Legion Bootcamp:
Building Abstractions for Legion Applications

Samuel K. Gutierrez
Applied Computer Science
Los Alamos National Laboratory
Legion is designed for two classes of users: **DSL & Library Authors** and **Advanced Application Devs**.
DSL & Library Authors

Developers of high-level languages and libraries that help increase application developer productivity.
Advanced Application Devs.

Users of MPI, SHMEM, CUDA, etc. that develop their applications and re-write for new architectures.
Legion focuses on providing a common framework which can achieve portable performance across a range of architectures.
Legion focuses on providing a common framework which can achieve portable performance across a range of architectures.

Developer productivity in Legion is a second-class design constraint.
Performance & Extensibility are #1.

Developer productivity in Legion is a second-class design constraint.
Performance & Extensibility are #1.

And this is perfectly reasonable.
Many ways to increase developer productivity when targeting Legion’s C/C++ interfaces directly.
Many ways to increase developer productivity when targeting Legion’s C/C++ interfaces directly.

This talk presents a few.
Interface: Odds are you’ll be writing to the C++ interface.

C Interface – Language Devs.
C++ Interface – Application Devs.
Build Containers that encapsulate container properties and manage storage through logical regions.
Build Containers that encapsulate container properties and manage storage through logical regions.

Goal: replicate familiar **structures** & **operations** on structures.
Goal: reproduce familiar function signatures at the top level.

Goal: replicate familiar structures & operations on structures.
Ex. 1: An Array Stickman

```c
struct Array {
    IndexSpace is;
    FieldSpace fs;
    LogicalRegion lr;
    LogicalPartition lp;
    Domain lDom;
};
```
Ex. 1: An Array Stickman

```c
struct Array {
    IndexSpace is;
    FieldSpace fs;
    LogicalRegion lr;
    LogicalPartition lp;
    Domain lDom;
};
```

Conceptual Structure of the Array
Ex. 1: An Array Stickman

```c
struct Array {
    IndexSpace is;
    FieldSpace fs;
    LogicalRegion lr;
    LogicalPartition lp;
    Domain lDom;
};
```

Used Primarily for Inquiry & Task Launch
Ex. 1: An Array Stickman

```cpp
template<typename T>
void create(uint64_t length,
            Context &context,
            HighLevelRuntime *lrt);
```

Type of Array Elements

Length of Array

Legion Handles Used in `create`
**Ex. 1: An Array Stickman**

```c
void partition(uint64_t n, Context &context, HighLevelRuntime *lrt);
```

The function `partition` creates \( n \) disjoint partitions of the entire array \( A \).
Ex. 1: An Array Stickman

```c
void free(Context &ctx,
           HighLevelRuntime *lrt);
```
Ex. 1: Using the *Array* Stickman

```c
double dotprod(Array &x, Array &y, Context &context, HighLevelRuntime *lrt);
```

\[ w \cdot x \cdot y = \]

\[
\begin{array}{c}
  \text{w} \\
  \text{x} \\
  \text{y}
\end{array}
\]
Ex. 1: Using the Array Stickman

/* dotprod() (Pseudo) Code Snippet */

double dotprod(Vector &x, Vector &y, . . .) {
    IndexLauncher il(DOT_TID, x.lDom,
        TaskArgument(NULL, 0), aMap);

    Create an IndexLauncher

    Here x and y’s Launch Domains are Equivalent, so One is Chosen

}
Ex. 1: Using the Array Stickman

/* dotprod() (Pseudo) Code Snippet */
double dotprod(Vector &x, Vector &y, . . .) {

  il.add_region_requirement(
      RegionRequirement(x.lp, 0, RO, EX, x.lr)
  );
  il.add_field(0, x.fid);

  /* Similarly, add RegionRequirement for y */

  Add Region Requirements

}
Ex. 1: Using the Array Stickman

/* dotprod() (Pseudo) Code Snippet */
double dotprod(Vector &x, Vector &y, . . .) {

Execute the IndexSpace
and
Return Result to Caller

Future f = rt->exec_idx_space(ctx, il, RED_ID);
return f.get_result<double>();
}
Ex. 1: Using the Array Stickman

```c
double dotProdTask(
    const Task *task,
    const vector<PhysicalRegion> &rgns,
    Context ctx, HighLevelRuntime *lrt
){
    /* Implementation of Dot Product */
}
```

![Diagram of task execution and reduction](http://legion.stanford.edu)
Ex. 2: Sparse Matrices and CG

CGData cgData(A.nRows, ctx, lrt);

for (int64_t k = 1; k <= maxIters
    && (normr / normr0 > tolerance); ++k) {
    if (doPreconditioning) mg(A, r, z, ctx, lrt);
    else waxpby(1.0, r, 0.0, r, z, ctx, lrt);
    spmv(A, p, Ap, ctx, lrt);
    dotprod(p, Ap, pAp, ctx, lrt);
    alpha = rtz / pAp;
    waxpby(1.0, x, alpha, p, x, ctx, lrt);
    waxpby(1.0, r, -alpha, Ap, r, ctx, lrt);
    dotprod(r, r, normr, ctx, lrt);
    normr = sqrt(normr);
}

cgData.free(ctx, lrt);
Ex. 3: Multigrid

```c
if (A.mgData) {
    const int64_t nPre = A.mgData->nPresmootherSteps;
    for (int64_t i = 0; i < nPre; ++i) {
        symgs(A, x, r, ctx, lrt);
    }
    spmv(A, x, A.mgData->Axf, ctx, lrt);
    restriction(A, r, ctx, lrt);
    mg(*A.Ac, A.mgData->rc, A.mgData->xc, ctx, lrt);
    prolongation(A, x, ctx, lrt);
    const int64_t nPost = A.mgData->nPostsmootherSteps;
    for (int64_t i = 0; i < nPost; ++i) {
        symgs(A, x, r, ctx, lrt);
    }
} else symgs(A, x, r, ctx, lrt);
```
Some Code Doing This:

https://github.com/losalamos/CODY/tree/master/legion/lgncg
Help Us Help You: We’re writing a Legion debugger and need input.

Anything About: Features, Use Cases, Tricky Bugs

Specifics Please 😊
Questions?

@samuelkgutierrez

samuel@lanl.gov