IN SITU POST-PROCESSING WITH VISIT, LIBSIM AND FIELDVIEW
Overview

- Intelligent Light Overview
- In situ
- VisIt/Libsimm and FieldView
- Workflow success story
- Instrumenting a simulation using Libsim
Intelligent Light

- Established in 1984
  - Nearly three decades in the software & services business
  - FieldView launched in 1990
  - Global Customer Base
  - Truly Solver Independent
  - Multiple CFD practitioners on staff
- We bridge CFD & IT for customers
Serving the CFD Community

• FieldView Software Products
  – Comprehensive scope, from laptops to HPC
    • Visualization, numerical analysis
    • Data management & Automation

• CFD Consulting Services Team
  – Training (on-site or at Intelligent Light)
  – Script development, FieldView customization
  – Workflow Automation & Optimization

• Applied Research Group (ARG)
  – R&D in advanced post-processing & CFD methods
  – Feeds technology into FieldView

Our Mission: To help our customers using CFD to do more with less and make better decisions
Applied Research Group
Strategic R&D for CFD, post-processing & visualization

• Led by Dr. Earl P.N. Duque
• Air Force Research Lab – EPISODE
  – Large Scale Extracts
  – POD
  – Reduced Order Models
• Department of Energy – FieldView-VisIt
  – Open Source Post-Processing Software
  – Commercialized for High End Users

3D Fractal Isosurface from VisIt on BlueGene/Q
Active Deployment / Continuous Development

• DOE has chosen Intelligent Light to commercialize VisIt for engineering use – SBIR Phase 2
  – Binary FieldView XDB libs with VisIt Open source
  – Open-source VisIt engine, with FieldView GUI in client
  – Supported Software

• Libsim already coupled to
  – CREATE/AV – Kestrel
  – FUN3D

• Other significant efforts
  – DOE-OASCR: Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery
    • Infrastructure for ExaScale
    • Team with Lawrence Berkeley National Lab (Lead), Argonne, Georgia Tech & Kitware
  – Lawrence Livermore National Laboratory: Blue Gene Q Port
In Situ

• In situ processing couples data analysis and visualization with the simulation’s execution so both are done in tandem

• There are different forms:
  – In transit – operates on the data when it is staged to another compute resource
  – Tightly coupled – operates on the data in the same address space as the simulation
Case For Using In Situ

- I/O in supercomputers has not kept pace with compute power
  - Some applications report 90% of time spent in I/O [Peterka et al.]
  - Post processing simulation files requires write then read, paying for I/O twice in different application
- In situ reduces I/O costs and makes it feasible to save analyzed results (much smaller) at higher temporal frequency
- In situ provides an opportunity to analyze more of the data rather than just the portion that is typically saved

<table>
<thead>
<tr>
<th>Machine</th>
<th>Year</th>
<th>Writable FLOPS</th>
<th>Whole-System Checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCI Red</td>
<td>1997</td>
<td>0.075%</td>
<td>300 sec</td>
</tr>
<tr>
<td>ASCI Blue Pacific</td>
<td>1998</td>
<td>0.041%</td>
<td>400 sec</td>
</tr>
<tr>
<td>ASCI White</td>
<td>2001</td>
<td>0.026%</td>
<td>480 sec</td>
</tr>
<tr>
<td>ASCI Red Storm</td>
<td>2005</td>
<td>0.035%</td>
<td>660 sec</td>
</tr>
<tr>
<td>ASCI Purple</td>
<td>2005</td>
<td>0.025%</td>
<td>500 sec</td>
</tr>
<tr>
<td>NCCS XT4</td>
<td>2007</td>
<td>0.004%</td>
<td>1400 sec</td>
</tr>
<tr>
<td>Roadrunner</td>
<td>2008</td>
<td>0.005%</td>
<td>480 sec</td>
</tr>
<tr>
<td>NCCS XT5</td>
<td>2008</td>
<td>0.005%</td>
<td>1250 sec</td>
</tr>
<tr>
<td>ASC Sequoia</td>
<td>2012</td>
<td>0.001%</td>
<td>3200 sec</td>
</tr>
</tbody>
</table>
Software to Enable In Situ Workflow

- The simulation is the host program that controls execution and generates data
- VisIt/Libsim forms the in situ infrastructure and its supporting runtime library which creates data products such as XDB extracts for post hoc analysis
- FieldView is the postprocessor and ultimate consumer for data products created in situ
VisIt

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

  - Started Summer 2000
  - R&D 100 award winner 2005
  - Used worldwide
  - Target use cases:
    - Quantitative Analysis
    - Comparative Analysis
    - Data Exploration
    - Visual Debugging
    - Presentation Graphics
  - Intelligent Light’s VisIt work partially supported by DOE Grant SC0007548.

VisIt is made for large problems

- 8 trillion cells
- 98K cores

3D Fractal Dataset on LLNL Vulcan BlueGene/Q (98 billion cells)
Libsim

- Libsim enables simulations to perform data analysis and visualization in situ by applying VisIt algorithms to data from the simulation.
- Libsim supports both interactive connections to running simulations as well as batch-only modes of operation.
- Codes: Ale3D, Mercury, Kull, Nek5000, Kestrel, FUN3D, …
FieldView

- FieldView been on the market for CFD post-processing since 1991
- Graphic User Interface design created by close collaboration with Industry engineers and analysts
- Over 3000 licenses of FieldView in use today throughout the world
- Industries ranging from aerospace and automotive to nuclear engineering, turbomachinery, wind energy and food processing.
- Scales well to 256 nodes per dataset, 30 Bn cell unsteady as high water mark
Put it all together: In Situ Extracts

- VisIt’s export mechanism lets it write XDB files in situ.
- FieldView can efficiently consume the XDB files generated in situ and visualize them.
- The volume data did not have to be written to disk!
- Analyzed results are far smaller, enabling frequent in situ extract dumps.

Extracts overcome in situ’s greatest weakness – that you need to have some idea of what you want to see in the end. Extracts are small enough to save frequently and permit interactive exploration using traditional post-processing methods.
A successful result

• Many timesteps of unsteady data are produced at Maui, Hawaii HPC center
• Post-processing via FieldView (interactive and movie making) done at Pax River, Maryland
• “The processing time of the XDB extracts are a very small fraction… I can run through 100 frames of data in minutes using the extracts compared to hours using the full solution.”
• About 15% of runtime spent on in situ
Libsim puts VisIt in situ

- VisIt provides Libsim, a library that simulations may use to let VisIt connect and access their data
- Share simulation’s arrays to avoid I/O and data movement
Libsim Programming Interface

- Control Interface handles connections and processing commands
- Data Interface handles passing data back to Libsim
- Libsim bindings exist for C, C++, Fortran, Python
  - Fortran functions may have abbreviated names, all lower case
  - Fortran functions take a length parameter after each string parameter
- Libsim allows for a lot of flexibility
  - Interactive vs Batch (or support both)
  - Blocking vs Polling
  - A lot of common patterns can be copied from examples with little modification
VisIt/Libsim Data Model

- **Mesh Types**
  - Structured meshes
    - Rectilinear/Curvilinear
    - I-Blanking
  - Particle meshes
  - Constructive Solid Geometry (CSG) meshes
  - Adaptive Mesh Refinement (AMR) meshes
  - Unstructured & Polyhedral meshes

- **Variables**
  - 1 to N components
  - Zonal and Nodal
  - Enumerated type

- **Materials**
- **Species**
Instrumenting a Simulation

Instrumentation can be performed incrementally in steps

- **Step 1: Initialization**
  - Handle Input
  - Detect input or connection requests from VisIt
  - Tell VisIt when the time step changes
  - Tell VisIt whether it should update plots

- **Step 2: Iteration**
  - Handle Iteration

- **Step 3: Adaptor**
  - Create adaptor functions that expose simulation data to VisIt
  - GetMetaData
  - GetMesh
  - GetVariable
  - GetDomainList

```c
Simulation
main()
{
    Initialization
    /* iterate */
    while(! done)
    {
        Handle Input
        Handle Iteration
        calc_time_step()
    }
}
```

Adaptor Functions
Environment / Setup

Step 1

- Pass options to Libsim, such as path to VisIt
- Libsim needs to know about the environment to load the VisIt runtime library

```c
/* Read environment on rank 0 */
char *env = NULL;
if (par_rank == 0)
    env = VisItGetEnvironment();
/* Pass the environment to all other processors collectively. */
VisItSetupEnvironment2(env);
if (env != NULL) free(env);
VisItSetDirectory("/usr/local/visitdir");
VisItSetOption("-debug 5");
```
Broadcast Functions (Parallel)

Step 1

- Libsim will on occasion need to communicate among parallel ranks
- The application must register functions to be used for communication (keeps MPI dependency out of Libsim)

```fortran
VisItSetBroadcastIntFunction2(bcast_int, NULL);
VisItSetBroadcastStringFunction2(bcast_string, NULL);
```

Fortran Versions

```fortran
integer function visitbroadcastintfunction(value, sender)  
  implicit none
  include "mpif.h"
  integer value, sender, IERR
  call MPI_BCAST(value, 1, MPI_INTEGER, sender, MPI_COMM_WORLD, ierr)
  visitbroadcastintfunction = 0
end
```

```fortran
integer function visitbroadcaststringfunction(str, lstr, sender)  
  implicit none
  include "mpif.h"
  character*8 str
  integer lstr, sender, IERR
  call MPI_BCAST(str, lstr, MPI_CHARACTER, sender, MPI_COMM_WORLD, ierr)
  visitbroadcaststringfunction = 0
end
```
Set Rank, Parallel Flag, and Communicator (Parallel)

Step 1

- Libsim needs to know the rank and size of the process group
- An MPI communicator can be installed for Libsim that can be used to restrict operations to a subset of processors

```c
/* Set parallel flag and rank*/
int par_rank = 0, par_size = 1;
MPI_Comm_rank (MPI_COMM_WORLD, &par_rank);
MPI_Comm_size (MPI_COMM_WORLD, &par_size);
VisItSetParallel(par_size > 1);
VisItSetParallelRank(par_rank);

/* Tell Libsim which MPI communicator to use. */
MPI_Comm comm;
MPI_Comm_dup (MPI_COMM_WORLD, &comm);
VisItSetMPICommunicator((void*)&comm);
```
Batch vs Interactive

Step 1  Libsim permits multiple ways of instrumenting the main loop

**Batch**
- VisIt 2.9.0 extends Libsim with a batch-only support
  - Forces load of VisIt runtime library
  - Does need to listen for interactive connections (simpler to implement)
  - Does not need VisIt clients to set up plots for in situ

**Interactive**
- The simulation must call Libsim periodically to respond to VisIt connection requests or commands
  - Opens a listen socket for inbound connections
  - Writes a sim2 file that VisIt can use to initiate a connection
  - A successful connection causes the VisIt runtime library to be loaded
Batch Initialization

Step 1

• Batch Initialization requires the VisIt runtime library to be loaded explicitly
• Once the runtime is loaded, register data adaptor functions
• Call functions to set up visualization

```c
VisItInitializeRuntime();

VisItSetGetMetaData(SimGetMetaData, NULL);
VisItSetGetMesh(SimGetMesh, NULL);

VisItRestoreSessionFile("/path/to/setup.session");
```
Interactive Initialization

Step 1

- Interactive initialization assumes that code for input processing will be added to the main loop
- VisIt connections are initiated by reading a “.sim2” file created by the simulation on rank 0

```c
if(par_rank == 0)
{
    /* Write out .sim2 file that VisIt uses to connect. */
    VisItInitializeSocketAndDumpSimFile("sim_name",
        "A useful description of the simulation",
        "/path/to/where/sim/was/started",
        NULL, /* reserved */
        NULL, /* reserved */
        NULL /* optional: pass filename for sim2 file */
    );
}
```
Interactive Main Loop

Step 2

- Libsim opens a socket and writes out connection parameters
- VisItDetectInput checks for:
  - Connection request
  - VisIt commands
  - Console input

Initialize

Visualization Request

Complete VisIt Connection

Process VisIt Commands

Process Console Input

Check for convergence, end of loop

Exit

Solve Next Step
int mainloop_interactive(simulation_data *sim)
{
    int visitstate, blocking = 0;
    while(1) {
        if(sim->par_rank == 0) {
            visitstate = VisitDetectInput(blocking, -1);
        }
        switch(visitstate) {
        case 0: /* No input from VisIt, return control to sim. */
            simulate_one_timestep(sim);
            break;
        case 1: /* VisIt is trying to connect to sim. */
            if(VisItAttemptToCompleteConnection() == VISIT_OKAY)
                SetupCallbacks(sim);
            break;
        case 2: /* VisIt wants to tell the engine something. */
            if(!ProcessVisItCommand(sim)) {
                /* Disconnect on an error or closed connection. */
                VisItDisconnect();
            }
            break;
        case 3: /* VisItDetectInput detected console input */
            break;
        default: /* Error */
            return visitstate;
        }
    }
    return 0;
}
Operations During an Iteration

Step 2

- The simulation can tell VisIt that the time step changed so new metadata will be pushed to VisIt
- The simulation can request that all plots be updated with new simulation data
- The simulation can save the plots to an image or it can export them

```c
VisItTimeStepChanged();
VisItUpdatePlots();

char fn[100];
static int count = 0;
sprintf(fn, "image%04d.png", count);
VisItSaveWindow(fn, 800, 800,
               VISIT_IMAGEFORMAT_JPEG);

visit_handle vars = VISIT_INVALID_HANDLE;
VisIt_NameList_alloc(&vars);
VisIt_NameList_addName(vars, "default");
sprintf(fn, "export%04d", count++);
VisItExportDatabase(fn,"FieldViewXDB_1.0", vars);
VisIt_NameList_free(vars);
```
An adaptor comprises a set of functions in the simulation that VisIt calls when it needs data

- Packages your simulation’s data in terms that VisIt can understand
- Return actual pointers to simulation data (*zero copy*)
- Return alternate representation that VisIt can free
- Written in C, C++, Fortran, Python
## Adaptor Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMetaData</td>
<td>Creates a metadata object that tells VisIt the entities advertised from the simulation</td>
</tr>
<tr>
<td>GetMesh</td>
<td>Returns a mesh object that contains the simulation’s mesh coordinates and connectivity</td>
</tr>
<tr>
<td>GetVariable</td>
<td>Returns a data array object containing a simulation field</td>
</tr>
<tr>
<td>GetMaterial</td>
<td>Return a material object describing how the mesh can be decomposed into various materials</td>
</tr>
<tr>
<td>GetSpecies</td>
<td>Return a species object indicating how the mesh’s materials are decomposed into various material species</td>
</tr>
<tr>
<td>GetDomainList</td>
<td>Return a list of domains owned by the current MPI rank</td>
</tr>
</tbody>
</table>

*Additional adaptor functions return data for advanced features*
Registering Adaptor Functions

Step 3

- Adaptor functions need to be registered with Libsim at runtime, once the VisIt runtime library has been loaded
  - Fortran adaptors rely on functions with specific names

```c
visit_handle SimGetMetaData(void *cbdata)
{
    visit_handle md = VISIT_INVALID_HANDLE;
    if (VisIt_SimulationMetaData_alloc(&md) == VISIT_OKAY)
    {
      /* Add items here */
    }
    return md;
}

VisItSetGetMetaData(SimGetMetaData, (void*)sim);
VisItSetGetMesh(SimGetMesh, (void*)sim);
VisItSetGetCurve(SimGetVariable, (void*)sim);
```

Simulation state / User-defined Data
Fortran Adaptors

Step 3

- Fortran simulations rely on specific function names being linked into the simulation to satisfy the Libsim bindings
- All adaptor functions must be provided for a successful link (functions can be minimal)

```fortran
integer function visitgetmesh(domain,name,lname)
implicit none
character*8 name
integer domain, lname
include "visitfortransimV2interface.inc"
visitgetmesh = VISIT_INVALID_HANDLE
end
```
Example GetMetaData Function

Step 3

- GetMetaData returns the inventory of data that will be exposed to VisIt (meshes, scalars, etc)
- Used to populate menus, etc

```c
visit_handle SimGetMetaData(void *cbdata)
{
    visit_handle md = VISIT_INVALID_HANDLE;
    if (VisIt_SimulationMetaData_alloc(&md) == VISIT_OKAY)
    {
        visit_handle mmd = VISIT_INVALID_HANDLE;
        VisIt_SimulationMetaData_setMode(md, VISIT_SIMMODE_RUNNING);
        VisIt_SimulationMetaData_setCycleTime(md, 0, 0.);

        if (VisIt_MeshMetaData_alloc(&mmd) == VISIT_OKAY)
        {
            VisIt_MeshMetaData_setName(mmd, "mesh2d");
            VisIt_MeshMetaData_setMeshType(mmd, VISIT_MESHTYPE_RECTILINEAR);
            VisIt_MeshMetaData_setTopologicalDimension(mmd, 2);
            VisIt_MeshMetaData_setSpatialDimension(mmd, 2);
            VisIt_MeshMetaData_setNumDomains(mmd, 1);
        }
        VisIt_SimulationMetaData_addMesh(md, mmd);
    }
    return md;
}
```
Example GetMesh Function

Step 3

```c
visit_handle
SimGetMesh(int domain, const char *name,
void *cbdata)
{
    SimData_t *sim = (SimData_t*)cbdata;
    visit_handle h = VISIT_INVALID_HANDLE;
    if(strcmp(name, "mesh") == 0) {
        if(VisIt_CurvilinearMesh_alloc(&h) != VISIT_ERROR) {
            visit_handle hxc, hyc;
            VisIt_VariableData_alloc(&hxc);
            VisIt_VariableData_alloc(&hyc);
            VisIt_VariableData_setDataF(hxc, VISIT_OWNER_SIM, 1,
              sim->nx, sim->xc);
            VisIt_VariableData_setDataF(hyc, VISIT_OWNER_SIM, 1,
              sim->ny, sim->yc);
            VisIt_CurvilinearMesh_setCoordsXY(h, hxc, hyc);
        }
        return h;
    }
}
```
### Example GetVariable Function

**Step 3**

```c
visit_handle GetVariable(int domain, char *name, void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;
    SimData_t *sim = (SimData_t *)cbdata;
    if(strcmp(name, "pressure") == 0)
    {
        VisIt_VariableData_alloc(&h);
        VisIt_VariableData_setDataD(h,
            VISIT_OWNER_SIM,
            1, sim->nx*sim->ny,
            sim->pressure);
    }
    return h;
}
```

Pass simulation buffer to Libsim

Simulation Buffers

- `nx=6` xc
- `ny=8` yc
- `pressure`
Example GetDomainList Function

Step 3

- The GetDomainList function returns the indices of the grids that exist on the current MPI rank
- Used in load balancer to constrain work assignments
- Possible to return decompositions for multiple meshes
- Mesh decomposition free to change over time (as in AMR)

```cpp
integer function visitgetdomainlist()
implicit none
include "visitfortransimv2interface.inc"
ccc PARALLEL state common block
    integer par_rank, par_size
    common /PARALLEL/ par_rank, par_size
ccc local vars
    integer h, dl, err

    Tell VisIt that there are as many domains as processors and this processor just has one of them.
    h = VISIT_INVALID_HANDLE
    if(visitdomainlistalloc(h).eq.VISIT_OKAY) then
        if(visitvardataloc(dl).eq.VISIT_OKAY) then
            err = visitvardatseti(dl, VISIT_OWNER_SIM, 1, 1, par_rank)
            if(err == VISIT_OKAY) then
                err = visitdomainlistsetdomains(h, par_size, dl)
            endif
        endif
    endif

visitgetdomainlist = h
```

[c]
Conclusions

• We can leverage VisIt’s data extraction capabilities at scale via Libsim to write XDB data for FieldView from running simulations
• Extracted files are far smaller than volume data and enable more productive post processing on smaller resources
• This should be of interest to people who develop or support solver codes, as it helps people to use their codes
Resources

• Getting Data Into VisIt (https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf)
• VisIt Example Simulations (http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/Simulations)
• VisIt Wiki (http://www.visitusers.org)
• VisIt Email List (visit-users@email.ornl.gov)