COMP 621 Suggested Projects - Winter 2014

Due Dates:

- Project descriptions available and signups, Wednesday Feb 19 to Monday Feb 24
- Project proposal due, Friday Feb 28 (pdf by email) (extension to Monday, March 3)
- Project update #1 due, Friday March 14 (pdf by email)
- Project update #2 due, Thursday March 28 (pdf by e-mail)
- Project presentations start: Monday March 31
- Project presentations end: Wednesday April 9
- Project reports due, Wednesday April 23 (pdf by e-mail, zipped source)

1 Introduction

The purpose of the final project is to give you a chance to focus on a specific problem, and to work out the solution to this problem in detail. Your project should give you the opportunity to review the current literature in your chosen area, to implement either the ideas you read about or to implement your own improvements on those ideas, and to evaluate the effectiveness of the ideas. You are encouraged to try to develop improvements over the current state-of-the-art. Even if your ideas do not turn out better, it is still worth trying (after all, that is what research is all about). It is worth noting that publishing novel work done in this course is not out of the question.

I have listed the various possible project areas below. Students should work individually. Projects must be done specifically for this course and should not contain material used for other course projects.

The projects will be given out on a first-come, first-served basis. You may sign-up for your selected project in the spreadsheet announced to the google group. Two students may choose the same project, but they must do individual projects using complementary solutions.

You are also free to design your own project, or to change a suggested project to more suit your interests. If you wish to modify or design your own project, please contact the instructor to discuss the project.

Your project proposal should be done using the document outline in:

`http://www.sable.mcgill.ca/~hendren/621/Docs`

Latex documents are preferred, but not absolutely necessary. In any case, you should follow the overall organization given in `http://www.sable.mcgill.ca/~hendren/621/Docs/prop621.tex`.

Your final report should follow the outline defined in `http://www.sable.mcgill.ca/~hendren/rep621.tex`.

Please submit .pdf files of the form prop621_yourname.pdf and rep621_yourname.pdf.

You are also asked to submit two project updates. These should be short (approx 1 page) .pdf documents which summarize the current state of your project, discusses the plan for the next two weeks, and discusses any changes to your project plan. These documents should be called update1_yourname.pdf and update2_yourname.pdf.

2 Suggested Projects

2.1 Velociraptor projects

Mentor Raul Garg (rahul.garg@mail.mcgill.ca)

The McLab group has developed a framework for generating GPU code from languages like Matlab that support high-level array operations. The main idea is that a host compiler selects a section of code that is suitable for execution on the GPU and then converts it to a high-level IR called VRIR. Velociraptor then takes care of generating OpenCL code so that it can be executed on the GPU. More information on the Velociraptor project can be found at: `http://www.sable.mcgill.ca/mclab/gpu/`.

There are many possible extensions or uses of Velociraptor that are suitable for COMP 621 course projects. Here are three possibilities.

1. Creating an R to VRIR compiler. There is an open source interpreter for the language R, this project would involve modifying the interpreter so that select pieces of the code were translated to VRIR, and then compiled on-the-fly by Velociraptor to an OpenCL kernel, and the interpreter would then call this kernel code instead of interpreting the original R code. For a course project one would focus on doing this for a small subset of R.

2. Integrating VRIR into Octave. Octave is an open source implementation of Matlab that is mostly interpreted (although the most recent version has a simple JIT compiler). In this project you would modify Octave to generate VRIR for key segments of code and then use Velociraptor to generate GPU code, which would then be called by the interpreter.

3. Adding sparse matrix support to VRIR. VRIR currently only supports dense matrices. However, sparse matrices are also important in numerical computing. This project would examine sparse matrices in several high-level languages (including Matlab) and come up with an extension to VRIR that would support sparse matrices.

2.2 Sparse Matrix optimization for MATLAB

Most of our work to date has concentrated on optimizing dense matrices. However, MATLAB also supports sparse matrices which can give good performance improvements (when used correctly). See `http://www.mathworks.com/help/pdf_doc/otherdocs/simax.pdf`.

This project would be to:

- understand and identify under what circumstances sparse matrices should be used
- understand how the MATLAB source code should be written so as to maximize the benefit of using sparse matrices
- identify a collection of at least 4-5 benchmarks (computation kernels) that use sparse matrices. You may find the aspect described in the AOSD 10 paper to be useful here. See `http://www.sable.mcgill.ca/mclab/aspectmatlab/`
- identify analyses and transformations that are useful for transforming MATLAB programs so as to benefit from sparse matrices
- as much as possible, implement the analyses and transformations
- study the performance benefit on the benchmarks

2.3 Rewrite loops to calls to library routines or vector statements

Mentor: Laurie (hendren@cs.mcgill.ca)

Problem Statement: MATLAB is an interactive programming environment for scientific computing. It is designed for vector and matrix computation. Naively written MATLAB code can suffer a performance slowdown. One way to address the performance problem of high-level domain-specic languages, such as Octave or MATLAB is to use optimized library routines instead performing operations on vectors and matrices iteratively(using loops).

While it is tempting for those with experience in traditional programming languages to use explicit loop constructs for mathematical operations, this temptation should be resisted strenuously. For example, instead of:

```
port_val = 0;
 for j = 1:n
      port_val = port_val + (holdings(j) * prices(j));
 end
```

Instead:

```
 port_val = holdings*prices;
     or
 port_val = mtimes(holdings,prices);
```

The latter is more succinct, far clearer, and will run much faster if the appropriate library calls are made by the MATLAB system. The goal of this project is to develop one or more analyses that detect loops that correspond to known and efficient library routines, or to equivalent vectorized operations and to automatically rewrite the loops to higher-level code that will be compiled to an efficient library routine.

See the paper at `http://portal.acm.org/citation.cfm?id=1267941` for one relevant reference.

2.4 Design and implement an analysis to identify for loops in MATLAB that can be safely transformed to parfor loops

Mentor: Vineet Kumar (vineet.kumar@mail.mcgill.ca)

In the MiX10 compiler (and other parallel backends) we can take advantage of target language's superior parallel performance compared to MATLAB, for the parfor loop construct. However, for various reasons, many MATLAB programmers use a for loop where they can actually take advantage of a parfor loop. It would be nice to identify such for loops and automatically transform them to parfor loops . You can read details about parfor at `http://www.mathworks.com/help/distcomp/getting-started-with-parfor.html#brdqn6p-1`.

This analysis can be made cleverer by not just looking what's inside the loop but also analysing code outside the for loop or may be by making some safe transformations to the loop ; for example , The code below cannot be transformed to using For loop because the value of 'd' after the loop depends on order of execution of iterations. a parfor loop,instead of throwing an error, will force 'd' to retain its value before the loop . Now, if we know that the definition of 'd' inside the loop is never used outside the loop, we can transform it to parfor.

```
clear A
d = 0; i = 0;
parfor i = 1:4
   d = i*2;
   A(i) = d;
```

```
end
A
d
```

You can read more about reduction variables at `http://www.mathworks.com/help/distcomp/advanced-topics.html`.

One might also consider developing an aspect which could observe loops and find those that behave in a way in which is suitable for parfor loops, and then use that information to guide the programmer to convert the loops to parfor loops.

2.5 Improving AspectMatlab Performance or Functionality

Mentors are Laurie (hendren@cs.mcgill.ca) and Andrew (abodzay@gmail.com)

AspectMatlab can be improved either by improving its performance or by adding new functionality.

In Assignment 1 we have seen that weaving aspects can introduce significant runtime overheads. Thus, one potential project would be to design new code generation strategies or optimizations, in order to reduce the overheads. You would start with the most recent version of AspectMatlab, which fixes many of the functionality problems encountered in Assignment 1. Previously written aspects, plus aspects used in Assignment #1, can be used for benchmarks.

The second option would be to improve AspectMatlab by adding new functionality. This could include new patterns or pattern operators, or it could include matching traces (for example, a simple version of tracematches as implemented for AspectJ).

2.6 Domain-specific language for specifying dataflow analyses

Mentor: Ismail (isbadawi@gmail.com)

In assignment #2, you are exposed to McSAF, McLab's static analysis framework. While McSAF makes the job of writing a dataflow analysis simpler, there is still a fair amount of boilerplate code involved. Also, McSAF is only used by McLab's Java-based tools; the McVM project, implemented in C++, does not use it. Actually the McVM project doesn't have an analysis framework (although one is planned); all of its analyses are implemented from scratch, and contain lots of duplicate code. Even if it did, we would still have to specify analyses twice; once for McSAF, once for McVM. Even now we have multiple implementations of e.g. reaching definitions and live variables analysis.

The purpose of this project is to address these issues with an extra level of abstraction, by designing and implementing a small declarative language for specifying dataflow analyses. An analysis could be specified at a relatively high-level as seen in class, by describing the data, the direction, the merging operation, and providing dataflow equations for different kinds of AST nodes. Coming up with a good natural syntax for this is part the project. A compiler would take such a description as input and generate code for the analysis (assuming the existence of an analysis framework to code against). A McSAF backend should be provided, but the compiler should be extensible enough to allow us to add (for example) a backend for McVM's analysis framework once it's ready.

This is probably a hard problem in general. We are restricting ourselves to analyses operating on the Matlab (or Natlab) AST, not any arbitrary tree. Also, we are assuming that a suitable (i.e. McSAF-like) analysis framework exists for the targets we generate code for.

2.7 Array Growth Optimization in Matlab

Matlab is a popular programming language for scientific computing. Several features of the language contribute to its appeal. Two of these features are dynamic typing and dynamic array growth. The two features contribute to the flexibility of the Matlab language and can aid rapid program development. However, dynamic array growth presents performance challenges. Consider the following Matlab function.

```
function grow0()
  s = 0;
  for i=1:10
    a(i) = sin(i);
    s = s + a(i);
  end
  disp('Sum of ');
  disp(a);
  disp(['is: ', num2str(s)]);
end
```

At each iteration of the loop, array a, which is undefined before the loop, will be expanded by 1 and the sin of the current i is assigned to the location in a indexed by i. This can cause frequent re-allocation of the array and can be a source of significant runtime overhead. After the execution of the loop, array a will contain 10 elements. Array concatenation operation can also grow an array dynamically. For instance,

```
a=0;
for i=1:10
  a = [a, i];
end
disp(a);
```

Array `a` grows with the loop's iterations. At each iteration, the current value of i is appended to the array so that the array grows in length by 1. After the execution of the loop, array a has 11 elements: 0 to 10. Other forms of dynamic array growth in Matlab exist. Multidimensional arrays can also grow dynamically.

2.7.1 Problem Statement

The problem to be addressed is the development an array growth optimization. This project will involve:

- an investigation of different forms of array growth in Matlab;
- the development of analyses and transformations that can detect dynamic array growths;
- a technique for estimating the optimal size of a growing array; and
- the transformation that pre-allocates arrays using the results of the analyses.

2.7.2 Potential Optimization Benefits

The ability to estimate the optimal size of an array statically can enable the pre- allocation of the array before a loop, which can lead to a significant performance improvement. It can also simplify the elimination of redundant array bound checks generated to ensure that array accesses are within the bounds of the array.

3 Propose a Project

You may also propose a new project that is not listed here. If you choose something else, it should have some element of ”program” analysis in it, although the ”program” being analyzed could be something from a different domain such as models or games.

If you are proposing your own project, please e-mail the instructor (hendren@cs.mcgill.ca) with a brief project proposal similar to the ones above, and add your project title to the list below.