Regent and Pygion

Elliott Slaughter
Staff Scientist, SLAC National Accelerator Laboratory

Legion Retreat
December 4, 2024

Programming Legion

Direct Interfaces

- C++, Fortran, Regent, Pygion
- Expose a concept of tasks, regions, partitions
- User is responsible for selecting task (data) granularity
- Features correspond 1-1 with Legion programming model
- Good for users who need complete control

Indirect Interfaces

- Legate, FlexFlow, FleCSI
- Expose domain-specific concepts
- User is not (usually) responsible for selecting task (data) granularity
- Features are higher level, sometimes with an optional fallback to explicit task-based programming
- Good for users who need ease of use (and may require additional effort to regain control when needed)

Direct Legion Interfaces

C++ API

- The venerable Legion C++ API, used directly from C++ applications
- Template-based metaprogramming
- Statically type checked, but limited (or no) checking of Legion features
- Code is verbose *
- Code has to be written “just right” to execute efficiently in Legion *
- Write GPU code manually in CUDA, HIP, Kokkos, etc.
- Immediate access to bleeding edge Legion features

Regent

- Language written to target the Legion programming model
- Powerful metaprogramming via Lua
- Statically type checked (includes full checking of Legion features)
- Code is compact
- Automatically optimizes Legion API calls to improve execution efficiency without user intervention
- Automatically generate GPU code for tasks

Pygion

- Programming interface for Legion in Python
- No metaprogramming (but dynamic)
- Dynamically type checked (includes full checking of Legion features)
- Code is compact
- API optimization partially automated, requires some knowledge of “good” code patterns, but ergonomic to write
- Call Python libraries for GPU (CuPy, PyTorch, etc.)

* BYOA: Bring Your Own Abstraction
Legion is generally intended to be used with user-provided abstractions

Code Sample: A Task Launch

C++ API

```
IndexSpace colors =
   runtime->create_index_space(ctx, Rect<1>(0, 1));
float a = 1.23;
IndexLauncher launch(
   TID_SAXPY, colors,
   TaskArgument((void *)&a, sizeof(a)),
   ArgumentMap());
launch.add_region_requirement(RegionRequirement(
   P, 0, READ_WRITE, EXCLUSIVE, S));
launch.add_region_requirement(RegionRequirement(
   P, 0, READ_ONLY, EXCLUSIVE, S));
launch.add_field(0, FID_Y);
launch.add_field(1, FID_X);
runtime->execute_index_space(ctx, launch);
```

Regent

```
for i = 0, 2 do
   saxpy(P[i], 1.23)
end
```

Pygion

```
for i in IndexLaunch([2]):
    saxpy(P[i], 1.23)
```

Code Sample: A GPU Task

C++ API (and CUDA)

```
__global__
void gpu_saxpy(const float a,
                         Rect<1> rect,
                         FieldAccessor<READ_ONLY, float, 1>
acc_x,
                         FieldAccessor<READ_WRITE, float, 1>
acc_y)
{
   int p = bounds.lo + (blockIdx.x * blockDim.x) +
threadIdx.x;
   if (p <= bounds.hi)
      acc_y[p] += a * acc_x[p];
}

__host__
void saxpy(const Task *task,
                const std::vector<PhysicalRegion> &regions,
                Context ctx, Runtime *runtime) {
   FieldAccessor<READ_WRITE,float,1> acc_y(
     regions[0], FID_Y);
   FieldAccessor<READ_WRITE,float,1> acc_x(
     regions[1], FID_X);
   float a = *(const float*)(task->args);

   Rect<1> rect =
      runtime->get_index_space_domain(
         ctx, task->regions[0].region.get_index_space());
   size_t num_elements = rect.volume();

  size_t cta_threads = 256;
   size_t total_ctas = (num_elements + (cta_threads-
1))/cta_threads;
   gpu_saxpy<<<total_ctas, cta_threads>>>(a, rect, acc_x,
acc_y);
}
```

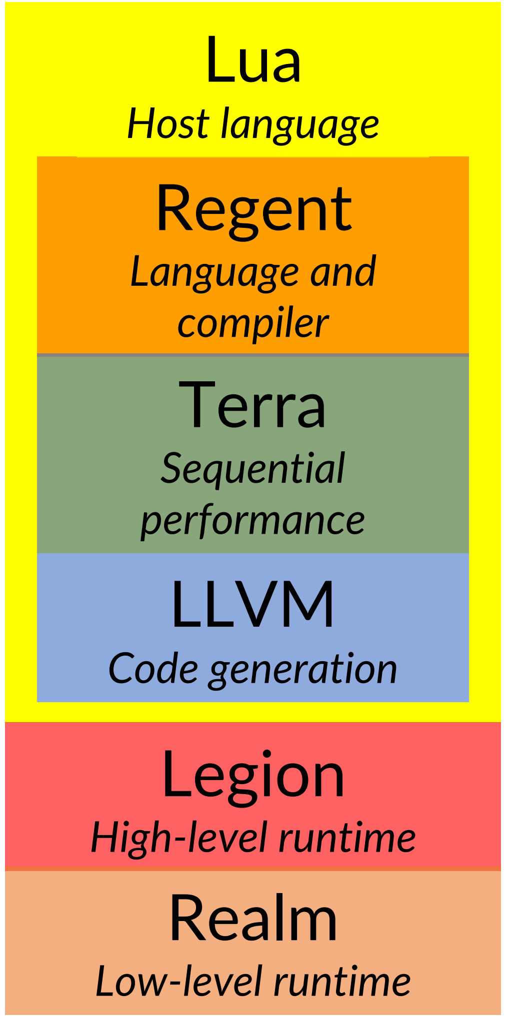
Regent

```
__demand(__cuda)
task saxpy(S : region(fields), a : float)
where reads writes(S.y), reads(S.x)
do
   for i in S do
      S[i].y += a * S[i].x
   end
end
```

Pygion

```
@task(privileges=[RW('y') + R('x')])
def saxpy(S, a):
   x = cupy.asarray(S.x)
   y = cupy.asarray(S.y)
   y += a * x
   S.y[:] = cupy.asnumpy(y)
```

Regent Stack



Terra: What's New

A Lot of New Activity In Terra

- LLVM 18 (and 17, 16, 15)
- SPIR-V backend (for Intel GPUs)
- RAII
- Allocators
- Smart pointers
- Concepts
- Ranges
- Linear algebra wrappers
- Test framework
- Package manager

Work in progress by:
Rene Hiemstra, PhD (TU/e)
Torsten Kessler, PhD (TU/e)

RAII In Terra

Resource Allocation Is Initialization

- Common programming style popularized by C++
- Implemented via metamethods in Terra
  - __init
  - __dtor
  - __copy
  - Can be methods or macros
- Note: no rvalue references, so this is equivalent to C++03 or Rust

Becomes

var a : A	→	**var** a : A a:__init()
a = b	→	A.methods.__copy(b, a)
var a : A = b	→	**var** a : A a:__init() A.methods.__copy(b, a)

Regent: What's New

Since December 2022

- Nested predication
- (More) Pygion interop
- Automatic future map elision in index launches
- HIP multi-GPU per rank
- ROCm 6.0
- Complex in std/format
- Compiler determinism fixes
- SCR removal: long live DCR
- FFT library: see talk later today

Coming Up Next

- Intel GPU support
  - We've made some progress, but still a ways out

Further Out (?)

- More flexible assignment of regions/partitions
- Gather/scatter copies
- Compact sparse instances
- Talk to me! These get prioritized based on user needs

Predication Update

Now Supports More Code Patterns

```
if c1 then
  some_task(...)
end
```

Becomes:

```
some_task(..., predicate=c1)
```

```
if c1 then
  x = other_task(...)
end
```

Becomes:

```
x = other_task(..., predicate=c1, else_value=x)
```

```
if c1 then
  if c2 then
    some_task(...)
  end
end
```

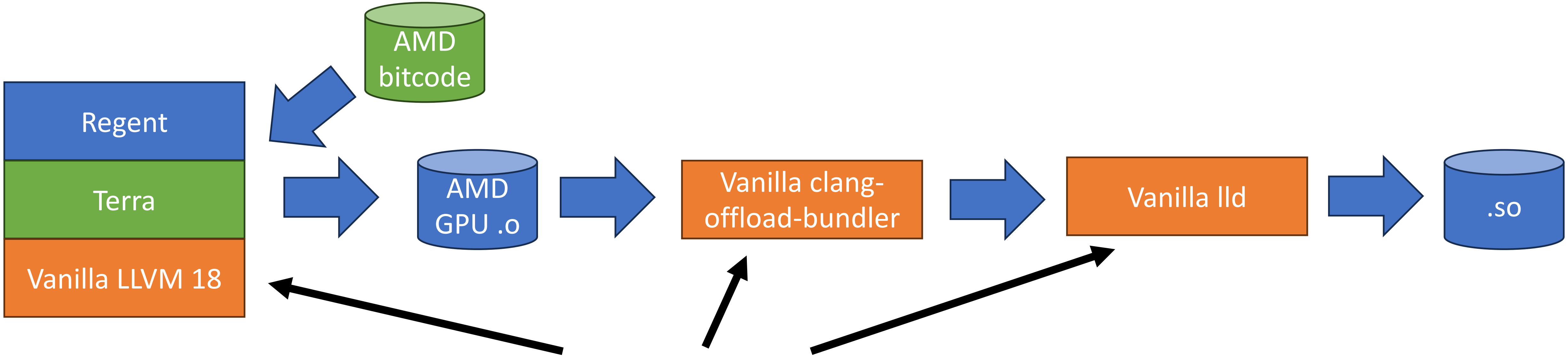
Becomes:

```
c1and2 = c1 and c2
```

```
some_task(..., predicate=c1and2)
```

ROCm 6.0 Support

- Requires LLVM 18
  - Previous LLVM versions generate incompatible code and *cannot* work with ROCm 6.0
  - Can't use AMD's LLVM fork either: it's (even more) badly broken
  - Plan is to track vanilla LLVM in the future (this is why keeping up to date matters!)

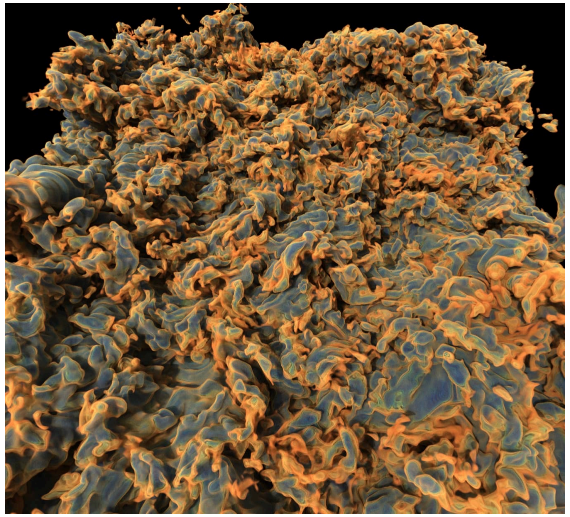


Use vanilla LLVM components at each stage of the pipeline
(except AMD's bitcode libraries)

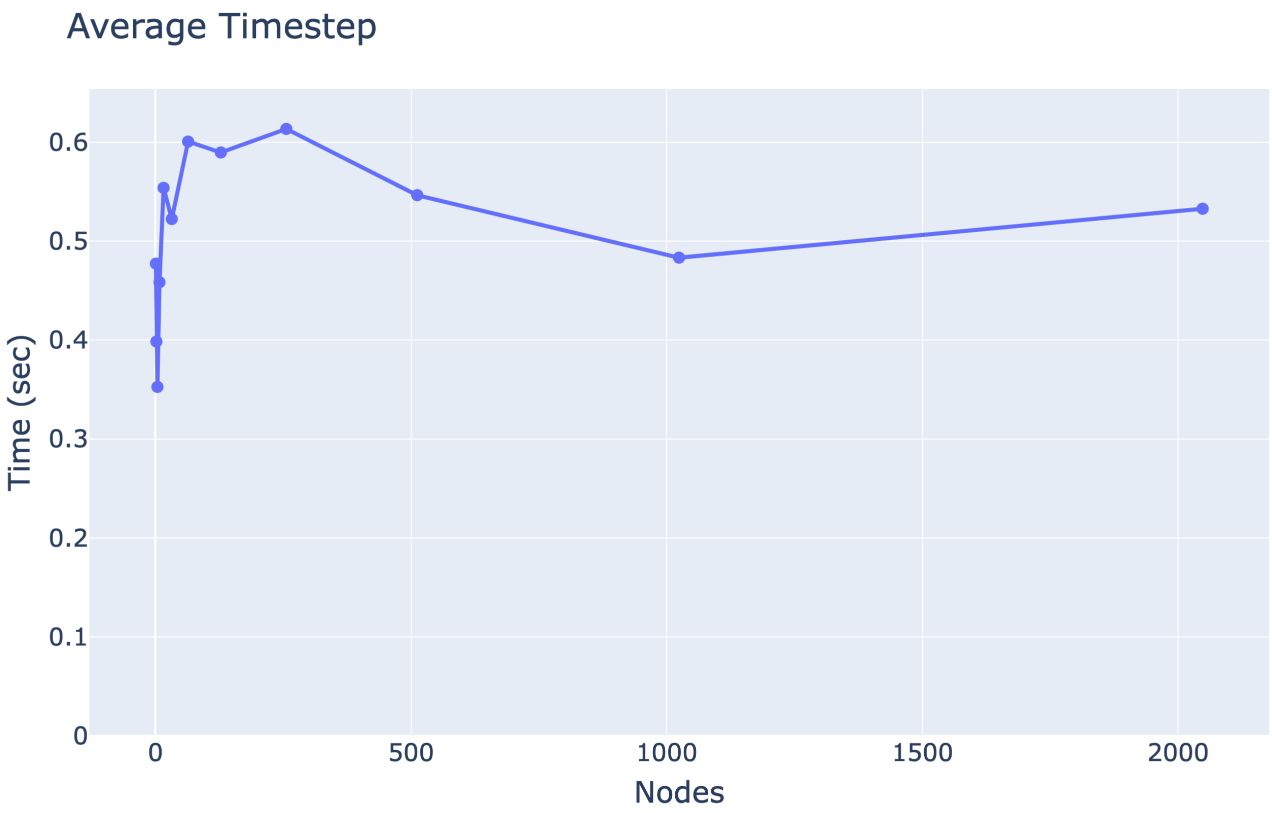
S3D Scaling on Frontier

Progress in Scaling S3D in 2024

- Regent-based code for direct numerical simulation of turbulent combustion chemistry from Sandia
- Combination of auto-generated and DSL-based kernels for NVIDIA and AMD GPUs
- Scaled up to 8192 nodes on Frontier
  - 2048 nodes shown at right



Heat Release from Ammonia Combustion



Weak Scaling on Frontier
Ammonia, 2×128^3 grid points/node

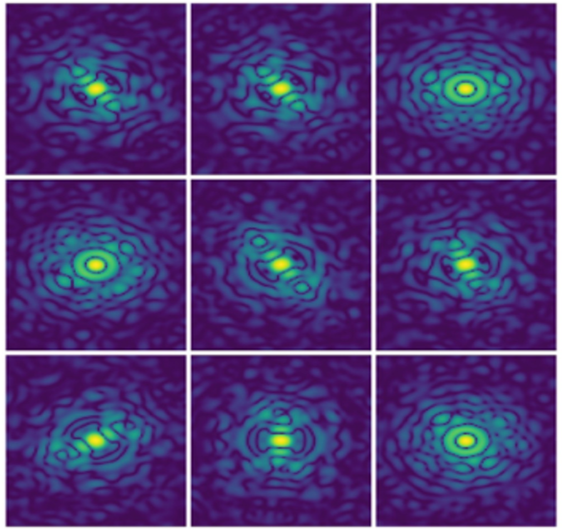
Pygion: What's New

Since December 2022

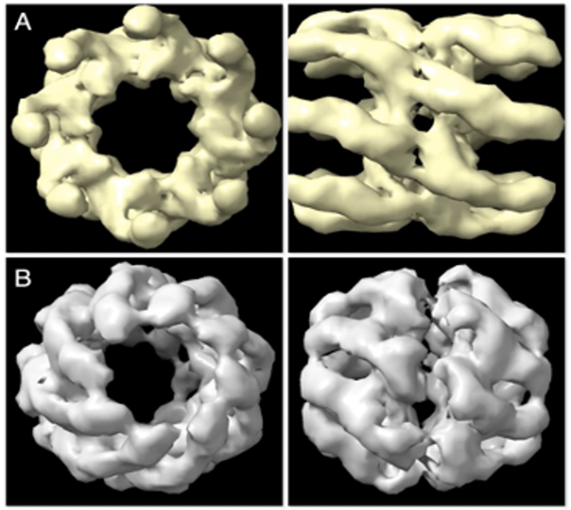
- Improved interop with Regent
- … That's it? 😆
- Pygion is stable, supports major use cases, and has been used in production
- We have a website! https://legion.stanford.edu/pygion

SpiniFEL: Single Particle Imaging for XFEL

Input: X-ray diffraction images



Output: 3D reconstruction of each protein conformation

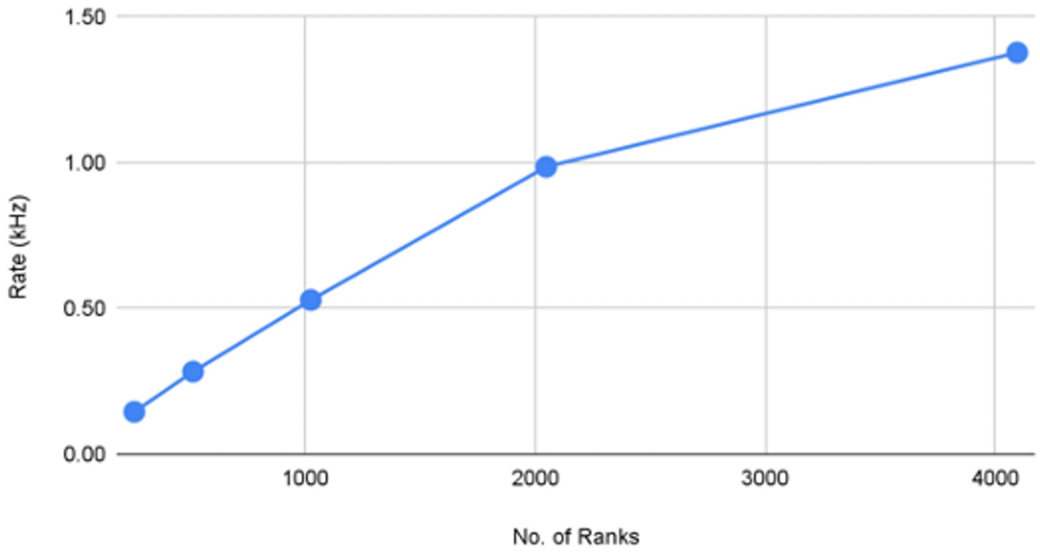


Design Principles

- Kitchen sink software design
  - If it exists, and it works, use it
  - NumPy, CuPy, Numba, hand-written CUDA, third-party CUDA libraries
- Pygion tasking as the orchestration layer

Lessons Learned

- Tasking layer was a non-issue
  - No production issues due to Pygion
- Kitchen sink approach caused porting issues



Weak scaling on up to 4096 GPUs on Frontier

[Mirchandaney et al., WAMTA 2024]

Questions?

eslaught@slac.stanford.edu