ASPA - Adaptive Sampling Proxy Application

Milo R. Dorr

Version 1.0
October 25, 2012
LLNL-SM-595112

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

1 Purpose

The purpose of ASPA (Adaptive Sampling Proxy Application) is to enable the evaluation of a technique known as *adaptive sampling* on advanced computer architectures. Adaptive sampling is of interest in simulations involving multiple physical scales, wherein models of individual scales are combined using some form of *scale bridging*. For example, a coarse-scale structural mechanics model might invoke a fine-scale model to obtain constitutive data that only the fine-scale model is capable of computing. Because the higher-fidelity fine-scale model is often very expensive to evaluate (which is why it's not used to solve the entire problem to begin with), it is desirable to perform as few such evaluations as possible. Adaptive sampling [1, 2] attempts to significantly reduce the number of fine-scale evaluations by dynamically constructing a database of fine-scale evaluations and interpolation models. When the response of the fine-scale model is needed at a new point, the database is searched for interpolation models centered at "nearby" points. Assuming that the interpolation models possess error estimators, they can be evaluated to determine if the fine-scale response at the current query point can be obtained to sufficient accuracy simply by interpolation from previously known states. If not, the fine-scale model must be evaluated and the new input/response pair added to the closest interpolation model.

The mechanisms and algorithms needed for adaptive sampling can be abstracted and implemented in a manner that is completely independent of any particular physical application. The ASPA described herein provides such an implementation, combining a type of sparse data interpolation known as *kriging* with an *M-Tree* database strategy for storing the resulting interpolation models. By encapsulating adaptive sampling in this application-independent form, algorithm variations and alternative implementations on new computer architectures can be explored more easily.

2 Requirements

ASPA was developed on an Intel Xeon-based cluster running Red Hat 4.4.6-3. Although we expect that ASPA will likely build and run in any modern Unix/Linux environment, this has not yet been tested on a wide range of platforms. ASPA is written in C++; it was built and tested using either the Gnu (GCC 4.4.6) or Intel (ICC 12.1.5) compilers.

GNU make (`www.gnu.org/software/make`) is required to build the ASPA executable, due to the particular method employed in the Makefile for handling auto-dependencies. ASPA also requires the LAPACK and BLAS (`www.netlib.org/lapack`) linear algebra libraries.

The M-tree database created by ASPA can be read/written from/to a file in HDF5 format (`www.hdfgroup.org/HDF5`), but this is optional. We used HDF5 version 1.8.5 during the development of ASPA.

3 Installation

ASPA is distributed as a tar file named `aspa.tar`. Execution of the command

```
% tar -xf aspa.tar
```

creates a directory named `aspa` in the current working directory, which contains three subdirectories: `doc`, `exec` and `src`.

```
% cd aspa
% ls
doc  exec  src
```

The `doc` directory contains documentation (*i.e.*, the LaTeXfiles that generate this document), the `exec` directory is where the executable is built, and the `src` directory contains the source files (other than the file `main.cc` containing `main()`, which is located in the `exec` directory).

To build ASPA, `cd` to the `exec` directory

```
% cd exec
% ls
Makefile README aspa.inp main.cc point_data.txt value_data.txt
```

and open the file `Makefile` with a text editor. Set the parameters described in the following table to match your environment:

Macro	Meaning	Example
CXX	C++ compiler	CXX = g++
CXXFLAGS	Compiler options	CXXFLAGS = -O2
LAPACK_LOC	LAPACK library location	LAPACK_LOC = /usr/local/lib/lapack
BLAS_LOC	BLAS library location	BLAS_LOC = $(LAPACK_LOC)
HDF5_LOC	HDF5 library location	HDF5_LOC = /usr/local/lib/hdf5-gnu-1.8.5

Depending upon installation defaults, the linker may be able to find the LAPACK and BLAS libraries without explicitly setting the `LAPACK_LOC` and `BLAS_LOC` macros. The simplest way to find out is just to leave those macros undefined and try to build. If unresolved externals are reported by the linker, it will then be necessary to find out where LAPACK is installed on your system (the BLAS library is typically installed there as well) and set the corresponding `Makefile` macros accordingly.

If HDF5 is not available on your system, or you don't care about reading and writing the M-tree database from/to disk, the `HDF5_LOC` macro can be left undefined.

4 Running the sample problem

A sample problem is provided in the `exec` directory, consisting of the file `aspa.inp` containing various algorithmic parameters, the file `point_data.txt` containing input point data, and the file `value_data.txt` containing the corresponding value data. The sample problem is run by executing

```
% aspa point_data.txt value_data.txt
```

at the command line. The parameter file `aspa.inp` is read automatically. The adaptive sampling algorithm is then performed on the set of points contained in `point_data.txt` and the current status of the calculation is printed to `stdout`. The values contained in `value_data.txt` are used to test the accuracy of the kriging interpolation using the associated error estimator. For example, the execution will report such things as

```
Processing point: +40
Adding point :+40
```

indicating that point number 40 was added to the database, or

```
Processing point: +91
Missed point: value Id +0 real value: +3.00787312054833000e+00
  interp. value: +3.00787336609095757e+00 error +2.45542627563821725e-07
```

indicating that the kriging interpolation error tolerance was not met. At the end of the run, a summary such as the following is printed:

```
Number of kriging models +1.03000000000000000e+02
Number of point/value pairs +3.25000000000000000e+02
Level +0
Node   Number entries   Number data leaf nodes
+0 +6 +6
+1 +4 +4
+2 +8 +8
+3 +6 +6
+4 +6 +6
+5 +7 +7
+6 +6 +6
+7 +9 +9
+8 +8 +8
+9 +6 +6
+10 +5 +5
+11 +6 +6
+12 +7 +7
+13 +9 +9
+14 +5 +5
+15 +5 +5
Level +1
Node   Number entries   Number data leaf nodes
+0 +6 +37
+1 +10 +66
Level +2
Node   Number entries   Number data leaf nodes
+0 +2 +103
```

The sample run also creates the text file `kriging_model_centers.txt` containing the locations of the centers of the resulting kriging interpolation models. The generated M-tree database is placed in the subdirectory `kriging_model_database_000000`, unless the executable was configured without HDF5 support, in which case the directory will be empty.

5 Algorithms

5.1 Adaptive sampling

This section contains some additional details about the kriging interpolation algorithm and M-tree database.

5.2 Kriging interpolation

Kriging is a type of scattered data interpolation first developed in the field of geostatistics. Given a set of experimental measurements, the goal is to estimate the response at points for which no measurements exist. Rather than basing the interpolant simply on some measure of the distance to known points, kriging interpolation attempts to also incorporate information about data correlation. If, for example, a large number of experimental observations exist in a small region, it is possible that the data is sufficiently correlated that much of it is redundant. Depending upon the interpolation goals, failure to account for the correlation of clustered data could potentially result in undesirable over-weighting. The estimation of data correlation is therefore conducted as a precursor to kriging interpolation, involving, for example, the creation of a variogram.

Kriging interpolation assumes that, over some region $\Omega \subset \mathbb{R}^D$, the function being interpolated can be expressed as the sum of a regression (a.k.a. trend) model and a stochastic deviation

$$s(\mathbf{x}) = m(\mathbf{x}) + Z(\mathbf{x}), \tag{1}$$

where $Z(\mathbf{x})$ is assumed to have zero mean and covariance of the form

$$Cov\left\{Z(\mathbf{x}, \mathbf{w})\right\} = \sigma^2 R(\mathbf{x}, \mathbf{w}) \tag{2}$$

with σ^2 denoting the process variance. Typically, a linear regression model

$$m(\mathbf{x}) = \mathbf{p}^{\mathbf{T}}(\mathbf{x})\beta \tag{3}$$

is assumed, where the components of $\mathbf{p}(\mathbf{x}) \equiv [\mathbf{p}_1(\mathbf{x}), \mathbf{p}_2(\mathbf{x}), \ldots, \mathbf{p}_D(\mathbf{x})]$ form a basis for the space of linear mappings of Ω into $\mathbb{R}$ and $\beta \in \mathbb{R}^D$ is a vector of regression coefficients. Given known values of s at points $\mathbf{x}_i \in \Omega$, $i = 1, \ldots, N$, the kriging interpolant is defined as

$$\hat{s}(\mathbf{x}) \equiv m(\mathbf{x}) + \sum_{i=1}^{N} \lambda_i(\mathbf{x})(s(\mathbf{x}_i) - m(\mathbf{x}_i)), \quad \mathbf{x} \in \Omega, \tag{4}$$

where the coefficients $\lambda_i(\mathbf{x})$ are chosen to minimize the error variance

$$\begin{aligned} Var(\hat{s}(\mathbf{x}) - s(\mathbf{x})) &= Var\{\hat{s}(\mathbf{x}) - m(\mathbf{x}) - (s(\mathbf{x}) - m(\mathbf{x}))\} \\ &= Var\left\{\sum_{i=1}^{N} \lambda_i(\mathbf{x})(s(\mathbf{x}_i) - m(\mathbf{x}_i))\right\} + Var\left\{s(\mathbf{x}) - m(\mathbf{x})\right\} \\ &\quad - 2Cov\left\{\sum_{i=1}^{N} \lambda_i(\mathbf{x})(s(\mathbf{x}_i) - m(\mathbf{x}_i)), s(\mathbf{x}) - m(\mathbf{x})\right\} \\ &= \sigma^2\left(\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i(\mathbf{x})\lambda_j(\mathbf{x})R(\mathbf{x}_i, \mathbf{x}_j) + 1 - 2\sum_{i=1}^{N} \lambda_i(\mathbf{x})R(\mathbf{x}_i, \mathbf{x})\right), \end{aligned} \tag{5}$$

subject to the constraint

$$\mathbf{p}(\mathbf{x}) - \sum_{i=1}^{N} \lambda_i(\mathbf{x})\mathbf{p}(\mathbf{x_i}) = 0. \tag{6}$$

The condition (6) ensures that the error expectation $E\{\hat{s}(\mathbf{x}) - s(\mathbf{x})\}$ vanishes independently of the regression coefficients β. By introducing Lagrange multipliers corresponding to (6), the resulting unconstrained minimization problem is solved by finding the critical point of the augmented system. The result is [2]

$$\lambda(\mathbf{x}) \equiv [\lambda_1(\mathbf{x}), \lambda_2(\mathbf{x}), \ldots, \lambda_N(\mathbf{x})] = \mathbf{r^T}(\mathbf{x})\mathbf{R}^{-1}, \tag{7}$$

$$\beta \equiv [\beta_1, \beta_2, \ldots, \beta_D] = \left(\mathbf{P^T}\mathbf{R}^{-1}\mathbf{P}\right)^{-1}\mathbf{P^T}\mathbf{R}^{-1}\mathbf{v}, \tag{8}$$

where $\mathbf{v} \equiv [s(\mathbf{x}_1), s(\mathbf{x}_2), \ldots, s(\mathbf{x}_N)]$ is the vector of known data,

$$\mathbf{P} \equiv \begin{bmatrix} p_1(\mathbf{x}_1) & \ldots & p_D(\mathbf{x}_1) \\ \ldots & \ddots & \ldots \\ p_1(\mathbf{x}_N) & \ldots & p_D(\mathbf{x}_N) \end{bmatrix}, \tag{9}$$

$$\mathbf{R} \equiv \begin{bmatrix} 1 & R(\mathbf{x}_1, \mathbf{x}_2) & \ldots & R(\mathbf{x}_1, \mathbf{x}_N) \\ R(\mathbf{x}_2, \mathbf{x}_1) & 1 & \ldots & R(\mathbf{x}_2, \mathbf{x}_N) \\ \ldots & \ldots & \ddots & \ldots \\ R(\mathbf{x}_N, \mathbf{x}_1) & \ldots & R(\mathbf{x}_N, \mathbf{x}_{N-1}) & 1 \end{bmatrix}, \tag{10}$$

and

$$\mathbf{r}(\mathbf{x}) \equiv \begin{bmatrix} R(\mathbf{x}_1, \mathbf{x}) \\ \ldots \\ R(\mathbf{x}_N, \mathbf{x}) \end{bmatrix}. \tag{11}$$

5.3 M-tree database

In the adaptive sampling algorithm described above, the kriging interpolation models are stored in an M-tree database [3]. Objects in an M-tree database are regarded as points in a general metric space for which a positive, symmetric distance function satisfying the triangle

inequality is defined. Using the distance function, M-trees are constructed and modified to enable efficient pruning of the tree during range (find all objects closer than a given distance to a given query object) and nearest-neighbor (find the closest k objects to a given query object) search queries. In the present context, the inputs to the fine-scale model are regarded as a single point, and a distance function is defined between pairs of points using weights that account for scalings of the respective physical quantities. Each kriging model is identified with the centroid of the collection of points defining the interpolation model, for which fine-scale model evaluations have previously been performed. The distance between two kriging models is then defined as the distance between the respective centroids.

An M-tree database provides the ability to dynamically insert and delete objects without the need to periodically rebalance to maintain performance. M-trees are paged, meaning that a parent node of the tree and some number of subtrees are stored in the same physical record (page), such as a group of disk sectors. Objects indexed by an M-tree database (in this case the kriging models) are stored only in leaf nodes. Interior nodes contain so-called routing objects, consisting of a pointer to a subtree of objects within a so-called covering radius of the routing object, as well as the distance between the routing object and its parent.

To enhance performance, it is desirable to maintain well-clustered subtrees. Therefore, new objects are assigned to leaf nodes so as to avoid increasing the covering radius of router objects. If this is not possible, the increase is at least minimized. Since each M-tree node occupies finite resources (limited, for example, by the size of a disk sector or a processor's local memory), as new entries are added to the database nodes, overfull nodes are split creating a new node at the same level. New routing objects must then be created on the parent node in a procedure called *promotion* in [3]. If a promotion causes the parent to overflow, a split is then performed at the parent level, etc. M-trees therefore grow bottom up, adding a new level as the current root node itself splits.

The performance of an M-tree database ultimately depends upon the "policy" for splitting nodes. A split policy specifies the selection of two objects to be promoted to the parent node and the partitioning of objects between the nodes newly created by the split. The choice of policy is guided by heuristics and experimentation. A list of policies is given in [3].

6 Acknowledgments

Most of the code contained in ASPA was extracted from the Adaptive Sampling Framework developed at LLNL by Nathan Barton, Richard Hornung and Jaroslav Knapp.

References

[1] N. R. Barton, J. Knap, A. Arsenlis, R. Becker, R. D. Hornung and D. R. Jefferson, "Embedded Polycrystal Plasticity and Adaptive Sampling," International Journal of Plasticity 24 (2008), pp. 242–266.

[2] J. Knap, N. R. Barton, R. D. Hornung, A. Arsenlis, R. Becker and D. R. Jefferson, "Adaptive Sampling in Hierarchical Simulation," International Journal for Numerical Methods in Engineering 76 (2008), pp. 572–600.

[3] P. Ciaccia, M. Patella and P. Zezula, “M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”, Proceedings of the 23rd Conference on very Large Database (VLB ’97), Athens, Greece, 1997.