New Mapping Interface

Mike Bauer
NVIDIA Research
Why Have A Mapping Interface?

- Scheduling is hard
- Lots of runtimes have heuristics
  - What do you do when they are wrong?
- Legion mapping interface exposes all these decisions
  - Customize decisions/heuristics for applications + machines

Legion Program → Legion → Legion Mapper
Old Mapping Interface

Let’s be honest: the current interface is not clean

There are good reasons for this:
- No one had ever designed a dynamic one before
- We had no idea what we really wanted

Result: evolutionary interface
- No coherence in the design
New Mapping Interface

We now have some experience writing mappers

We know (mostly) what we want

Time for a new interface with a coherent design
Mapper Call Format

```cpp
struct MapTaskInput {
    ...
};
struct MapTaskOutput {
    ...
};
virtual void map_task(const Task &task,
                       const MapTaskInput &input,
                       MapTaskOutput &output) = 0;
```

- Most mapper calls have three arguments
  - Reference to the operation (task, copy, inline mapping, etc)
  - Input argument struct
  - Output argument struct

- Clear delineation of inputs and outputs

- Extensible: can easily add new parameters
Physical Instances

- Old mapper based around memories
  - `std::vector<Memory>` target_ranking;
  - This was alright before logical regions had fields

- New mapper based around physical instances
  - `std::vector<PhysicalInstance>` chosen_instances;
  - Give explicit names to physical instances
  - No more guessing what the runtime does

- Consequences:
  - New way of managing creation/deletion of physical instances
  - New way of mapping tasks
Instance Management

- Mappers can hold references to instances
  - Have names for instances
  - Prevent de-allocation

- Mappers can request instances be reclaimed
  - For when memories are full

- Mapper call to rank instances that are ready for deletion

Today’s Policy

- Pending or Executing Task
- Valid Data
- Mapper Reference

Physical Instance
Specifying Data Layout

- Currently: Legion is minimally aware of layout

- Blocking factor: describe density of fields

- Two problems:
  - Insufficient for describing all interesting data layouts
  - Not captured as properties of task variants

- Need more expressiveness
Layout Constraint Language

A small set-constraint language

Can describe the following:
- Dimension ordering
- Field ordering
- Sub-dimensions for tiling
- Alignment
- Field offsets
- Memory kinds
- ...

AOS, C-order (left)

SOA, Fortran-order (right)

2-D Slices (bottom)
Layout Constraints Example

Task
2-D stencil
Two fields: A + B
Fields are 8 bytes

Machine
CPU w/ AVX
16 KB L1 cache

Constraints
split(X, 32)
split(Y, 32)
inner_X < inner_Y
inner_Y < fields
A < B
fields < Z
Z < outer_X
outer_X < outer_Y
align(inner_X, 32-bytes)

inner_X = 32
inner_Y = 32

32 * 32 * 8 bytes * 2 fields = 16 KB

Why bother?
Legion DMA code automates data transformation

December 7, 2015
http://legion.stanford.edu
Satisfying Region Requirements

- Can satisfy region requirements with multiple instances
  - Today: only one instance per region requirement
  - Reduce the number of unnecessary copies

Today:
Make new instance
Issue copies

Tomorrow:
Re-use instances
No copies

Only works with SOA layouts
Satisfying Region Requirements

- Map multiple region requirements to the same instance
  - Useful for pointers of type \( \text{ptr}(r1+r2+...) \)
  - No need for conditional statements on pointer dereferences

Solution: put them in a big instance that is the union of \( r1+r2 \)

\[ *(\text{ptr}(r1+r2)) \]

How do we guarantee correctness?
Task Variant Registration

- Need set constraint language for tasks too
  - Co-location constraints (regions mapped to the same inst)
  - Processor ISA (x86, Power, ARM, PTX, …)
  - Resource constrains (cache sizes, registers, …)

- New task variant registration API
  - Specify all constraints on task variant
  - Specify layout constraints on all region requirements

- Support for dynamic task variant registration
  - Anticipating DLLs and JIT
Mapping Tasks

- Mapping tasks is now a little different

- Mapper picks:
  - Processor on which to run
  - Instance(s) for all region requirements

Mapper picks task variant

Correctness is guaranteed

Runtime verifies all constraints are satisfied

Mapper is lazy: doesn’t pick

Runtime picks variant with most closely satisfied constraints

What if it can’t find one?

http://legion.stanford.edu
Task Generators

- What if we can’t find a satisfactory variant?
  - Today: mapping failure -> retry
  - Better answer: make the right variant

Task Variant Generators:
- A function invoked by the runtime to generate a task variant
- One registered for each kind of task (with optional static data)

A Generic Interface for Dynamic Compilation with Any Compiler

http://legion.stanford.edu
Dealing with Close Operations

Close operations move data between partitions
- Automatically inserted by the runtime where needed
- Normally transparent

Except: \texttt{rank\_copy\_targets}
- The most misunderstood and feared mapper call
- Create physical instance(s) for close operations
- Now gone!

Replaced by \texttt{map\_close(…)}
Composite Instances

- Perform close without building a big instance

- Create composite instance
  - Snapshot of region tree
  - Capture existing instances

- Issue minimal copies from existing instances
  - Legion automatically performs intersection tests
  - Memoizes results

Can use this today:
return ‘true’ from rank_copy_targets
Manipulating Dependence Graphs

- Legion builds a dependence graph internally
  - Discovers all the parallelism possible

- How much is too much?
  - It depends
  - Make it a mapper decision

- Allow mapper to manipulate the dependence graph
Fusing Tasks

- **Idea:** let the mapper fuse tasks together

- **Fusion:** run tasks consecutively
  - Leverage locality
  - Amortize analysis costs

- **Specialize by the kind of machine and graph shape**

---

**Fuse Dependent Tasks for Locality**

```
  Task
  ↓
  Task
```

**Fuse Independent Tasks for Locality**

```
  Task
  ↓
  Task
```

**Fuse Independent Tasks for Reduced Analysis**

```
  Task
  ↓
  Task
  ↓
  Task
```

---

December 7, 2015

http://legion.stanford.edu
Replicating Tasks

- Replicate tasks to reduce communication
  - (or parallelize it)

- Works both within nodes and across nodes

  From S3D (intra-node)
  - calc_ydiff_flux
  - calc_species
  - calc_tau

  From Multigrid (inter-node)
  - Coarse stages of v-cycle

CPU
- calc_ydiff_flux
- calc_species
- calc_tau

GPU
- calc_ydiff_flux
- calc_species
- calc_tau

Node
- Node
- Node
- Node

http://legion.stanford.edu
A New Default Mapper

A new mapper interface requires...

... a new default mapper implementation

- Better heuristics for management of data
- Better load balancing algorithms
- More generalized algorithms for constructing mappers
Bishop Mapping Language

C++ mapping interface is still verbose

Bishop: a language for mapping

Prototype version part of tomorrow’s exercise
Open Mapper Questions

The mapper interface is still open for modifications

- What are the best ways to manage deferred execution?
- How do we compose multiple mappers?
- What are the best practices for mapper data structures?
- What are good abstractions for mapper construction?