A Brief History of Legion

Alex Aiken

Stanford

Joint work with LANL, SLAC & NVIDIA

Context

- This talk is about Legion the project
  - What worked
  - What didn’t
  - The surprises
- Not about Legion the research
  - Except a couple of ideas relevant to the story

Prehistory

- Worked on Sequoia (PSAAP)
  - With Pat Hanrahan and Bill Dally
  - Strong performance results
  - But very static model was overly restrictive

- Wanted to investigate a more dynamic approach
  - Needed to start over …
  - A runtime system
  - Task-based
  - Asynchronous
  - Hardware agnostic
  - First-class data partitioning

2012: Legion

- The original group
  - Mike Bauer (systems, computational science)
  - Sean Treichler (long-time NVIDIA engineer)
  - Elliott Slaughter (programming languages)

- Pat Hanrahan brought us into the EXaCT Center
  - Met Jackie Chen’s combustion chemistry group
  - Began to interact more with Pat McCormick

2014: S3D

- Ported S3D to Legion
    - 100KLOC FORTRAN => 10KLOC Legion C++
    - True codesign effort
    - 7X improvement over FORTRAN-MPI at scale
    - Immediately became a production code

- The discovery
    - Legion successfully late-binds performance decisions
    - Makes finding a very fast implementation possible

- Variety of reactions
    - From credulous to incredulous

2015: Regent

- Programming language targeting Legion API
  - Simplified the programming model
  - Ability to write kernels that took advantage of Legion
  - True portability through code generation

- Not everyone wants or can use Regent
  - Other constraints sometimes dictate working in C++
  - We now had two programmer interfaces to support

Late 2015: An Inflection Point

- Project was ~4 years old
  - Already ancient for an academic effort
- Original students were close to graduating
  - Mike (2014) and Sean (2016) went to NVIDIA Research
  - Elliott (2017) went to SLAC
- Stop or try to continue?

2015: Start of Phase 2

- Design flaws that had to be fixed
  - Partitioning too hard to use and too slow
  - Solution: Dependent Partitioning (2016)

  - Control bottleneck in launching 100's of tasks
  - Solution: Control replication (2017, static)
  - Solution: Control replication (2018, dynamic)
- Interoperation
  - Solution we liked in 2017
- Invested in testing, debugging, profiling tools
  - Transitioning past a research project

Broadening

- Extensive collaborations with Los Alamos
  - FleCSI
  - Later ECP
  - Summer internships
- Bootcamps 2014, 2015, 2017, ?
- Graduate class at Stanford
  - Teach Regent
  - Students do a substantial project
  - Used in multiple PhD theses

2016 PSAAP II

- Multiphysics problem
  - Turbulence, particles, radiation
  - Close collaboration with ME at Stanford

- Initial plan: Develop two codes
  - One in DSL that targeted Legion
  - One in MPI
  - Rationale: Risk mitigation, ability to do comparisons

2017-8 A Crisis

- Two-system effort had practical problems
  - Divided effort meant less progress on both
  - Challenge to keep the systems equivalent
- MPI system became too difficult to manage
- Could we use Legion for the one and only system?
  - DSL addressed turbulent fluid flow
  - But DSL couldn’t be extended to handle particles/radiation
  - Didn’t have the capacity to write two more DSLs
- Solution
  - Write particle/radiation portions in Regent
  - Continue to use the DSL for fluid flow

The Resolution

- But developers decided they wanted one language
  - Regent
- Led to the Regent auto-parallelizer
  - DSL compiler technology generalized and incorporated into the Regent compiler (2019)
- Result is Soleil-X (2019)
  - A full multiphysics code
  - Runs on multiple supercomputers w/o code changes
  - At scale and efficiently
  - And is < 10KLOC

2020: Entering Phase 3

- Legate (NVIDIA)
  - Accelerated Numpy built on Legion
- FlexFlow
  - TensorFlow replacement built on Legion
  - Used by FaceBook, ECP, others
- Pygion
  - Python interface to Legion API
  - ECP ExaFel project
- PSAAP III

Summary

- Bootstrapping a programming model takes time
  - The more new technology, the more time
  - Team must write both the system and initial applications
  - Second system effect
  - Many bumps, turns in the road
- Partners are critical
  - Initial user(s) with a high tolerance for pain
  - Backers who can tolerate risk
- Tech transfer is different than research
  - Requires longer time scales, non-research elements
  - Necessary to keep the original team involved

Legion

Legion website: http://legion.stanford.edu