/O optimizations

Idea: Integrate external resources by mapping them to regions => attach operation
Attach Operation

Attach external resource to a region

- Normal files, formatted files (HDF5), opaque data structures

```
PhysicalRegion attach_hdf(
    const char *filename,
    LogicalRegion lr,
    const std::map<FieldID, const char*> &fieldmap,
    AccessMode mode);
```

IndexSpace $\Leftrightarrow$ HDF DataSpace

Fields $\Leftrightarrow$
HDF Datasets
Attach Operation

Semantics

- Invalidate existing physical instance of \textit{lr}
- Maps \textit{lr} to a new physical instance that represents external data (no external I/O)
Attach Operation

Semantics

- Invalidate existing physical instance of $lr$
- Maps $lr$ to a new physical instance that represents external data (no external I/O)
Attach Operation

- Attached region accessed using *simultaneous coherence*
  - Different tasks access the region simultaneously
  - Requires that all tasks must use the *only valid* physical instance

*Copy restriction*
- Simultaneous coherence implies tasks cannot create local copies
- May result in inefficient memory accesses

To address inefficiency => acquire/release
Acquire/Release

- Mechanism to notify Legion runtime when it is safe to allow local copies

Acquire removes copy restriction
- Can create a copy in any memory

Release restores copy restriction
- Invalidates all existing local copies
- Flushes dirty data back to external resource
Acquire/Release Example

Application

Legion Runtime

Local Inst

Task

Node 1

Local Inst

Task

Node 2

Ext Inst

External Resource

Acquire
Region r

Flush
More on Attach Semantics

- Attach to in-memory opaque data structures
  - External data comes from other applications
  - Legion may not understand the data format

- User could attach opaque data structures to regions

Field holds pointers/refs to the opaque data structures
Custom SerDes

- Bit-wise copy no longer work
- Legion requires custom SerDes methods for fields requiring non-trivial copies
- Users define a class with SerDes methods

```cpp
class SerDesObject {
    static size_t serialized_size(const FIELD_TYPE& val);
    static size_t serialize(const FIELD_TYPE& val, void *buffer);
    static size_t deserialize(FIELD_TYPE& val, const void *buffer);
    static void destroy(FIELD_TYPE& val);
}
```

- SerDes registration is similar to reduction operation

  ```cpp
  runtime->register_custom_serdes_op<SerDesObject>(serdes_id);
  ```

- Specify SerDes methods when allocating fields

  ```cpp
  allocate_field(sizeof(FIELD_TYPE), field_id, serdes_id);
  ```
Optimization: Deferred Execution

- Legion runtime manages/reschedules external I/O
  - maximize resource utilization
  - overlap external I/O with computation

Matrix multiplication
- Load large input matrices from files on disk
Optimization: Reduce Data Transfer

Region $r$

External Inst → Copy → Local Inst

Node 1

Memory Runtime

Node 2

Memory Runtime

Task 1

Task 2

Task 3

up-to-date w/ better performance
Optimization: Reduce Data Transfer

- Distributed graph rendering
  - Each node renders a portion of the screen
  - Communication: copy physical objects between nodes

![Graph showing execution time per iteration vs. total nodes (8 CPUs per node)]
Optimization: Write-After-Read

Region r

Application

Node 1

External Inst

Copy

Flush

Task (RW)

Node 2

Local Inst

Acquire

Release

Task (RW)

Task (RO)

Application

External Inst

Local Inst

Task (RW)

Task (RO)

Copy

Flush

December 7, 2015

http://legion.stanford.edu
Optimization: Write-After-Read

Database benchmark
- Perform read queries and read/write queries on external databases on disk

![Chart showing execution time for different read query task percentages. The chart compares Baseline and Legion+ER.](chart.png)
S3D

- A production combustion simulation
- Checkpoint after fixed time steps
- Legion implementation is 7X faster than Fortran

![Checkpoints Overhead Chart]

<table>
<thead>
<tr>
<th>Save Interval (in Timesteps)</th>
<th>200</th>
<th>100</th>
<th>50</th>
<th>76.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>7.1</td>
<td>2.7</td>
<td>4.6</td>
<td>2.9</td>
</tr>
<tr>
<td>Legion</td>
<td>2.1</td>
<td>2.1</td>
<td>11.5</td>
<td>3.3</td>
</tr>
<tr>
<td>Legion+ER</td>
<td>2.1</td>
<td>2.1</td>
<td>11.5</td>
<td>3.3</td>
</tr>
</tbody>
</table>
Questions