Task-Based Programming with Legion

Alex Aiken
Stanford/SLAC
What is Legion?

A task-based programming model for heterogeneous, parallel, distributed machines

Designed to be
• High performance
• Performance portable
• Productive
An Example: S3D

- Simulates chemical reactions
  - DME (30 species)
  - Heptane (52 species)
  - PRF (116 species)

Two parts

- Physics
  - Nearest neighbor communication
  - Data parallel
- Chemistry
  - Local
  - Complex task parallelism
Weak Scaling: PRF

![Graph showing throughput per node compared to the number of nodes. The graph indicates scalability with arrows demonstrating 3X and 7X improvements.]
What Led to the Improvement?

• Sequential semantics

• Asynchronous tasks

• Late binding of performance decisions
  • Where tasks execute
  • Where data is placed
  • How data is partitioned
  • …
Sequential Semantics

S3D Skeleton

task top_level() {
    V = simulation volume
    P[N] = partition V
    G[N] = ghost cells of V
    repeat
        Chem(P[i]) for i = 1..N
        Phys(P[i],G[i]) for i = 1..N
    until done
}

task Chem(V) { … }
task Phys(V,G) { … }

• A sequential program
  • With a parallel execution

• Greatly simplifies debugging
  • No race conditions!

• Sequential semantics can be relaxed if desired
  • E.g., for reductions
if compression() then
    __demand(__index_launch)
    for color in is_rank do
        CalcGammaTask(lp_int_rank[color])
    end

    __demand(__index_launch)
    for color in is_rank do
        Sum3Task(lp_int_rank[color].{X=RHS_1_DX, Y=RHS_1_DY, Z=RHS_1_DZ},
                 lp_q_rank[color].{RHO_U},
                 false)
    end

    __demand(__index_launch)
    for color in is_rank do
        Sum3Task(lp_int_rank[color].{X=RHS_2_DX, Y=RHS_2_DY, Z=RHS_2_DZ},
                 lp_q_rank[color].{RHO_V},
                 false)
    end

    __demand(__index_launch)
    for color in is_rank do
        Sum3Task(lp_int_rank[color].{X=RHS_3_DX, Y=RHS_3_DY, Z=RHS_3_DZ},
                 lp_q_rank[color].{RHO_W},
                 false)
    end

end

...
The Benefits of Asynchrony

- Overlap communication and computation
- Overlap runtime analysis with the application
  - Runtime analysis is distributed SPMD fashion across nodes
- In general, also get task parallelism
Late Binding of Decisions

After
- the program is written
- the machine is selected
- the input is chosen

It is easy to
- Change the partitioning of data
- Change the assignment of tasks
  - E.g., move a task from GPU to CPU
- Change the placement of data
  - E.g., from the framebuffer to zero-copy memory
- And more …
Mapping

Task * GPU,CPU;  # tasks run on GPUs by default

Task AwaitMPITask, CalcDummyTask, HandoffToMPITask, InitPartitionsTask, InitScaleTask, InitTemperatureTask, fill_cpe, fill_lr_int, fill_masses CPU;

Region * * GPU FBMEM;  # for all GPU tasks, arguments use FBMEM as default
Region * * CPU SYSMEM;  # for CPU tasks, arguments use SYSMEM as default

Layout * * * SOA F_order;  # all regions use struct of array and Fortran order

...
The Secret Sauce

- The ability to easily change performance-relevant decisions after the program is running on a machine has been key
  - We often try a lot of different strategies!

- The biggest improvements of Legion over other approaches have not been because Legion’s implementation strategy cannot be imitated.

- The improvements were because it was more productive to experiment in Legion to find an implementation strategy that works well.
S3D: Heptane 48³

Profile from one node

Runtime analysis on CPUs

CPUs

GPU

Time
S3D: Heptane $96^3$

Profile from one node

Problem: $96^3$ points per GPU did not fit on the GPU.

Solution: Move some tasks to the CPU to reduce memory pressure.
Impact on Portability & Productivity

- Many more ports of Legion-S3D than MPI-S3D
- Titan
- Summit
- Piz Daint
- Lassen
- Cori
- Perlmutter
- Frontier

- Many more variations of Legion-S3D
  - Different boundary conditions
  - Different reactions

- Example: Simulation of PRF with 116 chemical species
  - The most complex such simulation ever done
Comparison with MPI

Legion

- Sequential semantics
- Asynchronous by default
- Strong data model
  - System understands the partitioning of data
- Late binding of performance decisions

Downside: Higher runtime overhead

MPI

- Explicit parallel programming
- Synchronous by default
- Bag-of-bits data model
- Many performance decisions baked into the code

Upside: Minimal runtime overhead
Data in Legion

- Data partitioning
- Partitioning primitives
- Examples
Partitioning

Partitioning data is a distinctive feature of distributed computing
  Or whenever there are multiple, distinct memories

How should data be partitioned?
Partitioning
Partitioning
Hierarchical Partitioning

P
  p₁  …  p₃

S
  s₁  …  s₃

N

Diagram of hierarchical partitioning.
Multiple Partitions
task distribute_charge(rpn, rsn, rgn : region(node),
    rw : region(wire))
where
  reads(rw.in_ptr, rw.out_ptr, current)
reduces +(
    rpn.charge,
    rsn.charge,
    rgn.charge)
{

Tasks are the unit of parallel execution.

Privileges declare how a task will use its region arguments.

Regions are n-dimensional tables (tensors) with typed columns (fields).
Legion Example

task distribute_charge(rpn, rsn, rgn : region(node), rw : region(wire))

where

reads(rw.{in_ptr, out_ptr, current})
reduces +(rpn.charge, rsn.charge, rgn.charge)

{...

Uses both views of the shared nodes simultaneously.
Observation: Compositionality

*Multiple partitions of the same data are needed for scalable software composition*

Consider two libraries
- Written independently
- Using different partitioning strategies
- How can they be composed?

Examples
- A simulation, a solver, and a visualization library
- A data analysis pipeline
Partitioning Operators

- Legion has a rich subsystem of partitioning primitives

- Each primitive is designed for efficient, scalable parallel implementation

- Combinations of primitives express sophisticated partitioning strategies
Partitioning by Field

PartitionByField(nodes, nodes.SorP)

Nodes

<table>
<thead>
<tr>
<th>Index</th>
<th>Voltage</th>
<th>SorP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6.2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td></td>
</tr>
</tbody>
</table>
Independent Partitions

Partitioning by field is an *independent partition*
- A partitioning that depends on no other partitions
- Another example: PartitionEqual(R,5)

Legion also has *dependent partitioning* primitives
- Compute new partitions from existing partitions
- Allows regions to be co-partitioned easily
Partition By Image

- Treat a pointer field as a function

- Construct compatible partition of destination region
  - Some elements of destination may be in more than one subregion
  - Or in no subregion

Region 1

Region 2
Partition By PreImage

- Again treat a pointer field as a function
- Construct a compatible partition of the source region
Sparse Matrix Representations

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & a & b & c \\
1 & & d & e \\
2 & f & & \\
3 & g & h & \\
\end{array}
\]
Coordinate Trees

- **Region 1**
  - Nodes: 0, 1, 3
  - Colors: Yellow
  - Letters: a, b, c

- **Region 2**
  - Nodes: 0, 1, 3
  - Colors: Green
  - Letters: d, e, f

- **Region 3**
  - Nodes: 0, 1, 3
  - Colors: Red
  - Letters: g, h

The diagram illustrates the coordinate trees with regions marked by different colors and letters.
Images and Preimages
Sparse Matrix Partitioning Level-by-Level

Partition one level first

Use images and preimages to compatibly partition the other levels
Task-Based Libraries

- Task graphs naturally compose
  - Combining two or more task graphs is a task graph

- Late binding of decisions makes interfaces flexible
  - Libraries can be parameterized in ways that are impossible in other approaches

- And we can automate the search for the best partitioning and mapping
  - For a specific machine and workload
Task-Based Libraries

Task1(Args)

Composed:

Task1(Args)
Task2(Args’)

Task2(Args’)

Composed:

Task1(Args)
Task2(Args’)

**DISTAL & SpDISTAL**

- DISTAL is a Legion system for dense tensor algebra
- SpDISTAL is a variant for sparse tensor algebra

\[
A(i, j) = B(i, k) \times C(k, j)
\]

\[
A(i, j) = \sum_{k} B(i, k) \times C(k, j)
\]

- DISTAL is a DSL for tensor algebra
  - Given an expression \( e \) in tensor algebra, generate a task-based library to compute \( e \)
  - Integrated with a compiler to generate tuned kernels
Distributed Dense Matrix Multiply

Cannon’s Algorithm (1969)

PUMMA (1994)

SUMMA (1995)

Johnson’s Algorithm (1995)

Solomonik’s Algorithm (2011)

COSMA (2019)

Schedule describes how kernel interacts with the distributed data

Data partitioning and distribution

Data partitioning and distribution
Comparison with MM Libraries

- 0.95x COSMA
- 1.25x ScaLAPACK
- 0.85x COSMA
Generalizes to All of Tensor Algebra (CPUs)

\[ A(i, j) = B(i, j, k) \cdot c(k) \]  
Nodes (CPU Cores)

\[ A(i, j, l) = B(i, j, k) \cdot C(k, l) \]  
Nodes (CPU Cores)

\[ a = B(i, j, k) \cdot C(i, j, k) \]  
Nodes (CPU Cores)

\[ A(i, l) = B(i, j, k) \cdot C(j, l) \cdot D(k, l) \]  
Nodes (CPU Cores)
Generalizes to All of Tensor Algebra (GPUs)

\[ A(i, j) = B(i, j, k) \cdot c(k) \quad \text{Nodes (GPUs)} \]

\[ a = B(i, j, k) \cdot C(i, j, k) \quad \text{Nodes (GPUs)} \]

\[ A(i, j, l) = B(i, j, k) \cdot C(k, l) \quad \text{Nodes (GPUs)} \]

\[ A(i, l) = B(i, j, k) \cdot C(j, l) \cdot D(k, l) \quad \text{Nodes (GPUs)} \]
And Sparse Tensor Algebra
FlexFlow: Deep Neural Networks

FlexFlow is a Legion library for DNN training and inference

Idea #1: Exploit Legion’s expressive data partitioning to partition tensors in DNN’s in ways that Pytorch and TensorFlow do not consider

- E.g., tensor = [image, height, width, channel]
- Standard approaches partition the image dimension

FlexFlow can partition/parallelize data/computations in many more dimensions
FlexFlow: Deep Neural Networks

Idea #2: Automate the partitioning process

- Instead of searching for a good partitioning by hand

Use the fact that program structure remains the same – only the partitioning of data changes

And do this for every layer of the network

- Allow different layers to have different partitioning strategies
FlexFlow

Data parallelism

Sample → Parameter

GPU1
GPU2
GPU3
GPU4
Results: Bert-Large

Unity is the latest version of FlexFlow …
Selected Other Legion Libraries

CuNumeric (NVIDIA)
- A open source, drop-in replacement for NumPy
- See Seshu Yamajala’s talk at 11:30 on Thursday

LegionSolvers (in progress)
- Sparse iterative distributed solvers

Distributed Sparse SciPy (in progress)
Summary

- Task-based programming systems provide a sequential programming model with implicit parallelism

  Late binding of performance decisions has proven key to achieving the best performance
  - Makes it possible to easily explore a large space of configurations

- Strong data model enables data partitioning that is understood by the system
Questions?