Applied Parallel Computing LLC
http://parallel-computing.pro

OpenACC in R language

Dmitry Mikushin Aleksei Ivakhnenko

September 6, 2016

The background

- R is nowadays the state-of-art “Matlab” for statistical computing
- R is high-level, e.g. implements collective operations on matrices and vectors
  $\Rightarrow$ OpenACC for R is unlikely possible in the form it exists for C/C++/Fortran

The background

- R is nowadays the state-of-art “Matlab” for statistical computing
- R is high-level, e.g. implements collective operations on matrices and vectors
  $\Rightarrow$ OpenACC for R is unlikely possible in the form it exists for C/C++/Fortran

So what could be done?

The background

- R is nowadays the state-of-art “Matlab” for statistical computing
- R is high-level, e.g. implements collective operations on matrices and vectors
  $\Rightarrow$ OpenACC for R is unlikely possible in the form it exists for C/C++/Fortran

So what could be done?

- R supports packages development in C++ with *Rcpp*
- Why not to develop a package in C++ exposing GPU through OpenACC?

Rcpp approach at glance

1. Install the *Rcpp* package:

```
$ sudo apt-get install r-cran-rcpp
```

2. Create package skeleton in R:

```
> library(Rcpp)
> Rcpp::Rcpp.package.skeleton("MarkovChain")
```

3. Add a C++ class implementation to *src/* folder:

```
$ ls src/
Makefile
Makefile.inc
MarkovChain.cpp
MarkovChainCPU.cpp
MarkovChainImpl.h
MarkovChainOpenACC.cpp
Rstreambuf.h
```

4. Build package library using R and custom Makefile:

```
$ make
R CMD INSTALL .
* installing *source* package 'MarkovChain ...
...
* DONE (MarkovChain)
```

5. Use the created package in R:

```
$ R

> library(MarkovChain)
> m = new("MarkovChain", c("a", "a", "b"), TRUE)
Using OpenACC implementation of MarkovChain
> m$GetTransitionMatrix()
     [,1] [,2]
[1,]    1    0
[2,]    0    0
```

Example port of package code into OpenACC kernel

```
NumericMatrix p(Rank, Rank);
...
NumericVector s(Rank, 0.);
...
for (int i = 0; i < Rank; i++)
{
    if (s[i] == 0)
        continue;
    for (int j = 0; j < Rank; j++)
        p(i,j) = p(i,j) / s[i];
}

this->TransitionMatrix = p;
```

```
...
double* parr = REAL(p);
double* sarr = REAL(s);
...
openacc_kernel(sarr, parr, Rank);
...
static void openacc_kernel(double* sarr, double* parr, int Rank)
{
    #pragma acc kernels loop independent copyin(sarr[0:Rank]) copy(parr[0:Rank*Rank])
    for (int i = 0; i < Rank; i++)
    {
        #pragma acc loop independent
        for (int j = 0; j < Rank; j++)
        {
            if (sarr[i] != 0)
                parr[i + j * Rank] /= sarr[i];
        }
    }
}
```

Problems & solutions

- PGI OpenACC is not perfect for C++: suffers from crashes when adding directives into complex C++ class methods

  Solution: Extract OpenACC-ported loops into separate static functions:

```
...
openacc_kernel(sarr, parr, Rank);
...
static void openacc_kernel(double* sarr, double* parr, int Rank)
{
    ...
}
```

- C++ data containers need a lot of work to get ported to GPUs

  Solution: Much easier to switch to plain pointers where possible:

```
...
double* parr = REAL(p);
double* sarr = REAL(s);
...
```

Check whether OpenACC is actually running

Add *PGI_ACC_TIME=1* to have the profiling info after exiting from R console:

```
$ PGI_ACC_TIME=1 R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"

> library(MarkovChain)
> m = new("MarkovChain", c("a", "a", "b"), TRUE)
Using OpenACC implementation of MarkovChain
> q()
Save workspace image? [y/n/c]: n

Accelerator Kernel Timing data
/usr/lib/R/site-library/Rcpp/include/Rcpp/routines.h
  _ZN44_INTERNAL_22_MarkovChainOpenACC_cpp_c334669514openacc_kernelEPdS0_i  NVIDIA  devicenum=0
    time(us): 37
    9: compute region reached 1 time
        14: kernel launched 1 time
            grid: [1x2]  block: [128]
            elapsed time(us): total=41 max=41 min=41 avg=41
    9: data region reached 2 times
        9: data copyin transfers: 2
             device time(us): total=26 max=16 min=10 avg=13
        19: data copyout transfers: 1
             device time(us): total=11 max=11 min=11 avg=11
```