Legion Overview: 
What’s New in 2015?

Alex Aiken
Bootcamp Logistics

- **Monday**
  - Parking
  - Lunch
  - Dinner

- **Tuesday**
  - Programming exercise
  - Bring your laptops!
Bootcamp Focus

- Writing Legion programs
  - Different from the academic papers
  - Cover many pragmatic, usability aspects

Today
- Brief overview of the programming model
- Deeper dives on major changes in 2015
- Overview of a familiar application (MiniAero)
- Debugging & profiling

Tomorrow: Programming exercise
Programming System Goals

High Performance
We must be fast

Performance Portability
Across many kinds of machines and over many generations

Programmability
Sequential semantics, parallel execution
Can We Fulfill These Goals Today?

Yes ... at great cost:

Who will schedule the graph?
(High Performance)

Who will re-schedule the graph for every new machine?
(Performance Portability)

Who is responsible for generating the graph?
(Programmability)

Today: programmer’s responsibility

Tomorrow: programming system’s responsibility

Task graph for one time step on one node...
... of a mini-app

December 7, 2015

http://legion.stanford.edu
Legion Overview

- A programming model for heterogeneous, distributed machines

- Heterogeneous
  - Mixed CPUs and GPUs

- Distributed
  - Large spread, and variability, of communication latencies
  - Caches, RAM, NUMA, network, …
Philosophy

- Designed to be a real programming system
- Good abstractions, clear semantics
- But can also “open the hood”
  - Ways to drop down to lower levels of abstraction
  - Within the programming model
Legion: Tasks & Regions

- A *task* is the unit of parallel execution

- Task arguments are *regions*
  - Collections
  - Rows are an *index space*
  - Columns are *fields*

- Tasks declare how they use their regions

```plaintext
task saxpy(is : ispace(int1d), x,y: region(is, float), a: float )
where reads(x, y), writes(y)
```
Example Task

task saxpy(is : ispace(int1d), x: region(is, float), y: region(is, float), a: float)

where

reads(x, y), writes(y)

doi

for i in is do

  y[i] += a*x[i]

end

end
Regions

- Regions can be *partitioned* into *subregions*

- Partitioning is a primitive operation
  - Supports describing arbitrary subsets of a region
Partitioning
Partitioning

\[
[ P, S ] = \text{partition}(ps\_map, N)
\]
Partitioning

\[
\begin{align*}
[ P, S ] &= \text{partition}(\text{ps\_map}, N) \\
\text{private} &= \text{partition}(\text{private\_map}, P) \\
\text{shared} &= \text{partition}(\text{shared\_map}, S) \\
\text{ghost} &= \text{partition}(\text{ghost\_map}, S)
\end{align*}
\]
Summary: Regions

- Regions have
  - Entries (rows)
  - Fields (columns)

- Regions can be
  - Partitioned by rows
  - Sliced by fields
Tasks

- Tasks can call *subtasks*
  - Sequential semantics, implicit parallelism
  - If tasks do not *interfere*, can be executed in parallel

```
task foo(x,y,z: region(...))
where reads(x,y,z), writes(x,y,z) do
  bar(y,x)
  bar(x,y)
  bar(x,z)
  bar(z,y)
end

task bar(r,s: region(...)) where reads(r), writes(s)
```
Deferred Execution

```plaintext
task foo(x,y,z: region(...))
where reads(x,y,z),writes(x,y,z) do
  bar(y,x)
  bar(x,y)
  bar(x,z)
  bar(z,y)
end
task bar(r,s: region(...)) where reads(r), writes(s)
```

Legion Runtime

```
  bar(y,x)
  bar(x,y)  bar(x,z)
  bar(z,y)
```
Mapping Interface

Application selects:
- Where tasks run
- Where regions are placed

Mapping computed dynamically

Decouple correctness from performance
Who will schedule the graph?  
*(High Performance)*

Who will re-schedule the graph for every new machine?  
*(Performance Portability)*

Who is responsible for generating the graph?  
*(Programmability)*
More on Permissions

Tasks declare *permissions* on regions

\[ \text{task bar}(r: \text{region}(...)) \text{ where reads}(r) \]

\[ \text{task bar}(r: \text{region}(...)) \text{ where writes}(r) \]

\[ \text{task bar}(r: \text{region}(...)) \text{ where reduces } + (r) \]
And Coherence

- Tasks declare coherence of regions
- With respect to sibling tasks

\[
\text{task bar}(r: \text{region}(\ldots)) \text{ where exclusive}(r)
\]

\[
\text{task bar}(r: \text{region}(\ldots)) \text{ where atomic}(r)
\]

\[
\text{task bar}(r: \text{region}(\ldots)) \text{ where simultaneous}(r)
\]
Atomic Coherence

task foo(x: region(…)) where reads(x), writes(x), exclusive(x)

do
    bar(x)
    bazz(x)
end

task bar(r: region(…)) where reads(r), writes(r), atomic(r)

task bazz(r: region(…)) where reads(r), writes(r), atomic(r)
Simultaneous Coherence

task foo(x: region(...)) where reads(x), writes(x) do
  bar(x)
  bazz(x)
end

task bar(r: region(...)) where reads(r), writes(r), simultaneous(r)

task bazz(r: region(...)) where reads(r), writes(r), simultaneous(r)
Simultaneous Coherence

- Progressive relaxation of coherence
  Exclusive > Atomic > Simultaneous

- Simultaneous coherence
  - Implies programmer involvement in managing concurrency
  - Additional primitives
    - acquire(r), release(r), phase barriers

- An example of “opening the hood”
  - Programmer takes responsibility for coordination between tasks using simultaneous coherence
S3D

- Combustion simulation, explicit method
  - Physics and complex chemistry
  - Collaboration with Jackie Chen’s group (Sandia)
  - Part of the ExaCT Center

- Structure of S3D
  - Partition volume across nodes
  - Launch one long-running task per node
    - Some private data (exclusive)
    - Some shared data (simultaneous)
    - Use acquire/release to mediate access
  - Within a node
    - Tasks launch subtasks with exclusive or atomic coherence
    - Some tasks mapped to GPU, some to CPU
Legion Heptane Performance

1.73X - 2.85X faster between 1024 and 8192 nodes
Heptane Mapping for $96^3$

- Different mapping than smaller problem sizes
  - Not enough room in 6 GB GPU framebuffer
  - OpenACC requires code changes

- Note “ragged phases”
  - Deferred execution tolerant of latency/execution variance

- Not shown: Overlap of data movement
Legion PRF Performance

- 116 species mechanism, >2X as large as heptane
- New science, never before done

![Graph showing Legion PRF Performance](http://legion.stanford.edu)
The Crux

- Crucial design decisions in a Legion program are:

- What are the regions?

- How are the regions partitioned?

- The answers drive the program’s design
Legion Overview Summary

- **The programmer**
  - Describes the structure of the program’s data
    - Regions
  - The tasks that operate on that data

- **The Legion implementation**
  - Guarantees tasks appear to execute in sequential order
    - Unless the programmer relaxes coherence
  - Ensures tasks have the correct versions of their regions
The Past Year

- The project has changed
  - Legion group has grown substantially
  - Lots of interaction with users
  - Learned a lot about Legion, including flaws!

- Mid-2015 strategic plan
  - Focus on fixing core issues
  - Even if it involves major changes
  - Will not get any easier in the future!

- Results are starting to roll out now.
Problem #1

- C++ API is verbose, a lot to learn
- Many semantic requirements are unchecked
- No help with kernel code
  - Legion is about managing data and black-box tasks
  - Doesn’t address generating efficient task code

Decision: These issues can’t and shouldn’t be addressed in the C++ API
Legion Architecture

- Application
- Regent
- DSL

High-Level Runtime

Realm: Low-Level Runtime

GasNet
Regent: A Legion Language

task saxpy(is : ispace(int1d), x: region(is, float), y: region(is, float), a: float)

where reads(x, y), writes(y)

do
   for i in is do
      y[i] += a*x[i]
   end
end
Problem #2: Partitioning

- Creation of partitions is hard to fully distribute
  - Inherent in the original design
  - Deal-breaker for some applications

- Solution
  - Design a new partitioning system
  - More concise and much higher performance
Problem #3: Mapping

- Mapping interface is at the wrong level of abstraction
  - User has to say “do exactly this”
  - Much better would be “do at least this”
  - Or “do at most this”

Solution
- A new constraint-based mapper interface
Problem #4: I/O

- Must be able to
  - Read/write files
  - Produced by other applications
  - In parallel

Solution

- A new I/O subsystem
- Understands how to interpret distributed file formats as partitioned regions
Problem #5: Breaking Changes

- More developers + more users
  - Users getting blocked by research-level software practices

- Introduce more disciplined development
  - Clean-up, rationalization of the repository
  - Investing in testing infrastructure
  - Including the mundane and the high-end
Today’s Talks

- Regent (Elliott)
- Partitioning (Sean)
- Mapping (Mike)
- I/O (Zhihao)
- Debugging & Profiling (Wonchan)

- Application walkthrough (Wonchan)
- User experiences (Galen, Steve, Hemanth, Philippe)
More To Come

- These are not the only changes/features coming

- More at the end of the day
Questions?