Notes for instructors

Libby Shoop

Assignment file names

The suggested starting points for assignments are as follows, with corresponding pdf files:

1. *SystemsCourseBoids.md* -- this is intended for students in core courses who are studying C/C++ or have some experience with C/C++. I called it SystemsCourse because that is where I would use it in our curriculum, but other contexts may work for you. Corresponding code is in the **OMPpsPlot** folder. A sequential version with Makefile is provided as a starting point.

2. *PDCCourseOpenACC-X.md* -- this is intended for an advanced course where students have practiced OpenMP and are studying OpenACC. I labeled it PDCCourse because that s where I have used it. It emphasizes the OpenACC additions that can be made to run on the GPU, while keeping the original X display code from FLake. Corresponding code is in the **GPUpsPlot** folder. A sequential version with Makefile is provided as a starting point; two new versions can be made for: OpenMP and OpenACC.

3. *PDCCourse-TSGL.md* -- like *PDCCourseOpenACC-X.md*, this is intended for an advanced course where students are studying OpenACC, but also requires that they obtain or have access to an installation of TSGL (see below for link). In this case, some familiarity with C++ is needed. Corresponding code is in the **tsgl** folder. Three different versions of the program can be developed from the original sequential version -- the Makefile has provisions for building three different versions for: OpenMP, OpenACC multicore, OpenACC GPU.

For all of these assignments, the code for displaying the boids can be a black box for the students. The concept is to concentrate on the code for updating the position of each boid at each time step and how that can be parallelized.

Motivation

This video segment from a PBS show is quite interesting to explain the basic algorithm, using flocks of starlings. Although the code we have isn't exactly the same, it is quite close to how it has been done by scientists.

Readings and original code

The original paper by Craig Reynolds [1] is quite approachable and students may find it interesting. I recommend providing a copy for them (I can't include it here for copyright reasons). He is extremely enthusiastic about modeling nature and how he did it.

The C code written by Gary Flake can be found here: https://github.com/gwf/CBofN. It has many different examples. The boids code contained several global array variables. For the versions that use his psplot code

(OMPpsPlot, GPUpsPlot folders), I updated the code to remove these and place the parameters needed for a run into a single C struct with default values set.

Information from Flake's book

I have included an excerpt of chapter 16.3 from Flake's book as a pdf file, which I was able to download from an EBSCO electronic copy from my library. I believe under fair use copyright I am able to use this small excerpt for courses at my institution. **If you are able to get this yourself, I would encourage it so that we properly follow copyright laws.**

In addition, here is information is taken from Gary Flake's book about the various options of the program:

```
"The boids program has many options, but most of them should be familiar by
now. There are four options used to specify the radius in which the four boid
rules are active, and four corresponding weighting factors. The angle option
is used to specify the number of viewing degrees that the boids possess. The
boids can see only other bolds that fall within a cone of the specified
number of degrees in front of them. The vangle option has a similar meaning,
but is used only for the view rule. The dt and ddt options specify time
increment and momentum factors, and minv is used to specify the minimum speed
that the boids can fly.

For display purposes, len can be used to specify the length of the boids in
pixels. And finally, psdump is used to tell the program to emit a PostScript
image of
the boids at the end of the simulation.

A boid is displayed as a flying arrow. The head of the arrow defines an arc
that swings from the left edge, to the front, and then to the right edge of
the head. This arc is the same as the viewing angle; thus, if you set the
viewing angle to something large, say 320 degrees, then the head will make a
very sharp point. With a viewing  angle of 180 degrees, the head will be a
flat line."
```

A stipulation: need X graphics for psPlot versions, TSGL for third version

Flake's code included files to enable a graphical X window display of the boids as arrows moving around the screen. To run this version of the code from a linux server in our department, I have instructions for my students about how to use XQuartz on a Mac and use ssh -X to connect, or from Windows I suggest that they use MobaXterm to connect and run the code over ssh. There are certainly other ways to accomplish this, such as running Windows subsystem for linux on your own Windows laptop, or providing linux lab machines.

You will need the X11 development version installed under linux. On Ubuntu, it's installed like this:

```
    sudo apt install libx11-dev
```

If you want to have students try the TSGL versions, that will need to be installed on the machine it is built on. Install instructions here: https://github.com/Calvin-CS/TSGL/wiki/Installing-TSGL

If the machine is remote, the information about remote display over ssh applies to the TSGL version.

We have tested the code versions on Ubuntu linux and Windows Subsystem for Linux (WSL).

OpenACC compiler notes

The compiler flags may be different for the version of the OpenACC compilers from NVIDA you have installed. We have used ones for the SDK from Fall of 2023.

The computations on a GPU give slightly different results from the CPU version. We surmise that this is because we are using compilers with different math libraries and that the floating point operations are implemented differently in GPU hardware. This becomes an excellent learning point for advanced students.

Assignments ask for reports

When I teach PDC, I prefer to emphasize writing reports as an important part of each assignment. In many cases, changing the code to be parallelized is just the first part of the work. Students should learn how to determine when it is worth it to use the parallel version by measuring speedup and examining scalability from experiments that they run. You will see this emphasis in each assignment. Use as you see fit.

Scalability

When I teach scalability, I always tell students to try their code with a certain *problem size* and try it at 1, 2, 4, and 8 threads on a multicore machine. Adding 16 almost always does not help, but for some problems and hardware it does. If the code is strongly scalable (i.e. has good speedup), the time should roughly cut in half each time. This depends on the problem size. In the case of the boids, a small number of them is not worth parallelizing. Students should try to determine what sizes scale- some stop after 2 or 4 threads, some larger sizes scale up through 8 threads.

I did not include the above information in the systems course level assignment, but you could if don't want to discuss scalability in class before using this assignment.

For the PDC course level assignment, I spend time having students practice running scripts to run tests that form experiments for gathering data for strong scalability speedup plots and for weak scalability plots. I have sample scripts and spreadsheets they work from. The plots can then be used as evidence of the scalability of the OpenMP or multicore solutions. If you want more information about this, please contact me (shoop at macalester edu).

For OpenACC and GPU solutions, traditional speedup calculations don't really apply. I try to have an activity and a brief lecture about comparing a serial to a CPU threaded to a GPU version by observing how many times faster a GPU one is to the others. For this example, it's not always a slam dunk that GPU is remarkably faster, likely because of the data movement time (the time for the number of calculations needs to overshadow this). I tried to give hints in the assignments for advanced students about how to experiment and find when it is useful to use the GPU.

Necessary Background

I have referred to material that I have written in two interactive textbooks that you can find on learnpdg.org.

Parts of the following information is included in each assignment document. If you have other materials, you could add yours.

You need to have studied some examples from OpenMP to be able to apply them to this situation. One reference is the PDC for Beginners book, chapter 1, where you can see some OpenMP examples in action.

Another is the full set of pattern examples in Chapter 2 of the Intermediate PDC book.

To consider the scalability of your parallel solution, you might want to read about how this is determined in PDC programs by reading PDC for Beginners book, chapter 0, section 3.

To complete an OpenACC version for the GPU, you should study and practice the code examples in Chapters 7 and 8 of the Intermediate PDC book. It is also helpful to consider the CUDA GPU Programming model described in Chapter 4, section 2 of the PDC for Beginners book. The OpenACC compiler creates CUDA code, and the programming model of 1 thread per array data element computation applies here.

For a Challenge

After advanced students have completed the primary tasks for either the X version or the TSGL version of this assignment, you could suggest that they experiment further with two aspects of random number generation that are used.

1. The current version creates new random positions for the boids using a sequential random number generator with a seed of 0. Students could create a version that either uses this constant seed or uses a command line argument to use a runtime varying seed based on the time. Also, this process of creating them could be parallelized if a proper parallel PRNG is used. There are examples of this in Chapter 3 of my Intermediate PDC book, but it does require that you have a particular library installed for this called trng.

2. Flake had introduced some random movement of each boid as it computed its new heading. Since the code works well without this, I commented this out to make the parallelization simpler. However, if you want to include proper use of a parallel random number generator library like trng, you could suggest to students that they could enhance the computation of the hnew heading by adding this feature back in. For OpenACC, I have given students an example of how this can be done using CUDA streaming within an OpenACC verion of code. I intend to add this to the Intermediate PDC book, but it isn't there as of Jan. 2024. Contact me if you are interested in this.

References

[1] Reynolds, Craig W. (1987) Flocks, herds and schools: A distributed behavioral model. Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 25--34.