Summer School Parallel Programming
From Multi-core to Many-core and beyond

Day 4

Declarative Array Programming with Single Assignment C (SAC)

Miguel Sousa Diogo
Miguel.SousaDiogo@student.uva.nl

Roeland Douma
R.J.Douma@uva.nl

Clemens Grelck
C.Grelck@uva.nl

July 12, 2012

UNIVERSITY OF AMSTERDAM

VU UNIVERSITY AMSTERDAM

1 SAC Setup on the DAS-4

As expected we again work on the DAS-4 cluster. For more information go to `http://www.cs.vu.nl/das4/`. And again we will be using PPM images which can be viewed using the *display* command.

The bleeding edge version of SAC is available on the DAS-4 and you can make easy use of it by adding the following line to your *.bashrc*:

```
source /home/sac/sacrc
```

After this logout and login again or just run line in your current shell to set the right environment variables. The skeleton code, the SAC tutorial, a VI syntax file and the example images are available on the DAS-4 (/home/sac/assignment4.tgz) and on
`http://student.science.uva.nl/~bakkerr/multimoore/assignment4.tgz`.

2 Compiling and running SAC code

2.1 For sequential execution

Compilation should just be done on the head node. This is very easy:

```
sac2c <program>.sac -o <program> -O3
```

The SAC compiler takes a while to run all the optimizations and to generate *C* code that is then compiled by your favourite backend compiler (the default is GCC). The `-o` option allows you to specify the name of the resulting binary file; the `-O3` option instructs the backend C compiler to use maximum optimisation. Whenever you are after performance the latter is obviously very important. As a research compiler, `sac2c` has many, many options that allow you to control all aspects of the compilation process. For an idea, try `sac2c -help`.

To run your program on the DAS-4 do the following:

```
prun -v -np 1 <program> <arguments>
```

2.2 For multi-core execution

To make use of multiple cores there are only a few steps required in SAC. First, you need the `-mt` option to compile a SAC program for multithreaded execution:

```
sac2c -mt <program>.sac -o <program> -O3
```

Now your code is ready for multithreaded execution! But of course we still need to tell the runtime system how many threads we want to use so when starting your program we have to provide this as a command line argument to the SAC binary program.

```
prun -v -np 1 <program> -mt <number of threads> <arguments>
```

It is that easy!

2.3 For many-core execution

SAC recently got many-core support by using CUDA as a backend. It works by turning with-loops into kernels. As you may recall from the CUDA exercise, you can't have function calls in the kernels, so you'll have to avoid using those. Otherwise, SAC won't be able to parallelize them and they'll run sequentially on the CPU, thus killing performance. To run your program on a GPU, there are again only a few steps required. To compile, run

```
sac2c -target cuda64 <program>.sac -o <program> <optimizations>
```

And just like that, your code now runs on the GPU! There are a number of optional optimization flags which can improve your runtimes, these are:

- `-dolao` lifts the expensive CUDA memory allocations out of loops
- `-doexpar` expands the parallel regions by creating kernels for the non-parallel sections, thus avoiding copying arrays from device to host
- `-doshr` enables the use of CUDA shared memory
- `-docoal` improves coalescing of memory accesses

Of course, we can now only run these SAC programs if a GPU is available. To select the GPU-enabled nodes on the DAS-4, you need the following commandline argument.

```
prun -v -np 1 -native '-l gpu=GTX480' <program> <arguments>
```

Note: There are a few outstanding problems with the CUDA backend of SAC. The most noticeable one is that you cannot use the `printf` or `fprintf` functions. You have to replace these with alternatives such as `print`. If your SAC program does not compile for CUDA, try disabling some optimizations or using fixed sizes for the arrays.

3 Documentation

Besides the lecture slides there is a comprehensive tutorial on SAC available from the SAC homepage:
`http://www.sac-home.org/publications/tutorial.pdf`.

4 Assignments

4.1 Play around

We do not want to copy/paste the tutorial here. So, browse through the document, in particular the introductory parts, as you like and do some of the examples and exercises in there. This will help you to get a feeling for the language.

4.2 Image Filters

The provided skeleton code already implements the *rotate90* filter, which rotates an image by 90 degrees (surprise!). The other filters are described in the skeleton file. Try to implement some of them or write your own awesome image filter in SAC.

4.3 Mandelbrot

Implement Mandelbrot in SAC. The tutorial touches upon this towards the end.