In Situ Analysis and Visualization with SENSEI

13 November 2017
Supercomputing 2017
Welcome! Why are we here?

Problem: FLOPS >> I/O, potential for lost science

Approach: do as much processing as possible while data still resident in memory?

Why This Tutorial? To inform you of issues involved, to show you what technologies are available and how to use them.
Outline

• Introduction to *In Situ* Analysis and Visualization
• SENSEI *In Situ* Data Interface
• Instrumenting data sources and endpoints (C++)
• SENSEI *In Situ* Demonstrations with Coupled Infrastructures
  – Data extracts with Libsim
  – Computational monitoring with ParaView Catalyst
  – Autocorrelation with ADIOS
  – Using SENSEI via Python
• *In Situ* Costs and Performance
• Closing thoughts
What are the problems?

Not enough I/O capacity on current HPC systems, and the trend is getting worse.

If there’s not enough I/O, you can’t write data to storage, so you can’t analyze it: lost science.

Energy consumption: it costs a lot of power to write data to disk.

Opportunity for doing better science (analysis) when have access to full spatiotemporal resolution data.
Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

- Computation: 125 PB/s
- Node memory: 4.5 PB/s
- Interconnect: 24 TB/s
- Storage: 1.4 TB/s

On Node Visualization

Image courtesy Ken Moreland
The problem is not going away

## How does Summit compare to Titan

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summit</th>
<th>Titan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Performance</td>
<td>5-10x Titan</td>
<td>Baseline</td>
</tr>
<tr>
<td>Number of Nodes</td>
<td>~3,400</td>
<td>18,688</td>
</tr>
<tr>
<td>Node performance</td>
<td>&gt; 40 TF</td>
<td>1.4 TF</td>
</tr>
<tr>
<td>Memory per Node</td>
<td>&gt;512 GB (HBM + DDR4)</td>
<td>38GB (GDDR5+DDR3)</td>
</tr>
<tr>
<td>NVRAM per Node</td>
<td>800 GB</td>
<td>0</td>
</tr>
<tr>
<td>Node Interconnect</td>
<td>NVLink (5-12x PCIe 3)</td>
<td>PCIe 2</td>
</tr>
<tr>
<td>System Interconnect (node injection bandwidth)</td>
<td>Dual Rail EDR-IB (23 GB/s)</td>
<td>Gemini (6.4 GB/s)</td>
</tr>
<tr>
<td>Interconnect Topology</td>
<td>Non-blocking Fat Tree</td>
<td>3D Torus</td>
</tr>
<tr>
<td>Processors</td>
<td>IBM POWER9™, NVIDIA Volta™</td>
<td>AMD Opteron™, NVIDIA Kepler™</td>
</tr>
<tr>
<td>File System</td>
<td>120 PB, 1 TB/s, GPFS™</td>
<td>32 PB, 1 TB/s, Lustre®</td>
</tr>
<tr>
<td>Peak power consumption</td>
<td>10 MW</td>
<td>9 MW</td>
</tr>
</tbody>
</table>

Data courtesy A. Geist (ORNL)
What is *in situ* data analysis and visualization?

Two use models:

- **Post processing** (*post hoc*): save to disk, then later, a separate analysis/vis program reads that data and operates on it.

- **In situ processing**: process data as it produced without writing to and reading from storage. Processed “in place”.
  - Many flavors/terms: tightly coupled, loosely coupled, in transit, co-processing, etc.
  - Practical view: anything processed but not written to persistent storage is *in situ*

---

**In situ** – no data movement:
Simulation and *in situ* methods share memory

**In transit** – data is moved:
Simulation and *in situ* methods do not share memory
The story is much more interesting than “in-situ” vs. “in transit”

In situ vs. in transit is an oversimplification of a much richer problem space

The “In Situ Terminology Project”
- A community effort (>50 participants)
- Identify “basis vectors” for describing aspects of in situ processing
  - Integration Type, Proximity, Access, Division of Execution, Operation Controls, Output Type
**In situ:** an “umbrella definition”

*In situ* is a term that covers a lot of territory:

*In Situ* Terminology project:
http://ix.cs.uoregon.edu/~hank/insituterminalogy/
Community effort to identify basis vectors and name them.
**In situ** has been around a long time: ancient history

E. Zajac, CACM 7(3), Mar 1964.

Direct-to-film process (simulation, calligraphic display exposes film) movie of a satellite orbiting a planet.

Is this *in situ*?
- Yes: no data ever landed on disk.

Why did he do it?
- “Standard practice” for that era, and many years that followed: direct-to-media more efficient.

[Link to movie page](#)
The 1990s: the golden era of coprocessing

Main idea: systems/methods that support interactive computation, computational monitoring and steering.

Packages from this era (partial list):

- pV3: custom distributed memory code (Haimes)
- AVS: co-routine processing (serial, mostly)
- CUMULVS: distributed memory M-to-N visualization, steering (based on PVM) (Kohl, et al.)

Common design patterns of 1990s

Many-to-one: AVS

“Tightly coupled”: pV3, custom projects

“Loosely coupled”, M-to-N: CUMULVS
Computational steering – human in the loop

Main idea: rapid convergence

Example: protein structure prediction, find optimal-energy conformation from initial conditions (NP-hard problem)

Approach:

• parallel computations that minimize energy for individual conformations

• User can examine any of these, perform manual tweaks to get “unstuck” from local minimum, then resume calculations.

Integrated computational environments

- Simplify building, running codes
- Many add-on capabilities for vis, analysis, debugging, data I/O, etc.

Examples: SCIRun, Cactus

Explorable extracts

Basic ideas:

• Overcome in situ primary weakness: know before you go.

• Use in situ computation to produce reduced-size datasets, e.g., images, data subsets, “extracts” like collections of features, etc.

• These “data extracts” are much smaller in size compared to doing full resolution data I/O.

• Use some post-processing tool to view/analyze/interact with these extracts.

Climate modeling example using Catalyst and Cinema in our STAR paper.

Chen et al., Interactive, Internet Delivery of Visualization via Structured, Prerendered Multiresolution Imagery. TVCG 14(2), 2008.

In situ projects over the years (approximate, partial)

1964: Zajac, direct-to-film animations

1990s: Code coupling, computational steering:
   - AVS
   - pV3
   - CUMULVS

2000s (early): Integrated Computational Environments:
   - SCIRun
   - CACTUS

2000s (late): Computing Extracts for Post Hoc Use
   - Multiresolution, precomputed images
   - Topology
   - Geometry

Present day:
   - VisIt/Libsim, Paraview/Catalyst: scalable vis infrastructure accessible in situ
   - ADIOS: I/O library approach
   - SENSEI: generic in situ interface
   - Other nascent efforts
Roadmap of *In Situ* Software Infrastructure for Today

Miniapps from SENSEI software collection, C++ and Python

Sim codes: LAMMPS, AVF-LESLEE

SENSEI Generic *In Situ* Interface

ADIOS

ParaView/Catalyst

VisIt/Libsim

Other endpoints or methods: OSPray, etc.
Generic processing sequence (sim code view)

Initialize

Here’s some data, do something with it

Finalize

Here’s where things get interesting.
This could be an entry point into some significant processing, like a complex distributed workflow.
Tutorial VM & web-site

• USB drive available which contains:
  • All demos shown here
  • A pdf of the slides for reference
    • Includes hidden slides with more details not covered here due to time restrictions
• www.sensei-insitu.org/tutorials web-site also has this information
SENSEI *In Situ* Data Interface
how to choose?
Can WE....

Enable use of any in situ framework?

Develop analysis routines that are portable between codes?

Make it easy to use?
The *current* problem set

www.olcf.ornl.gov/center-projects/adios
wci.llnl.gov/simulation/computer-codes/visit
www.paraview.org/in-situ
**In situ infrastructures**

Relatively new

- Until recently, *ad hoc, proof-of-concept prototypes*
- However, several *production quality in situ infrastructures* have emerged

**ADIOS** provides tools for *in situ I/O* and some *analysis*

- ADIOS allows simulations to adopt *in situ* techniques by *leveraging* their *advanced I/O infrastructures* that enable co-analysis pipelines *rather than changing the simulator*.
- The non-intrusive integration *provides resilience* to third party library bugs and possible jitter in the simulation.

**ParaView** and **VisIt** both provide tools for *in situ analysis* and *visualization*

- ParaView **Catalyst** can be *tightly* or *loosely* linked to a simulation, allowing the simulation to *share data* with Catalyst for analysis and visualization.
- Similar capabilities are available within VisIt with the **Libsim** library.
- Catalyst (through **Live**), Libsim, and ADIOS enable the *opposite flow of information*, sending data from the client to the simulation, enabling the possibility of *in situ* and/or *monitoring/simulation steering*.
Our approach

Data model
- The lingua franca allowing an analyses to access simulation data consistently across a variety of simulations

Data adaptor
- Convert simulation data to/from the data model

API
- For instrumenting simulation and driving analyses

Library
- Providing off the shelf access to Libsim, Catalyst and ADIOS capabilities
Write once run everywhere

The **SENSEI API** enables connection of simulation data sources to visualization and analysis back ends

- From the perspective of the simulation, the back ends (analysis/vis codes) are interchangeable

The **SENSEI data model** enables viz & analysis codes to access data through a unified API.

- From the perspective of the analysis/visualization code, data sources (simulations) are interchangeable
Data model: VTK

Used by ParaView Catalyst and VisIt/Libsim

Supports common scientific dataset types

On going independent efforts to evolve for exascale

Supports using simulation memory directly (zero-copy) for multiple memory layouts

www.vtk.org
vtkDataSet subclasses

- vtkImageData
- vtkUniformGrid
- vtkRectilinearGrid
- vtkStructuredGrid
- vtkPolyData
- vtkUnstructuredGrid

www.vtk.org/doc/nightly/html/classvtkDataSet.html
Field information

Store information defined over grids
stored in concrete classes that derive from vtkDataArray
- vtkFloatArray
- vtkIntArray
- vtkDoubleArray
- vtkUnsignedCharArray
- ...

Point data

Cell data
Architecture

simulation → data adaptor → bridge → analysis adaptor → analysis
The data adaptor

- Provides the API through which data is accessed
- Converts simulation data structures into VTK data structures on demand
  - Try make use of VTK’s array zero copy facility
- Is used by the analysis adaptor to access simulation data on demand
sensei::DataAdaptor pure virtual class

/// DataAdaptor is an abstract base class that defines the SENSEI data interface.
class DataAdaptor : public vtkObjectBase
{
 public:
    /// Return the data object with appropriate structure.
    virtual vtkDataObject* GetMesh(bool structure_only = false) = 0;

    /// Adds the specified field array to the mesh.
    virtual bool AddArray(vtkDataObject* mesh, int association, const std::string& arrayname) = 0;

    /// Return the number of field arrays available.
    virtual unsigned int GetNumberOfArrays(int association) = 0;

    /// Return the name for a field array.
    virtual std::string GetArrayName(int association, unsigned int index) = 0;

    /// Release data allocated for the current time step.
    virtual void ReleaseData() = 0;

    /// Convenience method to set and get the time
    double GetDateTime();
    void SetDateTime(double time);

    /// Convenience method to set and get the time step
    int GetDateTimeStep();
    void SetDateTimeStep(int index);
...};
The analysis adaptor

• Provides the API for driving the analysis
• Invoked by the bridge through the simulation when it is time for analysis
• You pass in a data adaptor instance, which the analysis code uses to access simulation data structures
sensei::AnalysisAdaptor pure virtual class

/// @brief AnalysisAdaptor is an abstract base class that defines
/// the analysis interface.
class AnalysisAdaptor : public vtkObjectBase
{
public:
    /// @brief Execute the analysis routine.
    /// @brief Execute the analysis routine.
    virtual bool Execute(DataAdaptor* data) = 0;
};
ConfigurableAnalysisAdaptor

- Generalized SENSEI analysis adaptor reads in XML file specifying what available analyses to compute during a simulation run
- Can be used with internal SENSEI analyses, ADIOS, ParaView Catalyst, and VisIt/Libsim endpoints
- Specifies which analysis endpoints and what in situ analyses to use

```xml
<sensei>
    <analysis enabled="1" type="catalyst" pipeline="pythonscript" filename="slice_contourcut.py"/>
    <analysis enabled="0" type="autocorrelation" array="data" association="cell" window="10" k-max="3"/>
    <analysis enabled="1" type="adios" filename="oscillators.bp" method="MPI"/>
    <analysis enabled="0" type="libsim" options="-no-icet" plots="Pseudocolor" plotvars="cell_data" slice-origin="32.5,32.5,32.5" slice-normal="0,0,1" image-filename="slice%ts" image-width="1600" image-height="1600" image-format="png"/>
</sensei>
```
The bridge

- Is where you create, initialize, and manage your data and analysis adaptors
- Is where you execute the analyses adaptors as needed
- Typically consists of 3 functions: Initialize, Compute and Finalize
Instrumenting Data Sources and Endpoints with SENSEI
Instrumentation tasks

1. Data
   - Decide if you can use `sensei::VTKDataAdaptor`
   - Or write an adaptor derived from `sensei::DataAdaptor`

2. Analysis
   - Decide if you can use existing analyses: Libsim, Catalyst, Adios, etc
   - And/Or implement new analyses derived from `sensei::AnalysisAdaptor`

3. Bridge
   - Implement Initialize, Compute, and Finalize methods/functions
   - Instrument the simulation to call the bridge code at the right times
Oscillator miniapp overview

- MPI based C++ code that simulates a collection of periodic, damped, or decaying oscillators over a Cartesian grid
- Each oscillator is convolved with a Gaussian of a prescribed width
- Executable inputs are oscillator parameters, time resolution, length of the simulation, grid dimensions and grid partitioning
Instrumenting the oscillator mini-app to use SENSEI

Most of the work is in creating VTK objects to represent simulation grid and field data

- Create a class that derives from sensei::DataAdaptor and implements:
  - virtual vtkDataObject* GetMesh(bool structure_only=false) = 0;
  - virtual bool AddArray(vtkDataObject* mesh, int association, const std::string& arrayname) = 0;
  - virtual unsigned int GetNumberOfArrays(int association) = 0;
  - virtual std::string GetArrayName(int association, unsigned int index) = 0;
  - virtual void ReleaseData() = 0;
Creating the VTK grid – GetMesh() method

```cpp
vtkDataObject* DataAdaptor::GetMesh(bool vtkNotUsed(structure_only))
{
    if (!this->internals->Mesh)
    {
        this->internals->Mesh = vtkSmartPointer<vtkMultiBlockDataSet>::New();
        this->internals->Mesh->SetNumberOfBlocks(static_cast<unsigned int>(internals.CellExtents.size()));
        for (size_t cc=0; cc < internals.CellExtents.size(); ++cc)
        {
            internals.Mesh->SetBlock(static_cast<unsigned int>(cc), this->GetBlockMesh(cc));
        }
    }
    this->AddArray(this->internals->Mesh, vtkDataObject::FIELD_ASSOCIATION_CELLS, "data");
    return this->internals->Mesh;
}

vtkDataObject* DataAdaptor::GetBlockMesh(int gid)
{
    vtkSmartPointer<vtkImageData>& blockMesh = this->internals->BlockMesh[gid];
    const diy::DiscreteBounds& cellExts = this->internals->CellExtents[gid];
    if (!blockMesh && areBoundsValid(cellExts))
    {
        blockMesh = vtkSmartPointer<vtkImageData>::New();
        blockMesh->SetExtent(
            cellExts.min[0], cellExts.max[0]+1,
            cellExts.min[1], cellExts.max[1]+1,
            cellExts.min[2], cellExts.max[2]+1);
    }
    return blockMesh;
}
```
Creating the VTK cell data – AddArray() method

```cpp
bool DataAdaptor::AddArray(vtkDataObject* mesh, int association, const std::string& arrayname)
{
    (void)association;
    bool retVal = false;
    DInternals& internals = (*this->Internals);
    vtkMultiBlockDataSet* md = vtkMultiBlockDataSet::SafeDownCast(mesh);
    for (unsigned int cc=0, max=md->GetNumberOfBlocks(); cc < max; ++cc)
    {
        if (!internals.Data[cc])
        {
            continue;
        }
        vtkSmartPointer<vtkImageData>& blockMesh = internals.BlockMesh[cc];
        if (vtkCellData* cd = (blockMesh? blockMesh->GetCellData(): NULL))
        {
            if (cd->GetArray(arrayname.c_str()) == NULL)
            {
                vtkFloatArray* fa = vtkFloatArray::New();
                fa->SetName(arrayname.c_str());
                fa->SetArray(internals.Data[cc], blockMesh->GetNumberOfCells(), 1);
                cd->SetScalars(fa);
                cd->SetActiveScalars("data");
                fa->FastDelete();
            }
            retVal = true;
        }
    }
    return retVal;
}
```
Implementing the bridge to SENSEI

Typically 3 calls:

- **Initialize()**
  - For the Oscillator we store the static Cartesian grid parameters
  - Specify what analysis will be done. For the Oscillator we use the ConfigurableAnalysis class.

- **Compute()**
  - For the Oscillator we do this with two calls: set_data() and analyze(), so that SENSEI may be disabled in benchmarks

- **Finalize()**
void initialize(MPI_Comm world, size_t window, size_t nblocks, size_t n_local_blocks,
    int domain_shape_x, int domain_shape_y, int domain_shape_z, int* gid, int* from_x,
    int* from_y, int* from_z, int* to_x, int* to_y, int* to_z,
    const std::string& config_file)
{
    (void)window;
    GlobalDataAdaptor = vtkSmartPointer<oscillators::DataAdaptor>::New();
    GlobalDataAdaptor->Initialize(nblocks);
    GlobalDataAdaptor->SetDataTimeStep(-1);

    for (size_t cc=0; cc < n_local_blocks; ++cc)
    {
        GlobalDataAdaptor->SetBlockExtent(gid[cc],
            from_x[cc], to_x[cc], from_y[cc], to_y[cc],
            from_z[cc], to_z[cc]);
    }

    int dext[6] = {0, domain_shape_x, 0, domain_shape_y, 0, domain_shape_z};
    GlobalDataAdaptor->SetDataExtent(dext);

    GlobalAnalysisAdaptor = vtkSmartPointer<sensei::ConfigurableAnalysis>::New();
    GlobalAnalysisAdaptor->Initialize(world, config_file);
}
Executing the in situ

```c
void set_data(int gid, float* data)
{
    GlobalDataAdaptor->SetBlockData(gid, data);
}

void compute(float time)
{
    GlobalDataAdaptor->SetDateTime(time);
    GlobalDataAdaptor->SetDataTimeStep(GlobalDataAdaptor->GetDateTimeStep() + 1);
    GlobalAnalysisAdaptor->Execute(GlobalDataAdaptor.GetPointer());
    GlobalDataAdaptor->ReleaseData();
}
```
void finalize(size_t k_max, size_t nbloclks)
{
    (void)k_max;
    (void)nbloclks;
    GlobalAnalysisAdaptor = NULL;
    GlobalDataAdaptor = NULL;
}
Overview of autocorrelation

Autocorrelation is a statistical test of a function with itself
• Generally done in time, although can also be done over a spatial integral, or in some cases both.

Simple definition:
• $C(\tau) = \frac{\sum_{i=1}^{N} F(i)F(i-\tau)}{N*C(\emptyset)}$

A plot of a series of 100 random numbers concealing a sine function
The sine function revealed in a correlogram produced by autocorrelation
Initializing the autocorrelation analysis adaptor

In order to compute autocorrelation need to provide:

- MPI communicator
- Autocorrelation window size
- Field type and name

```cpp
void Autocorrelation::Initialize(MPI_Comm world, size_t window, int association, std::string& arrayname, size_t kmax)
{
    AInternals& internals = (*this->Internals);
    internals.Master = make_unique<diy::Master>(world, -1, -1, &AutocorrelationImpl::create,
                                               &AutocorrelationImpl::destroy);

    internals.Association = association;
    internals.ArrayName = arrayname;
    internals.Window = window;
    internals.KMax = kmax;
}
```
Executing the autocorrelation analysis adaptor

Implement the Execute() method

- Use the passed in DataAdaptor object to get the desired data (grid and field data to compute the autocorrelation)
- Operate on grid and field information to compute desired result
Another example: instrumenting LAMMPS with SENSEI

- Large-scale Atomic/Molecular Massively Parallel Simulator
- Classical molecular dynamics code
- Runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain
- Accelerated performance on CPUs, GPUs, and Intel Xeon Phis
- Distributed by Sandia National Laboratories

http://lammps.sandia.gov/

Enabling in situ interactive visualization for large-scale molecular simulations

• LAMMPS is a good representative application of large scale molecular dynamics simulations
• Use LAMMPS as a library
  – Big advantage: No need to recompile or instrument LAMMPS original code
• Drive LAMMPS from a simple application instrumented with SENSEI
• Integrate OSPRay (Intel Software-Defined visualization) as an additional SENSEI infrastructure for interactive visualization
SENSEI architecture

SENSEI in situ

data adaptor

analysis adaptor
Architecture of LAMMPS instrumentation with SENSEI

- LAMMPS as library
- DRIVER
- LAMMPS input file

Simple app drives LAMMPS

OSERay as another infrastructure

- LAMMPS
- OSPRay
- SENSEI

data adaptor
bridge
analysis adaptor
analysis
Data format

• LAMMPS particle format is basically x,y,z coordinates with additional fields like atom type or radius)

• Add LAMMPS fix/external command in input file for LAMMPS to share pointers to its internal data after computing every timestep of the simulation

• Additional information here: Coupling LAMMPS to other codes http://lammps.sandia.gov/doc/Section_howto.html#howto-10
Callback function from LAMMPS (every timestep)

```c
void LAMMPSCallback(void *ptr, bigint ntimestep,
                      int nlocal, int *id, double **x, double **f)
{
  Info *info = (Info *) ptr;

  // extents
  double boxxlo = *((double *) lammps_extract_global(info->lmp,"boxxlo");
  double boxxhi = *((double *) lammps_extract_global(info->lmp,"boxxhi");
  double boxylo = *((double *) lammps_extract_global(info->lmp,"boxylo");
  double boxyhi = *((double *) lammps_extract_global(info->lmp,"boxyhi");
  double boxzlo = *((double *) lammps_extract_global(info->lmp,"boxzlo");
  double boxzhi = *((double *) lammps_extract_global(info->lmp,"boxzhi");

  // get pointer to atom types
  int *type = (int *) lammps_extract_atom(info->lmp,"type");

  // update SENSEI bridge
  bridge::Set_data(nlocal, id, type, x, boxxlo, boxylo, boxzlo, boxxhi, boxyhi, boxzhi);

  // visualize
  bridge::Execute();
}
```
OSPRay as an additional infrastructure

- Connect to SENSEI endpoint to query data
- Pull data back to distributed OSPRay client app running using OSPRay’s distributed device to provide an interactive viewer of the latest timestep

Image courtesy Will Usher, SCI, Univ. of Utah
Live demo

- Live demo on virtual machine
  - Running LAMMPS coupled to OSPRay for interactive visualization
  - Navigation: Use RIGHT click to zoom in/out, LEFT click to rotate

- Steps:

  In one terminal
  `% cd ~/sc17/demos/lammps_ospray`
  `% ./rundriver.sh`

  In a second terminal
  `% cd ~/sc17/demos/lammps_ospray`
  `% ./runviewer.sh`
SENSEI In Situ Demonstrations with Coupled Infrastructures
Data Extracts with VisIt/Libsim
Libsim puts VisIt in situ

- VisIt provides Libsim, a library that simulations may use to let VisIt connect and access their data.
- Avoids I/O and data movement.
- Supports automated data product generation.
- Also supports user-driven exploration of simulation data.

VisIt
- Versatile open source software for visualizing and analyzing petascale simulation datasets.

Libsim
- Enables simulations to perform data analysis and visualization in situ by applying VisIt algorithms to data.
Libsim enables flexible workflows

- Use the VisIt GUI to connect to your simulation and explore!
- Simulations are like any other data source
- Create automated routines to generate data in batch
- Program directly using Libsim
- Use VisIt session files
XDB workflow

- Use Libsim to instrument simulation so it produces **FieldView XDB files** for later visualization in Fieldview.

- **XDB** is a CFD format made of surfaces and streamlines, which provides geometry and field data that enables useful post-hoc analysis.

- Extractions are orders of magnitude smaller than other data, avoiding I/O bottlenecks.

- XDB’s are easily loaded into Fieldview.

- FieldView treats XDBs like it would the full data, except XDB’s are far faster!

- All operations available for volume-data are available for XDB data.

- Numerical operations match exactly (e.g., integration).

- XDB’s overcome in situ’s greatest perceived weakness: that you need to have some idea of what you want to see in the end.

- Permits interactive exploration using post-processing methods.

- Cheap enough to save frequently.
Flexible Extract Export

- Hard-coding plots and extracts limits flexibility
- SENSEI XML input file can select plots for extract creation and for rendering
  - Provides hints to Libsim
  - Specifies extracts, variables, files to write

```xml
<!-- SENSEI ConfigurableAnalysis Configuration file. set enabled="1" on analyses you wish to enable -->
<sensei>
</!-- Custom Analyses -->
<analysis type="histogram" array="P" association="point" bins="10" enabled="0" />
<analysis type="histogram" array="Rho" association="point" bins="10" enabled="0" />
<analysis type="histogram" array="T" association="point" bins="10" enabled="0" />

</!-- Libsim Analyses -->
<analysis type="libsim" frequency="5" operation="export"
plots="Pseudocolor" plotvars="P,Rho,T,Y_OH,Y_H2O,VORT"
filename="results/slicey%ts" slice-origin="0.02,0.02,0.02" slice-normal="0.,1.,0."
enabled="1"/>

<analysis type="libsim" frequency="5" operation="export"
plots="Pseudocolor" plotvars="P,Rho,T,Y_OH,Y_H2O,VORT"
filename="results/slicez%ts" slice-origin="0.02,0.02,0.02" slice-normal="0.,0.,1."
enabled="1"/>

<analysis type="libsim" frequency="5" operation="render"
plots="Pseudocolor" plotvars="VORT"
slice-origin="0.02,0.02,0.02" slice-normal="0.,0.,1." slice-project="1"
image-filename="results/slicez_T_%ts" image-width="1200" image-height="1200"
image-format="png"
enabled="1"/>
</sensei>
```
Instrumenting AVF-LESLIE simulation

- Created adaptor library for AVF-LESLIE
- Calls Compute function when we want to generate extracts via SENSEI+Libsim
- Libsim adaptor in SENSEI directs Libsim to render or produce extracts and which are saved to XDB format
AVF-LESLIE in situ extract generation

- Combustion code / Turbulent mixing use case
- Save vorticity isosurface every 5th iteration to FieldView XDB format
- Write groups to partially aggregate extract I/O

For 1/30th to 1/50th the cost of full I/O, we get 10x better temporal sampling with extracts

File I/O (2 writes)  In Situ (20 extracts)  Solver (100 iterations)
Libsim information

- Information about instrumenting a simulation can be found at the following sources:
- Getting Data Into VisIt (https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf)
- VisIt Example Simulations (http://visit.ilight.com/trunk/src/tools/DataManualExamples/Simulations)
- VisIt Wiki (http://www.visitusers.org)
- VisIt Email List (visit-users@email.ornl.gov)
Live demo

- Live demo on virtual machine
  - Running AVF-LESLIE to produce extracts
  - Visualization of extracts in VisIt

Steps:
% cd ~/sc17/demos/visit_libsim
% ./demo.sh 0
% ./demo.sh 1
% ./demo.sh 2
% ./demo.sh 3
% ./demo.sh 4
Computational Monitoring with ParaView Catalyst
ParaView Catalyst information

Functionality:
• Batch and interactive *in situ* analysis and visualization
• In transit workflows done with standalone ParaView, ADIOS and GLEAN
• Generate Catalyst Python scripts to drive *in situ* analysis and visualization output
• Image, data extract and Cinema database outputs
• Instrumented with Fortran, C, C++ and Python based simulation codes

Notable achievements:
• Scaled to 1Mi MPI ranks on ALCF’s Mira BG/Q
• SC16 visualization showcase winner generated animation using Catalyst
• HPCWire Best HPC Visualization Product or Technology
• 2011 (VTK), 2012, 2014 (runner-up), 2016 Editor’s Choice (ParaView)
• 2015 Reader’s Choice – tie (Paraview)
• Used on Cray, BlueGene, SGI, etc. HPC architectures
ParaView Catalyst computational monitoring

Capabilities:

- Connect ParaView server to a running simulation
- ParaView server can be run separately (e.g. on HPC platform) or use the GUI’s built-in server
- Data can be extracted from Catalyst instrumented simulation to ParaView server
- Examine and change in situ analysis and visualization parameters
- Ability to disconnect and reconnect multiple times to a running simulation
- Can pause the simulation to examine results at specific points in the simulation
SENSEI example with Catalyst Python script

<SENSEI>
<analysis type="catalyst" pipeline="pythonscript" filename="catalystlive.py"/>
</SENSEI>
Catalyst Live through Python script
Computational monitoring VM example

- module load sensei/1.1.0-catalyst
- In ~/sc17/demos/paraview_catalyst directory:
  - Run the oscillator with “run_simulation.sh”
  - Run the ParaView GUI with “paraview”
    - Catalyst Menu
      - Connect…
      - Pause Simulation
      - Continue
      - Set Breakpoint
      - Remove Breakpoint
Live *in situ* example

Only transfer requested data from server (simulation run) to client

- Clip1 is already getting extracted

Click on to transfer to client from Catalyst

Use on client to stop transferring to client
Catalyst Live GUI feedback

Three pieces of feedback

• Simulation paused
• Simulation running
• Simulation running with a breakpoint set
ParaView Catalyst online help

ParaView User’s Guide:
- http://www.paraview.org/paraview-guide

ParaView Catalyst User’s Guide:

Email list:
- paraview@paraview.org

Websites:
- http://www.paraview.org
- http://www.paraview.org/in-situ/
- http://www.cinemascience.org/

Doxygen:

Sphinx:

Articles & blog posts:
- http://www.kitware.com/blog/home/post/606
- http://kitware.com/blog/home/post/722
- http://www.kitware.com/blog/home/post/737
- http://www.kitware.com/blog/home/post/752
- http://www.kitware.com/blog/home/post/733
- http://www.kitware.com/blog/home/post/709
Autocorrelation with ADIOS
What is ADIOS 🌿

An extendable framework that allows developers to plug-in

- **I/O methods**: N-to-M, N-to-N, N-to-1, In Situ (aka Staging)
- **Transformations**: Compression, Decompression, Indexing
- **Self describing** data format: ADIOS-BP
- **Indexing/Querying**: MinMax, FastBit, Alacrity

Incorporates the “best” practices in the I/O middleware layer

Released twice a year, now 1.12, under the completely free BSD license

- [https://www.olcf.ornl.gov/center-projects/adios](https://www.olcf.ornl.gov/center-projects/adios)
- [https://github.com/ornladios/ADIOS](https://github.com/ornladios/ADIOS)

Available at ALCF, OLCF, NERSC, CSCS, Tianhe-1,2, Pawsey SC, Ostrava

Applications are supported through OLCF INCITE program

Outreach via on-line manuals, and live tutorials
How to use ADIOS

ADIOS is provided as a library to users; use it like other I/O libraries, except

ADIOS has a simple approach for I/O

• User defines in application source code: “what” and “when”
  – Every process defines what data and when to output
• ADIOS takes care of the “how”

Biggest hurdle for users:

• Forget all of your manual tricks to gain I/O performance on your particular target system and target scale and just say what you want to write/read
• Trust ADIOS to deliver the performance

Performance Portability:

• Write once, perform well anywhere
  – It comes naturally with ADIOS
• ADIOS has many different I/O methods (strategies)
Data management tradeoffs at exascale → to hybrid staging

Explore node layout choices for data management

- Balance of memory size and speed
- Feedback for node designs with NVRAM, larger memory, on-chip NIC
- Network throughput and latency impact on SDMA tasks
- Placement of operations in concert with solver and network topology
Goals of the ADIOS Read API design

Staging I/O
- Insulate the scalable application from the variability inherent in the file system
- Enable the utilization of in situ and in transit analytics and visualization

Same API for reading data from files and from staging

Allow for read optimizations:
- Multiple read operations can be scheduled before performing them
- Allow for blocking and non-blocking reads
- Use generic selections in the read statements instead of describing a bounding box
- Option to let ADIOS deliver data in chunks, with memory allocated inside ADIOS not in user-space
Selections

ADIOS_SELECTION *

adios_selection_boundingbox (int ndim, uint64_t * offsets, uint64_t * readsize)

adios_selection_points (uint64_t ndim, uint64_t npoints, uint64_t * points)

adios_selection_writeblock (int index)

adios_selection_auto (char * hints)
Example of Read API: read a variable step-by-step

```c
int count[] = {10,10,10};
int offs[] = {5,5,5};

P = (double*) malloc (sizeof(double) * count[0] * count[1] * count[2]);
Q = (double*) malloc (sizeof(double) * count[0] * count[1] * count[2]);
ADIOS_SELECTION *sel = adios_select_boundingbox (3, offs, count);
while (fp != NULL) {
    adios_schedule_read (fp, sel, "P", 0, 1, P);
    adios_schedule_read (fp, sel, "Q", 0, 1, Q);
    adios_perform_reads (fp, 1, NULL);  // 1: blocking read
    // P and Q contains the data at this point
    adios_release_step (fp);  // staging method can release this step
    // ... process P and Q, then advance the step
    adios_advance_step (fp, 0, 60.0);
    // 60 sec blocking wait for the next available step
}
// free ADIOS resources
adios_free_selection (sel);
```
N to M reorganization with stage_write

heat transfer + stage_write running together

- Write out 6 time-steps.
- Write from 12 cores, arranged in a 4 x 3 arrangement.
- Read from 3 cores, arranged as 1x3
N to M reorganization with stage_write

$ cd ~/Tutorial/heat_transfer
edit heat_transfer.xml (vi, gedit)
set method to MPI
$ method group="heat" method="MPI"/>

$ mpirun -np 12 ./heat_transfer_adios1 heat 4 3 40 50 6 500
$ bpls -D heat.bp T
double T 6*{150, 160}
step 0:
  block 0: [ 0: 49, 0: 39]
  block 1: [ 0: 49, 40: 79]
  ...
  block 11: [100:149, 120:159]

$ mpirun -np 3 stage_write/stage_write heat.bp h_3.bp BP "" FLEXPATH "" 3
$ bpls -D h_3.bp T
double T 6*{150, 160}
step 0:
  block 0: [ 0:149, 0: 52]
  block 1: [ 0:149, 53:105]
  block 2: [ 0:149, 106:159]
Live demo

- Live demo on virtual machine
SENSEI + Python

SENSEI is a powerful tool to connect simulations to visualization and analysis tools for in situ use. Here we show how to leverage this from a Python based simulation.
SENSEI's Python bindings

- SENSEI based on VTK but we use SWIG (Simple Wrapper Interface Generator) to generate Python bindings.
- VTK's Python wrapper generator, doesn't wrap many methods due to types it doesn't understand. Too purpose specific and inflexible.
- SWIG has extensive C++ compatibility and can be taught to play nice with VTK’s wrapper generator
- Interface (.i) files control what gets wrapped. We wrap everything in SENSEI.
- Bound classes and API in Python have same names as in C++. Code looks and feels very C++ like.
For developers, extending or adding on to SENSEI

vtk.i : A SWIG interface file defining 2 macros:

1. VTK_SWIG_INTEROP(vtk_t)
   - defines typemaps for using VTK wrapped VTK classes in SWIG generated API (tells SWIG how to play nice with VTK)

2. VTK_DERIVED(derived_t)
   - enable SWIG memory management for wrapped classes derived from VTK classes (VTK has unique reference counting implementation)

Pass a VTK class to SENSEI

Pass a SENSEI class to VTK
Integrating SENSEI in a simulation written in Python

1. Compile VTK with Python enabled. Often a part of your chosen back-end. eg Catalyst, Libsim.

2. Compile SENSEI with Python features enabled

3. Select analysis and data adaptors. Use existing or write your own in C++ and wrap them. sensei::VTKDataAdaptor is a good choice.

4. Instrument your simulation, and bridge code. Sets up the data adaptor and invoke analysis periodically.

5. Create any analysis specific run time configurations needed, eg. SENSEI XML files, Catalyst Python scripts, VisIt session files, etc..
**Newton mini-app**

N-body Gravitational Simulation. A single file, <400 lines.

Solves Newton's law of gravitation

Velocity Verlet method

\[ F_i = F_j = G m_i m_j / r_{ij}^2 \]

\[ x_i' = v_i \]

\[ v_i' = F_i / m_i \]
Newton mini-app

- direct solver, $O(N^{**2})$
  - Velocity Verlet
    » second order, symplectic, conserves momentum exactly, time reversible
- the simplest possible code
  - a single file, <400 lines, to better focus on use of SENSEI interface
  - a production quality code could easily be thousands of lines (see NBODY6 ~6K lines)
Instrumenting the simulation

if __name__ == '__main__':
    # parse the command line
    ...

    # set up the initial condition
    n_bodies = args.n_bodies*n_ranks
    ic = uniform_random_ic(n_bodies, -5906.4e9, 5906.4e9, -5906.4e9, 5906.4e9, 10.0e24, 100.0e24, 1.0e3, 10.0e3)
    ids,x,y,z,m,vx,vy,vz,fx,fy,fz = ic.allocate()
    h = args.dt if args.dt else ic.get_time_step()

    # run the sim and analysis
    i = 1
    while i <= args.n_its:
        velocity_verlet(x,y,z,m,vx,vy,vz,fx,fy,fz,h)
        i += 1
Instrumenting the simulation

# set up the initial condition
n_bodies = args.n_bodies*n_ranks
ic = uniform_random_ic(n_bodies, -5906.4e9, -5906.4e9, 5906.4e9, 10.0e24, 100.0e24, 1.0e3, 10.0e3)
ids, x, y, z, m, vx, vy, vz, fx, fy, fz = ic.allocate()
h = args.dt if args.dt else ic.get_time_step()

# create an analysis adaptor(bridge code)
adaptor = analysis_adaptor()
adaptor.initialize(args.analysis, args.analysis_opts)

# run the sim and analysis
adaptor.update(0, 0, ids, x, y, z, m, vx, vy, vz, fx, fy, fz)
i = 1
while i <= args.n_its:
    velocity_verlet(x, y, z, m, vx, vy, vz, fx, fy, fz, h)
    adaptor.update(i, i*h, ids, x, y, z, m, vx, vy, vz, fx, fy, fz)
i += 1

# finish up
adaptor.finalize()
class analysis_adaptor:
    def __init__(self):
        self.DataAdaptor = sensei.VTKDataAdaptor.New()
        self.AnalysisAdaptor = None

    def initialize(self, analysis, args=''
        # select and configure SENSEI analysis adaptor
        ...

    def finalize(self):
        if self.Analysis == 'posthoc':
            self.AnalysisAdaptor.Finalize()

    def update(self, i,t,ids,x,y,z,m,vx,vy,vz,fx, fy,fz):
        # convert simulation data to VTK
        # invoke the analysis
        ...

Our analysis adaptor bridge selects and configures and drives one of a number of SENSEI analysis adaptors

Manages an instance of sensei::VTKDataAdaptor to which we will create and pass VTK objects to
Initializing the in situ analysis

```python
def initialize(self, analysis, args=''):  
    self.Analysis = analysis  
    args = csv_str_to_dict(args)  
    # Libsim  
    if analysis == 'libsim':  
        self.AnalysisAdaptor = sensei.LibsimAnalysisAdaptor.New()  
        self.AnalysisAdaptor.AddPlots('Pseudocolor','ids', False, False,  
                                (0.,0.,0.),(1.,1.,1.),sensei.LibsimImageProperties())  
    # Catalyst  
    elif analysis == 'catalyst':  
        if check_arg(args,'script'):  
            self.AnalysisAdaptor.AddPythonScriptPipeline(args['script'])  
    # VTK I/O  
    elif analysis == 'posthoc':  
        if check_arg(args,'file','newton') and check_arg(args,'dir','./')  
                        and check_arg(args,'mode','0') and check_arg(args,'freq','1'):  
            self.AnalysisAdaptor = sensei.VTKPosthocIO.New()  
            self.AnalysisAdaptor.Initialize(comm, args['dir'],args['file'],  
                                [],['ids','fx','fy','fz','f','vx','vy','vz','v','m'],  
                                int(args['mode']),int(args['freq']))  
    # Configurable  
    elif analysis == 'configurable':  
        if check_arg(args,'config'):  
            self.AnalysisAdaptor.Initialize(comm, args['config'])  
    if self.AnalysisAdaptor is None:  
        status('ERROR: Failed to initialize "%s"
' % (analysis))  
        sys.exit(-1)
```

Select and configure one of the existing SENSEI analysis adaptors from command line arguments

- We are using Libsim, Catalyst, and VTKPosthocIO SENSEI analysis classes directly through the bindings
- SENSEI's Configurable analysis class also exposes these and more and is configurable via an XML file. Eg ADIOS
Invoking in situ back analysis

```python
def update(self, i, t, ids, x, y, z, m, vx, vy, vz, fx, fy, fz):
    status('%5d\n%(' % i) if i > 0 and i % 70 == 0 else None
    status('.')
    # construct VTK a dataset
    node = points_to_polydata(ids, x, y, z, m, vx, vy, vz, fx, fy, fz)
    mb = vtk.vtkMultiBlockDataSet()
    mb.SetNumberOfBlocks(n_ranks)
    mb.SetBlock(rank, node)
    # pass it to the data adaptor
    self.DataAdaptor.SetDataTime(t)
    self.DataAdaptor.SetDataTimeStep(i)
    self.DataAdaptor.SetDataObject(mb)
    # execute the in situ analysis
    self.AnalysisAdaptor.Execute(self.DataAdaptor)
    # free up memory
    self.DataAdaptor.ReleaseData()
```

1. create and pass Multi-block (tree based) dataset to SENSEI data adaptor
   - each rank is responsible for a leaf in the tree
2. pass time and step number to data adaptor
3. invoke the SENSEI analysis adaptor
4. release memory held in the adaptor

Footer
Create the VTK dataset

```python
def points_to_polydata(ids, x, y, z, m, vx, vy, vz, fx, fy, fz):
    nx = len(x)
    # convert simulation to VTK data structures
    v_pts = to_vtk_points(nx, x, y, z)
    v_cells = to_vtk_cells(nx)
    v_ids = to_vtk_scalars(nx, 'ids', ids)
    v_m = to_vtk_scalars(nx, 'm', m)
    v_v, v_mv = to_vtk_vector(nx, 'v', vx, vy, vz)
    v_f, v_mf = to_vtk_vector(nx, 'f', fx, fy, fz)
    # package it all up in a poly data set
    pd = vtk.vtkPolyData()
    pd.SetPoints(v_pts)
    pd.GetPointData().AddArray(v_ids)
    pd.GetPointData().AddArray(v_m)
    pd.GetPointData().AddArray(v_v)
    pd.GetPointData().AddArray(v_mv)
    pd.GetPointData().AddArray(v_f)
    pd.GetPointData().AddArray(v_mf)
    pd.SetVerts(v_cells)
    return pd
```

Strategy

1. create VTK arrays
2. pass them to a VTK dataset

Who owns what?

- VTK uses reference counting. Python does too. Unfortunately they don't talk to each other without some extra code.
- Tell VTK to make a deep copy if the array goes out of scope
def to_vtk_points(nx, x, y, z):
    xyz = np.empty(3*nx, dtype=np.float32)
    xyz[::3] = x[:]
    xyz[1::3] = y[:]
    xyz[2::3] = z[:]
    vxyz = vtknp.numpy_to_vtk(xyz, deep=1)
    vxyz.SetNumberOfComponents(3)
    vxyz.SetNumberOfTuples(nx)
    pts = vtk.vtkPoints()
    pts.SetData(vxyz)
    return pts

def to_vtk_cells(nx):
    cids = np.empty(2*nx, dtype=np.int32)
    cids[::2] = 1
    cids[1::2] = np.arange(0, nx, dtype=np.int32)
    cells = vtk.vtkCellArray()
    cells.SetCells(nx, vtknp.numpy_to_vtk(cids, 
                   deep=1, array_type=vtk.VTK_ID_TYPE))
    return cells

Strategy
1. create an empty array
2. interleave x,y,z components or cell length and point ids
3. pass new array to VTK data structure

TODO – test new zero copy stuff from DG
def to_vtk_scalars(nx, name, s):
    scalar = vtknp.numpy_to_vtk(s, deep=1)
    scalar.SetName(name)
    return scalar

def to_vtk_vector(nx, name, vx, vy, vz):
    # vector in interleaved layout
    vxyz = np.zeros(3*nx, dtype=np.float32)
    vxyz[::3] = vx
    vxyz[1::3] = vy
    vxyz[2::3] = vz
    vector = vtknp.numpy_to_vtk(vxyz, deep=1)
    vector.SetName('v')
    # magnitude
    mv = np.sqrt(vx**2 + vy**2 + vz**2)
    mag = vtknp.numpy_to_vtk(mv, deep=1)
    mag.SetName('mag%s' % (name))
    return vector, mag

Scalars
1. pass new array to VTK data structure

Vectors/Tensors
1. create an empty array
2. interleave components
3. pass new array to VTK data structure

TODO – test new zero copy stuff from DG
Side bar: run time configuration

Adaptors

• SENSEI Configurable analysis. Parses XML and creates and configures one of the other analysis adaptors interfacing to the back-ends (Libsim, Catalyst, ADIOS, custom, etc).

Back-ends

• May expose control API via their SENSEI adaptor. In the Configurable analysis adaptor these are exposed via XML attributes.
• May be scriptable via their own Python bindings adding another layer of control.
• May be configured via "state" or "session" files.

Some adaptors or back-ends may be hard wired to do one thing
In transit demos

Simulation runs in 1st job

End-point runs in 2nd job

FLEXPATH transport moves data across network

XML selects one of these

newton mini-app
VTK data adaptor
Configurable analysis adaptor
ADIOS analysis adaptor
bridge code

ADIOS end-point
ADIOS data adaptor
Configurable analysis adaptor
bridge code

Catalyst analysis adaptor
Libsim analysis adaptor
Histogram analysis adaptor

XML selects one of these
SENSEI in transit configurations

Simulation (this config used with all end-point configs below)

```xml
<sensei>
  <analysis type="adios" filename="newton.bp" method="FLEXPATH" enabled="1" />
  <analysis type="adios" filename="newton.bp" method="DATASPACES" enabled="0" />
  <analysis type="adios" filename="newton.bp" method="MPI" enabled="0" />
</sensei>
```

End-point with Catalyst

```xml
<sensei>
  <analysis type="catalyst" pipeline="pythonscript" filename="catalyst_config.py" enabled="1" />
</sensei>
```

End-point with Libsim

```xml
<sensei>
  <analysis type="libsim" plots="Pseudocolor" plotvars="ids" image-filename="image_%ts"
           image-width="800" image-height="800" slice-project="1" image-format="png" frequency="1" enabled="1"/>
</sensei>
```

End-point with Histogram

```xml
<sensei>
  <analysis type="histogram" array="magv" association="point" bins="20" enabled="1" />
  <analysis type="histogram" array="magf" association="point" bins="10" enabled="0" />
</sensei>
```
Running the in transit demo

Job 1: Simulation
# launch simulation, run 100 iterations
$ ./newton_in_transit.sh 100

Job 2: End-point with Catalyst
# launch end-point configured with Catalyst. Renders and writes images
$ ./catalyst_in_transit.sh

Job 2: End-point with Libsim
# launch end-point configured with Libsim. Renders and writes images
$ ./libsim_in_transit.sh

Job 2: End-point with Histogram
# launch end-point configured with histogram. Computes histograms
$ ./histogram_in_transit.sh
In Situ Costs and Performance
Measuring the cost of \textit{in situ}

\textbf{Two questions:}
How much overhead associated with use of \textit{in situ} methods, \textit{infrastructure} (runtime, memory)?
Does this change with varying concurrency?

\textbf{Additionally:}
\textit{In situ} and in transit configurations  
\textit{In situ} and \textit{post hoc}: end-to-end comparison

U. Ayachit, A. Bauer, E. P. N. Duque, G. Eisenhauer, N. Ferrier, J. Gu, K. E. Jansen, B. Loring,  
Z. Lukic, S. Menon, D. Morozov, P. O’Leary, R. Ranjan, M. Rasquin, C. P. Stone, V. Vishwanath,  
Considerations, and Applications of Extreme-scale In Situ Infrastructures. In Proceedings of  
SC16, November 2016.
Methodology for measuring cost of *in situ*

Miniapplication: data source (next slide)

*In situ* methods
- Histogram computation
- Autocorrelation computation (temporal analysis)
- Extract and render a 2D slice from a 3D volume

*In situ* infrastructures
- VisIt/Libsim
- ParaView/Catalyst
- ADIOS

Measure:
- Runtime and memory footprint
- At varying levels of concurrency
- One-time and recurring

---

**Test Platform**

Cori Phase I at NERSC
Cray XC system
1630 compute nodes
Dual 2.3GHz 16-core Intel Haswell processors
128GB RAM/node

Concurrency levels of tests:
- 812 (~1K)
- 6496 (~6K)
- 45440 (~45K)
Miniapplication - oscillators

Bulk-synchronous parallel computation of periodic, damped oscillators (MPI-based app)

No interprocess communication - entirely analytic, embarrassingly parallel

For $m$ oscillators and per-rank grid size of $N^3$:
- Per-rank memory footprint: $2N^3$
- Per-rank complexity: $mN^3$
## Miniapp configurations – *in situ* methods

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Intention</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original</strong></td>
<td>Miniapp with no SENSEI interface, no I/O. Direct-coupling (subroutine call) to analysis methods Measure runtime/memory with no <em>in situ</em></td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>Miniapp with the SENSEI interface enabled No analysis or I/O Measure overhead of <em>in situ</em> interface in isolation</td>
</tr>
<tr>
<td><strong>Histogram</strong></td>
<td>Miniapp+SENSEI interface+histogram computation No <em>in situ</em> infrastructures Compare performance to <em>Original, Baseline</em></td>
</tr>
<tr>
<td><strong>Autocorrelation</strong></td>
<td>Miniapp+SENSEI interface+autocorrelation computation No <em>in situ</em> infrastructures Compare performance to <em>Original, Baseline</em></td>
</tr>
</tbody>
</table>
## Miniapp configurations – with *in situ* infrastructures

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Intention</th>
</tr>
</thead>
</table>
| Catalyst-slice        | **Miniapp + SENSEI interface + Catalyst**  
Catalyst performs a 2D slice extraction of 3D volume  
Followed by parallel rendering, produces an image  
Compare to *Original, Baseline*                       |
| Libsim-slice          | **Miniapp + SENSEI interface + Libsim**  
Libsim performs a 2D slice extraction of 3D volume  
Followed by parallel rendering, produces an image  
Compare to *Original, Baseline*                       |
| ADIOS-FlexPath        | **Miniapp + SENSEI interface + ADIOS/FlexPath**  
In transit implementation of histogram, autocorrelation,  
Catalyst-slice  
Compare to *Original, Baseline*                       |
Measuring impact of SENSEI interface

Run *Original* and *Baseline* configs, 3 levels of concurrency: 1K, 6K, 45K
- Original: miniapp + subroutine called autocorrelation
- Baseline: miniapp + SENSEI bridge to autocorrelation

Compare runtime (left), memory footprint (right)

No significant difference reflects zero-copy nature of the interface
Comparing *in situ* to *post hoc*

**Post hoc configuration**
- Simulation computes something
- Then writes results to disk
- Post hoc method reads from disk and performs analysis

**In Situ configuration**
- Simulation computes something
- Then *in situ* method computes something
- (No disk I/O involved)

**Post hoc study concurrency**

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Postprocess</th>
</tr>
</thead>
<tbody>
<tr>
<td>812</td>
<td>82</td>
</tr>
<tr>
<td>6496</td>
<td>650</td>
</tr>
<tr>
<td>45440</td>
<td>4545</td>
</tr>
</tbody>
</table>

**Weak-scaling Study**
- Measure post hoc end-to-end cost
  - Sim writes, post hoc reads, processing
- Compare to *in situ* configurations
- Also measure time-to-solution for 100 timesteps
Post hoc: cost of writes

*Baseline* miniapp with the addition of parallel I/O

- VTK I/O, non-collective
- MPI-IO collective is slower (see the paper)
- This is not an I/O study. 😊 We used the fastest I/O approach we could get our hands on.

Weak-scaling: linear increase with problem size

I/O cost is significant at high concurrency

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>1 step</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td>812</td>
<td>2 GB, 0.12s</td>
<td>0.2 TB, 12s</td>
</tr>
<tr>
<td>6496</td>
<td>16 GB, 0.67s</td>
<td>1.6 TB, 67s</td>
</tr>
<tr>
<td>45440</td>
<td>123 GB, 9.05s</td>
<td>12.3 TB, 905s</td>
</tr>
</tbody>
</table>
Post hoc: cost of reads + processing

Time required for reads, processing, and writing (results) for post hoc methods at varying level of concurrency.
In situ: time-to-solution
### Post hoc vs. *in situ* time to solution

<table>
<thead>
<tr>
<th>Configuration (45K)</th>
<th>In Situ</th>
<th>Post hoc: sim + write + read + process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram</td>
<td>~40s</td>
<td>~1230s = ~25s + ~905s + ~300s + (a few secs)</td>
</tr>
<tr>
<td>Autocorrelation</td>
<td>~225s</td>
<td>~2930s = ~25s + ~905s + ~300s + ~1700s</td>
</tr>
<tr>
<td>Catalyst-slice</td>
<td>~80s</td>
<td>~1505s = ~25s + ~905s + ~300s + ~275s</td>
</tr>
</tbody>
</table>

Post hoc fixed costs (at 45K): about 1200s and 12.3 TB disk space

Fewer ranks for analysis processing results in longer analysis runtime (in this 1:10 configuration, which is typical for post hoc use cases)
**In Situ at Scale on Real Science Problems: Computational Fluid Dynamics**

 PHASTA from UC Boulder run on Mira@ANL

- Simulation of realistic geometry tail rudders and active flow control
- Coupled via SENSEI interface to Catalyst-slice, producing an output image
  - Field data, nodal coordinates: zero copy
  - Connectivity data: full copy
- Runs with 256K and 1M MPI ranks
  - 1M run was 4 times larger than any known *in situ* analysis run
  - Key technologies include reduced library size, simplified output specification and static linking using IBM XL compilers for fastest run times
  - In situ overhead: 8.2%, 33%, 13%
    - The 33% traced to zlib/PNG compression on rank 0
Three key performance analysis focus areas

One-time costs: initialization
- Some *in situ* setups may entail non-zero initialization costs, e.g.:
  - Per-rank config file processing

Recurring costs
- Execution time:
  - Different methods require differing amounts of computation
  - Algorithmic complexity at scale
  - *In situ* methods that use reductions
  - *In situ* vs. in transit tradeoffs
- Memory consumption
  - Temporal analysis methods must buffer more data

One-time costs: finalization
- Some *in situ* setups may entail non-trivial initialization costs, e.g.:
  - Global reductions
- Gives insights into ways to optimize
What is the cost of *in situ* processing?

Concern: simulations want to use all available resources, so having an understanding of *in situ* resource utilization is useful.

In other words: In situ infrastructure must play nicely with simulation

Shared resources

• Initialization costs need to be monitored
  - Static build options important as HPC simulation size increases
  - Initialization costs do get amortized

• Finalization costs can be a factor for certain in situ algorithms

• Memory costs can be a factor
  - Shared memory usage for simulation and in situ arrays (“zero copy”)
  - Request only needed arrays through the DataAdaptor’s AddArray() method
  - Some analysis algorithms can require a lot of memory
  - Autocorrelation could potentially need to store full data at each time step. Use autocorrelation window size to reduce the amount of time steps stored
In situ compute

- In situ computation may not need to be done every time step
  - Lower fidelity time stepping output
  - Only when something “interesting” is happening

- Can still reduce output size
  - Image output is fixed size and independent of simulation size
  - Coarsen data extracts
  - Compute summary statistics (e.g. autocorrelation, histogram)
Measuring impact of SENSEI interface

Run *Original* and *Baseline* configs, 3 levels of concurrency: 1K, 6K, 45K

- Original: miniapp + subroutine called autocorrelation
- Baseline: miniapp + SENSEI bridge to autocorrelation

Compare runtime (left), memory footprint (right)

No significant difference reflects zero-copy nature of the interface
Comparing *in situ* to *post hoc*

**Post hoc configuration**
- Simulation computes something
- Then writes results to disk
- Post hoc method reads from disk and performs analysis

**In Situ configuration**
- Simulation computes something
- Then *in situ* method computes something
  - (No disk I/O involved)

### Post hoc study concurrency

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Postprocess</th>
</tr>
</thead>
<tbody>
<tr>
<td>812</td>
<td>82</td>
</tr>
<tr>
<td>6496</td>
<td>650</td>
</tr>
<tr>
<td>45440</td>
<td>4545</td>
</tr>
</tbody>
</table>

### Weak-scaling Study
- Measure post hoc end-to-end cost
  - Sim writes, post hoc reads, processing
- Compare to *in situ* configurations
- Also measure time-to-solution for 100 timesteps
Wrapping Up
SC17 In Situ Tutorial Summary

- Why should you care about *in situ*?
  - Flops >> I/O; *in situ* is a viable approach for coping with this problem
- What *in situ* infrastructures are available?
- What about interfacing my sim code to them?
- What are the performance issues to be thinking about?
Links

- Main page – http://www.sensei-insitu.org/
- Software repo – https://gitlab.kitware.com/sensei/sensei
- VisIt/Libsimg – https://www.visitusers.org/index.php?title=Category:Libsim
- ParaView Catalyst – http://www.paraview.org/in-situ/
Tutorial evaluation

• Was this tutorial useful to you?
• Were there any subjects you’d like to see covered?
  • More of some?
  • Less of others?
• Please provide SC17 with tutorial feedback
  • https://submissions.supercomputing.org/eval.html
• Also, can provide feedback to us at:
  • Andy Bauer: andy.bauer@kitware.com
  • Wes Bethel: ewbethel@lbl.gov
Conclusions and future work

Write once, use everywhere
Easy to add new analysis/frameworks
Understanding data transformation costs
Data Model: supporting arbitrary layouts for connectivity
Bigger runs – current best is 1Mi MPI processes on Mira@ALCF
More examples, tutorials, improved docs, etc.

SENSEI: Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery

This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy, Office of Advanced Scientific Computing Research, under Contract No. DE-AC02-05CH11231, through the grant “Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery,” program managers Dr. Lucy Nowell and Dr. Laura Biven.
Acknowledgment

This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy, Office of Advanced Scientific Computing Research, under Contract No. DE-AC02-05CH11231, through the grant “Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery,” program managers Dr. Lucy Nowell and Dr. Laura Biven.