Work Log for September

Logan Brown

September 16, 2014

3 Week of September 15th-19th

3.1 Goals for the Week

1. Blueprint adding PSUADE to the DIEL
2. PSUADE Parallelism (Can we control the number of parallel operations?)
3. Hard psuade Data (make some pictures?)
4. Home Matlab Installation (ssh is slow for imaging)

3.2 Progress/Notes

3.2.1 Adding PSUADE (no file writing)

Path to the Function - /PSUADE/Src/DataIo/PsuadeData.cpp

```
/*THIS IS CLIPPED FROM
   ~/PSUADE/Src/DataIO
  And will not work on its own. It's only here for reference.
*/


// ************************************************************************
// A function for reading PSUADE IO data
// ------------------------------------------------------------------------
void PsuadeData::readPsuadeIO(const char *fname)
{
   int    nInputs, nOutputs, *sampleStates, nSamples, ss, ii, idata;
   double *sampleInputs, *sampleOutputs;
   char   lineInput[500], keyword[500];
   FILE   *fIn;

   fIn = fopen(fname, "r");
```

```
assert(fIn != NULL);
fgets(lineInput, 500, fIn);
sscanf(lineInput, "%s", keyword);
while (keyword[0] == '#')
{
   fgets(lineInput, 500, fIn);
   sscanf(lineInput, "%s", keyword);
}
if (!strcmp(keyword, "PSUADE_IO")) /* data is in this section */
{
   fscanf(fIn, "%d %d %d\n", &nInputs, &nOutputs, &nSamples);
   if (nInputs <= 0 || nOutputs <= 0 || nSamples <= 0)
   {
      printf("readPsuadeIO ERROR: first parameters <= 0.\n");
      exit(1);
   }
   sampleInputs  = new double[nInputs*nSamples];
   sampleOutputs = new double[nOutputs*nSamples];
   sampleStates  = new int[nSamples];
   for (ss = 0; ss < nSamples; ss++)
   {
      fscanf(fIn,"%d %d", &idata, &sampleStates[ss]);
      if (idata != (ss+1))
      {
         printf("readPsuadeIO ERROR: incorrect sample no.\n");
         printf("        Incoming/expected sample number = %d %d\n",
                idata, ss+1);
         exit(1);
      }
      if (sampleStates[ss] != 1) sampleStates[ss] = 0;
      for (ii = 0; ii < nInputs; ii++)
         fscanf(fIn,"%lg",&sampleInputs[ss*nInputs+ii]);
      for (ii = 0; ii < nOutputs; ii++)
         fscanf(fIn,"%lg",&sampleOutputs[ss*nOutputs+ii]);
   }
   pInput_.nInputs_        = nInputs;
   pInput_.sampleInputs_   = sampleInputs;
   pOutput_.nOutputs_      = nOutputs;
   pOutput_.sampleOutputs_ = sampleOutputs;
   pOutput_.sampleStates_  = sampleStates;
   pMethod_.nSamples_      = nSamples;
   if (outputLevel_ > 1)
   {
         printf("readPsuadeIO: read sample data completed.\n");
         printf("   nInputs, nOutputs, nSamples = %d %d %d\n", nInputs,
                nOutputs, nSamples);
      }
   }
   else
   {
      if (outputLevel_ > 0)
         printf("readPsuadeIO: PSUADE_IO section absent.\n");
   }
   fclose(fIn);
}
```

Bad news? The file is 3920 lines long.

3.2.2 Blueprint adding PSUADE to the DIEL (With File Writing)

There's two different ways I've thought about running PSUADE on the DIEL. In my opinion, the first option is much better. It captures the high level of control that we have with PSUADE together with the various advantages of the tuple space.

- Tuple dump to psuadeData

Essentially, what we would do is dump the results from running the Ising Model into the PSUADE module, then format the dump from the tuple space into information readable by PSUADE. The psuade data file has this format:

```
PSUADE_IO (Note : inputs not true inputs if pdf ~=U)
2 1 1000
1 1
  2.4006382514204075e-02
  8.5390971001293963e+02
  3.9917210906753051e-03
2 1
  1.3490760269225929e-02
  3.2503036526857431e+02
  2.5058784546057317e-01
.
.
.
998 1
  1.8304650226135109e-02
  3.0831496043089544e+02
  3.9986950286898873e-02
999 1
```

```
  2.8442410000619672e-02
  2.8560985265509680e+02
  1.3828617926678157e-03
1000 1
  4.1479306059982304e-02
  2.8861687094779541e+02
  1.2286195092064936e-03
PSUADE_IO
PSUADE
(contents of the input file)
```

It will either be a function of simple file I/O
Advantages:

1. Once the data has been written to psuadeData, we can reuse it multiple times for multiple visualizations or calculations.

2. We have complete control over how the applications are run

Disadvantages:

1. Requires simulations to store their inputs AND outputs into the tuple space

2. We'll have to alter the simulations (or write a wrapper) to make the simulation write all the appropriate output

3. As we alter the tuple space (specifically, the ability to dump from the tuple space), we may have to alter the conversion

There is another way to run PSUADE in the DIEL

- PSUADE's Way

In the PSUADE input file, there is a line for "driver = (path to simulator)". You can put anything here, but most notably, we could do a C, python, or bash script that calls the DIEL for performing its operations. A shell script would likely be easier to maintain.

Pseudocode for dieldriver.sh:

```
Parse command line arguments (input filename, output filename)

Potentially, forego the files completely,
just use tuple comms to talk to the simulation

invoke the DIEL to run the simulation, using the input from psuade

write the output for psuade
```

Advantages:

1. Works out of the box, aside from writing the small script
2. May be able to use PSUADE gendriver to make a python driver as a starting point

Disadvantages:

1. All the disadvantages of DAKOTA, with none of the advantages of PSUADE
2. Slower
3. Worse on supercomputer
4. Must write a new script for every application

3.2.3 PSUADE Parallelism (Can we control the number of parallel operations?)

3.2.4 Hard psuade Data (make some pictures?)

3.2.5 Home Matlab Installation (ssh is slow for imaging)

3.3 Goals for next Week

1. Future Goal