Scientific Simulation with Eclipse - From Zero Code to Running on Lots of Cores in 10 Minutes

EclipseCon North America 2016

Reston, VA

March 9, 2016

Alex McCaskey
Eclipse ICE Team
Oak Ridge National Laboratory
Computer and Mathematics Division
mccaskeyaj@ornl.gov
What’s the Problem?

• When it comes to Scientific Discovery we have Theory, Experiment… and Computation
• Major effort in providing scientific computing technologies, but they are inaccessible to most
• What good is a tool if average domain scientists can’t use it?
• Can the Eclipse Platform help in any of this? It’s done it for software development, maybe also for science?
Outline

• An application of ICE to a real scientific computing tool - MOOSE.
• Standard Model from MOOSE perspective
• Details on ICE integration
• 10 Minute Demo - go from ZERO MOOSE code to running in parallel with Eclipse

Additional Resources:  http://www.eclipse.org/ice

Binaries @ Eclipse Downloads  http://download.eclipse.org/ice/builds/next

YouTube  https://goo.gl/HpclLq

Ohloh.net  ohloh.net/p/eclipseice
First off, what is MOOSE?

- Extensible, flexible, general FEM C++ framework - JFNK
- Problem dimension agnostic
- User Code is automatically parallel
- Your application is a definition of physics Kernels

This is all extensible! If you need new functionality for a given sub-system, just write a new subclass!
MOOSE is all about extending MOOSE!

- MooseApp Class, entry point for your physics application
- Declare your physics as a set of Kernel and Boundary Condition classes

Strong Form
\[ \rho C_p \frac{\partial T}{\partial t} - \nabla \cdot k(T, B) \nabla T = f \]

Weak Form
\[ \int_{\Omega} \rho C_p \frac{\partial T}{\partial t} \psi_i + \int_{\Omega} k \nabla T \cdot \nabla \psi_i - \int_{\partial\Omega} k \nabla T \cdot n \psi_i - \int_{\Omega} f \psi_i = 0 \]

Actual Code
```
return _k[_qp]*_grad_u[_qp]*_grad_test[_i][_qp];
```
Typical MOOSE Application Developer/User Workflow

1. Get MOOSE and Build it (not that easy)
2. Fork the Stork (could be streamlined)
3. Write C++ code for your application’s Kernels, BCs, etc. (most users could use an upgrade to a full featured IDE)
4. Build and run the new application (build locally, remotely, execute in parallel, all not easy)
A cooler model of Scientific Computing

It would be better to have a computer program handle all of that...

How can Eclipse, and specifically ICE, improve the utility of a framework like MOOSE?
A cooler model of Scientific Computing

What tools are in ICE for **Defining the Problem**...

- ICE IOService - provider of IReader/IWriter realizations
- Model Item - Expose Form to User for input model data
A cooler model of Scientific Computing

What tools are in ICE for **Running the Simulator**...

- ICE JobLauncher Item: handle local/remote execution, executions with docker, handle data tracking and execution with MPI,TBB,OpenMP parallelism
A cooler model of Scientific Computing

What tools are in ICE for **Analyzing the Output**...

- The new EAVP!
  - Hooks to VisIt, Paraview, 1D plotting tools
  - ICE Resource Component data structure

Define the Problem

Run the Simulator

Analyze Output

01100010
01101001
01101110
01100001
01110010
01111001

Archive Output
A cooler model of Scientific Computing

What tools are in ICE for Scientific Code Development...

Seriously???? This is Eclipse…
- CDT, EGit, PTP, Pydev... and the list goes on and on
- But even more so in ICE… The new Developer Menu
A cooler model of Scientific Computing

Let’s walk through this Standard Model from MOOSE’s perspective.
Standard Model of Scientific Computing... For MOOSE!

**Define the Problem**

Write an input file in a format reminiscent of a dead language

MOOSE model input involves GetPot and YAML.

Framework defines the allowed input with YAML

Users Provide GetPot Input

YAML

```yaml
parameters:
  subblocks:
    - name: /Kernels/*
      description: |

parameters:
  - name: type
    required: Yes
    default: !!str
    cpp_type: std::basic_string<char>
    group_name:
      description: |
      A string representing the Moose Object

  subblocks:
    - name: /Kernels/AdvDiffReaction1
      description: |

parameters:
  - name: A0
    required: No
    default: !!str 0
    cpp_type: double
    group_name:
      description: |
```

GetPot

```json
[Kernels]
  [/Convection]
    type = Convection
    variable = temp
    velocity = '0 0 1'
  [/Diffusion]
    type = Diffusion
    variable = temp
  [/TimeDerivative]
    type = TimeDerivative
    variable = temp

<Mesh>
  file = mug.e
  type = FileMesh
```
Define the Problem

Write an input file in a format reminiscent of a dead language

MOOSE model input involves GetPot and YAML.

Framework defines the allowed input with YAML

Users Provide GetPot Input

Standard Model of Scientific Computing… For MOOSE!

Hook into ICE IOService and provide user-friendly views of the data
MOOSE runs on anything from your laptop to a large cluster.

We’ve even run it on Windows using Docker!

MOOSE execution takes a GetPot input file and can be run serially or parallel.

Run the Simulator

MOOSE version: git commit 9fec80a on 2016-03-04
PETSc Version: 3.6.1

Parallelism:
- Num Processors: 32
- Num Threads: 1

Mesh:
- Distribution: serial
- Mesh Dimension: 3
- Spatial Dimension: 3
- Nodes:
  - Total: 3774
  - Local: 161
- Elems:
  - Total: 2476
  - Local: 77
- Num Subdomains: 1
- Num Partitions: 32
- Partitioner: metis
Standard Model of Scientific Computing… For MOOSE!

- MOOSE outputs large Exodus files with solution fields over the problem mesh.
- Postprocessor data in a CSV format

Typical Moose Output

MOOSE users produce large data sets primarily in Exodus format.

Analyze Output

01100010
01101001
01101110
01100001
01110010
01111001

Analyze simulation output in its most raw and unlimited form
Leverage tooling in EAVP

- VisIt/Paraview integration
- ICE CSV Plot Engine
- Add resultant files to an ICE ResourceComponent

Analyze output in its most raw and unlimited form.
Standard Model of Scientific Computing… For MOOSE!

Development

MOOSE Dev at a High-level:
1. Version control through Git
2. C++, MPI, Threading with TBB, OpenMP
3. Libmesh knowledge and expertise
4. Basic OO knowledge
5. Most MOOSE devs develop extensions to MOOSE in emacs or vim.

Make it do what it do...
MOOSE Items in ICE

Extend the ICE Item for MOOSE

- **MooseModel** - Keep track of Moose Input Tree, read write Moose input file
- **MooseLauncher** - Keep track of files required at launch, handle local/remote/docker job launching and parallelism
- **Moose** - Provide a more unified workflow, direct the model and launcher items to model the entire Moose workflow.
The MOOSE Developer Menu Contributions

Fork the Stork - fork the repo, rename it, clone it, run code gen script, import project, create make target… Phew!
The Finished Product
Demo - Compute the temperature as a function of time for a coffee mug with Convection and Diffusion physics at play.

From no code to running in parallel in 10 minutes!
Evaluate the Sessions

Sign in and vote at eclipsecon.org