Interactive Multidimensional I/O for Glu

Gord Brown and Feng Wang

Contact information: Gord Brown
phone: (604) 592-0401
fax: (604) 721-7292
e-mail: gdbrown@uvic.ca

Abstract: The use of a multidimensional spreadsheet-style browser for displaying and analyzing multidimensional GLU output is described. Some extensions of the work are proposed: dynamic output browsing (in which the browser generates demands directly to the GLU system), and the use of a spreadsheet-style form for input.

Topic: GLU I/O enhancements

Keywords: GLU, input, output, spreadsheet, browser, demand-driven
Interactive Multidimensional I/O for GLU

Gord Brown and Feng Wang

(extended abstract)

1.0 Introduction

The GLU (Granular LUCid) system provides a language for multidimensional programming. However, it does not provide for multidimensional input or output, and it allows only a single dimension (time) to be defined globally. These restrictions are not catastrophic, since it is possible to get around them by using C routines to do I/O, and so on, but they are unfortunate. It would be nice if there were a straightforward, intentionally well-grounded way to do I/O (and, in passing, to allow multiple dimensions to be globally defined).

This abstract briefly describes some work which has been done to handle multidimensional output from GLU programs, and some ideas for extending this work: a more dynamic output browser, and an input browser to go with it.

2.0 Multidimensional Programs, One-dimensional I/O

The problem we want to solve is that of GLU's current limitation to one-dimensional input and output, despite its being a language for multidimensional programming. It is possible to get around this limitation by doing I/O explicitly via C function calls, but it would be nice to avoid this non-intensional (and somewhat tedious) intermediate step.
The matrix transposition program from Multidimensional Programming [AFJW95] is a good example. Matrix transposition is a trivial GLU program:

```plaintext
realign.t,x(realign.x,y(realign.y,t (A)))
  where
dimension t;
  realign.a,b(A) = A @.a #.b;
end
```

The catch is, how do we get the matrix in there, and the transposed one back out again? There's no straightforward, demand-driven, intensional way.

This limitation also has an impact on debugging: it makes it difficult to "see inside" a GLU program, without explicitly writing output via C subroutines or some similar non-intensional approach.

3.0 Browsing Stored Output

The first step in the multidimensional direction is a multidimensional spreadsheet-style browser for GLU output. Feng Wang's MSc. thesis [Wang96] describes a technique for storing multidimensional GLU output in a data "warehouse", then viewing it using a browser. The basic technique for displaying n-dimensional data is to define a 2-dimensional slice of the data by setting n-2 of the dimensions to specific values in the appropriate ranges, then display the remaining two dimensions of the data in a spreadsheet form.

In addition to simply displaying the data, Wang's browser also allows a certain amount of analysis of the data, via spreadsheet-style functions like SUM, AVG, MIN and MAX. It is possible, for example, to display in two dimensions the sums obtained by adding up the data values in a third dimension (or, more simply, to display at the end of a row or column in the spreadsheet a function of the elements of that row).

Data in a multidimensional warehouse is stored as a linear sequence of values (since, after all, we don't have a multidimensional filesystem). The first part of the file describes the format of the data; the second part is the data itself. Data are stored in the file in the same order that elements of a multidimensional array in C are stored in memory.
This approach provides a workable way to view multidimensional output generated by GLU programs. One limitation of this approach is that it is not very flexible. To look at a slightly different data set, it is necessary to go back to the GLU system, generate a new data warehouse, run the browser on the new data, and so on. The other is that the GLU program has to explicitly write out the data set to the warehouse (i.e. you can’t examine the output of a GLU program unless it’s been specifically designed for it). The next section describes a more dynamic approach (as yet unimplemented), which would remove these limitations.

4.0 Browsing Output on Demand

The idea is to display output in a way that complements the eduction model of evaluation used in GLU. The basic technique is found in the previous section: display multidimensional output in a multidimensional browser. The additional step is to attach the browser to GLU, so that it actually generates demands to GLU to populate the output browser, skipping the intermediate “warehouse” step.

Currently, GLU is driven by a “demand generator” which generates successive demands for the result at time = 0, time = 1, and so on. We would remove that demand generator, replacing it with the interactive browser. When the user requested some set of output, the browser would generate the appropriate demands in the GLU system, and populate the output browser based on the results of those demands.

There don’t appear to be any fundamental issues preventing this technique from working. There are technical issues to deal with: determining the rank of the output object, allowing multiple global dimensions, and so on. How the user should request a particular output range of a particular program is another question. Further work will reveal whether these issues are showstoppers or not (but it doesn’t look like it).

5.0 Demand-driven Input

Since we’re generating output on demand, it seems only reasonable that we should do something analogous for input. The current GLU system requests input whenever it is necessary to compute the requested result. For large input data sets, this approach could get tedious. The suggested alternative is to analyze the program (based on the current request we’re trying to satisfy), determine what input is necessary to compute the current request, and open a (multidimensional) spreadsheet window for the
user to enter the data. Once the data are entered, the program can proceed
to compute the requested output. As an added feature for ease of use, it
would be possible, once the input spreadsheet was opened, to populate it
from a data file (say, from a multidimensional warehouse in the format
described in [Wang96]). This would allow straightforward input and sub-
sequent analysis of experimental data, or other externally-generated data
sets.

The question here is how to determine what input will be necessary to
compute some given output. One approach that comes to mind is some
kind of lazy partial evaluation which would evaluate the request until it
needed some input, add that need to a list, continue evaluating the rest of
the expression, add whatever other inputs it needs to the list, and so on,
then create the input form based on the list of unfilled needs.

A more elegant approach might be to analyse the GLU program itself, to
determine what data are needed. For example, knowing the semantics of
"fby", one could determine whether the value at index X depends on the
value at index X-1, and so on, and thereby, via induction, determine that
data for indices 0..n are needed. Similar rules might be derivable for
other language constructs.

Clearly, more thought is needed to determine how best to approach this
problem. Still, the basic idea seems straightforward enough. Virtually all
of what is necessary to solve this problem must already be happening
somewhere in the GLU system anyway, just to evaluate programs. The
rest is implementation details, right?

6.0 Summary

The basic claim is that a flexible mechanism for multidimensional input
and output would be a useful addition to GLU. It would expose the multi-
dimensional nature of GLU programs to the outside, allowing a much
closer link between what’s going on inside and what the programmer
sees. Being able to do this without having to code for it explicitly in the
GLU program (or its C procedures) would be convenient. One approach
to achieving this goal is multidimensional input and output spreadsheet-
style browsers.
7.0 Bibliography
