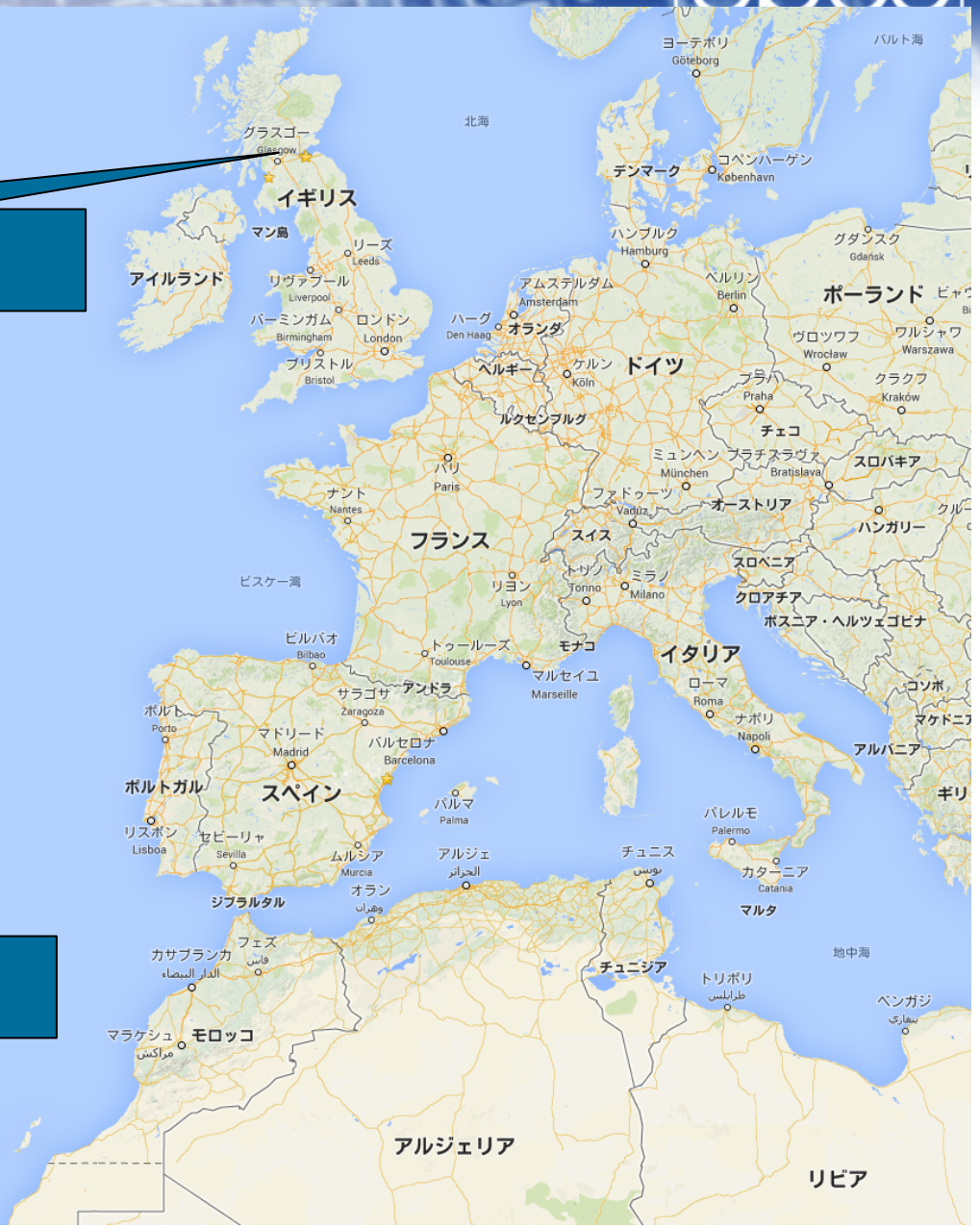
Overview of OpenACC and the SHMEM implementation of GROMACs

Ruyman Reyes, Ph.D
EPCC
rreyesc@epcc.ed.ac.uk

スペイン

Edinburgh (エディンバラ)

Pop. Density:
- Japan: 337.1/km²
- Tenerife: 442/km²



Islas Canarias (カナリア諸島)

Tenerife (テネリフェ島)

Overview of OpenACC

Ruyman Reyes, Ph.D
EPCC
rreyesc@epcc.ed.ac.uk

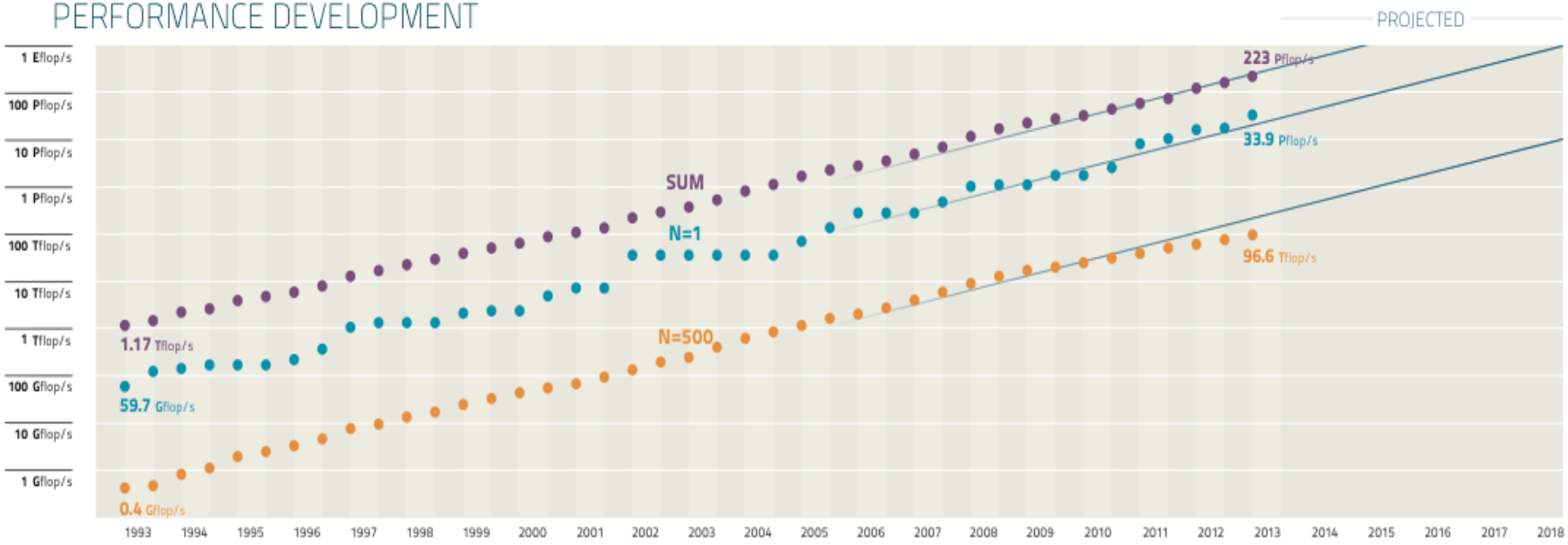
TOP500® JUNE 2013

PRESENTED BY UNIVERSITY OF MANNHEIM · ICL INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE · BERKELEY LAB Lawrence Berkeley National Laboratory

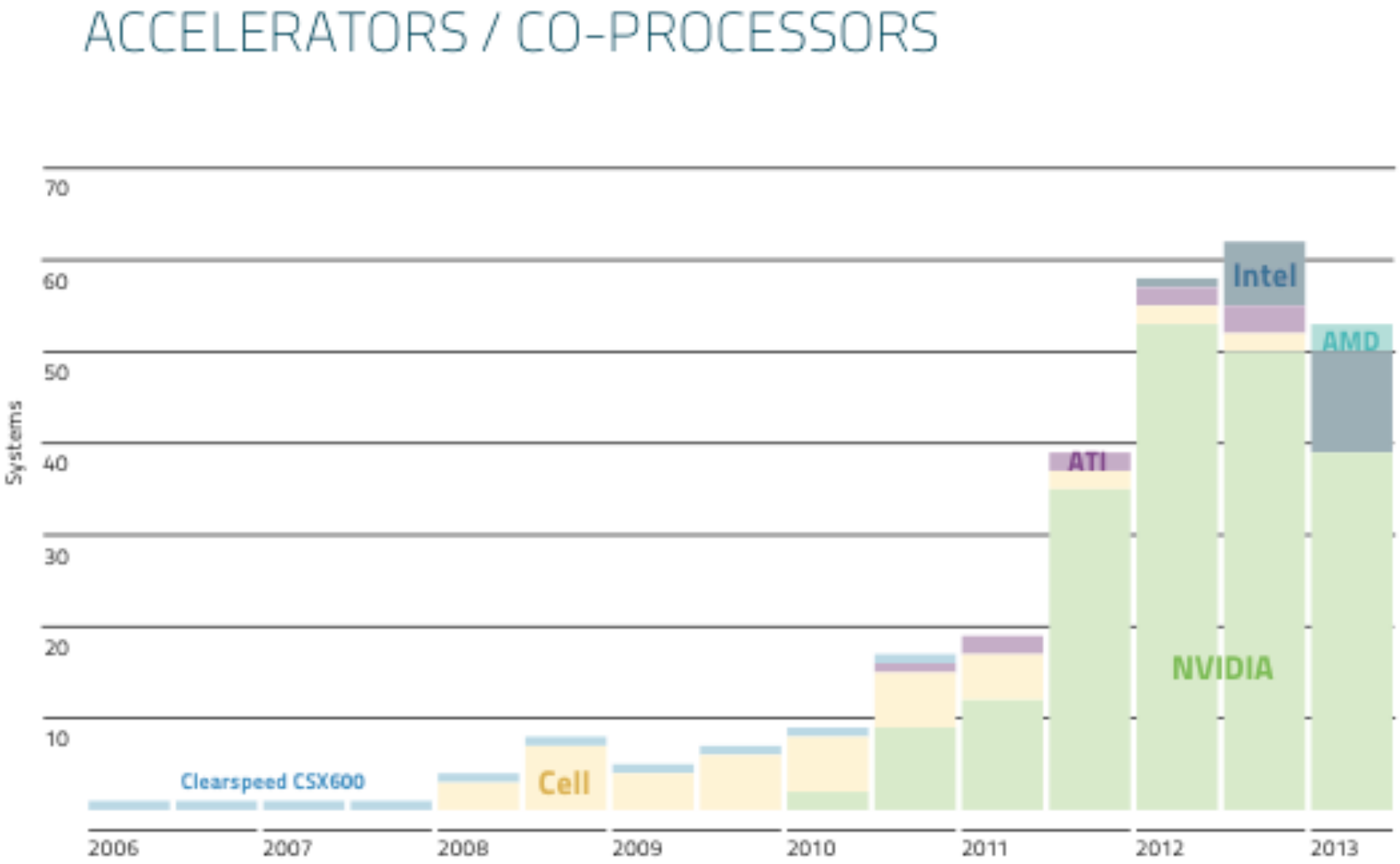
FIND OUT MORE AT www.top500.org

	NAME	SPECS	SITE	COUNTRY	CORES	R_{MAX} PFLOP/S	POWER MW
1	**Tianhe-2 (Milkyway-2)**	NUDT, Intel Ivy Bridge (12C, 2.2 GHz) & Xeon Phi (57C, 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	**33.9**	17.8
2	**Titan**	Cray XK7, Opteron 6274 (16C, 2.2 GHz) + Nvidia Kepler (14C, .732 GHz), Custom interconnect	DOE/SC/ORNL	USA	560,640	**17.6**	8.3
3	**Sequoia**	IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	**17.2**	7.9
4	**K computer**	Fujitsu SPARC64 VIIIfx (8C, 2.0GHz), Custom interconnect	RIKEN AICS	Japan	705,024	**10.5**	12.7
5	**Mira**	IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA	786,432	**8.16**	3.95

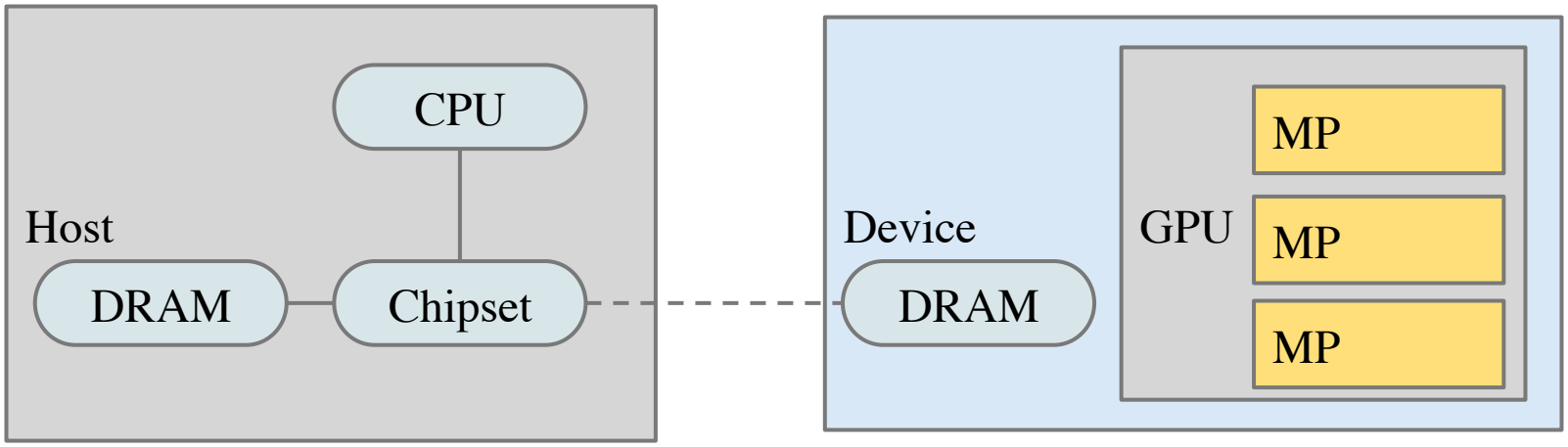
PERFORMANCE DEVELOPMENT



Co-Processors



GPU hardware



- Separate memory address space bw Host/Device
  - Newer devices have means to hide this fact
- Host handles the device
  - Populates Device memory
  - Create the program to execute
  - Collect the results

GPU programming

- Two main programming frameworks
  - CUDA: Nvidia programming model, quite common
  - OpenCL: Standard based on CUDA programming model but no longer focused on GPUs
- Both of them based on a two-source system
  - Host code with API calls to prepare execution and data
  - Kernel code exposing device-specific features and parallelism
- Specific compiler and/or libraries are required
  - CUDA: NVIDIA CC
  - OpenCL: Vendor-specific platform

Simple example

```
void saxpy_serial(int n, float a, float *x,
float *y)
{
for (int i = 0; i < n; ++i)
      y[i] = a*x[i] + y[i];
}
```

Original C code

Example of CUDA code (I)

```
__global__ void saxpy_parallel(int n, float
a, float *x, float *y)
{
int i = blockIdx.x*blockDim.x + threadIdx.x;
if (i < n) y[i] = a*x[i] + y[i];
}
```

Kernel code

Example of CUDA code (II)

```
// Allocate two N-vectors h_x and h_y
int size = N * sizeof(float);
float* h_x = (float*)malloc(size);
float* h_y = (float*)malloc(size);
// Initialize them...
// Allocate device memory
float* d_x; float* d_y;
cudaMalloc((void**)&d_x, size));
cudaMalloc((void**)&d_y, size));
// Copy host memory to device memory
cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice);
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(N, 2.0, d_x, d_y);
// Copy result back from device memory to host memory
cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost);
```

Host code

Simple OpenCL example

```
__kernel void saxpy(const unsigned int n,   \
    const float a,                          \
   __global float* x,                       \
    __global float* y)
{
    int i = get_global_id(0);
    if(i < n)
        y[i] = a * x[i] + y[i];
}
```

Kernel code

http://users.jyu.fi/~tro/TIEA342/OpenCL/saxpy.c

Simple OpenCL example

```
cl_platform_id platform;
 cl_uint num_platforms;
 err = clGetPlatformIDs(1,&platform,&num_platforms);

 if (err != CL_SUCCESS) {
   printf("Error: Failed to get a platform id!\n");
   return EXIT_FAILURE;
 }

  size_t returned_size = 0;
 cl_char platform_name[1024] = {0}, platform_prof[1024] = {0}, platform_vers[1024] = {0}, platform_exts[1024] = {0};
 err  = clGetPlatformInfo(platform, CL_PLATFORM_NAME,       sizeof(platform_name), platform_name, &returned_size);
 err |= clGetPlatformInfo(platform, CL_PLATFORM_VERSION,    sizeof(platform_vers), platform_vers, &returned_size);
 err |= clGetPlatformInfo(platform, CL_PLATFORM_PROFILE,    sizeof(platform_prof), platform_prof, &returned_size);
 err |= clGetPlatformInfo(platform, CL_PLATFORM_EXTENSIONS, sizeof(platform_exts), platform_exts, &returned_size);

 if (err != CL_SUCCESS) {
   printf("Error: Failed to get platform infor!\n");
   return EXIT_FAILURE;
 }
 }
 context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &err);
 if (!context) {
   return EXIT_FAILURE;
 }
 cl_command_queue commands;
 commands = clCreateCommandQueue(context, device_id, 0, &err);

 if (!commands) {
   return EXIT_FAILURE;
 }
 program = clCreateProgramWithSource(context, 1, (const char **) &KernelSource, NULL, &err);
 if (!program) {
   return EXIT_FAILURE;
 }
 err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

This is not the complete host code!

.....

http://users.jyu.fi/~tro/TIEA342/OpenCL/saxpy.c

Both P.M. are extremely verbose and cluttered!

Why don’t use something more minimalistic?

OpenMP: Exploiting multi-core nodes

```
void saxpy_serial(int n, float a, float *x,
float *y)
{
#pragma omp parallel for
for (int i = 0; i < n; ++i)
      y[i] = a*x[i] + y[i];
}
```

Compiler translates sequential code to a parallel implementation

Some previous approaches

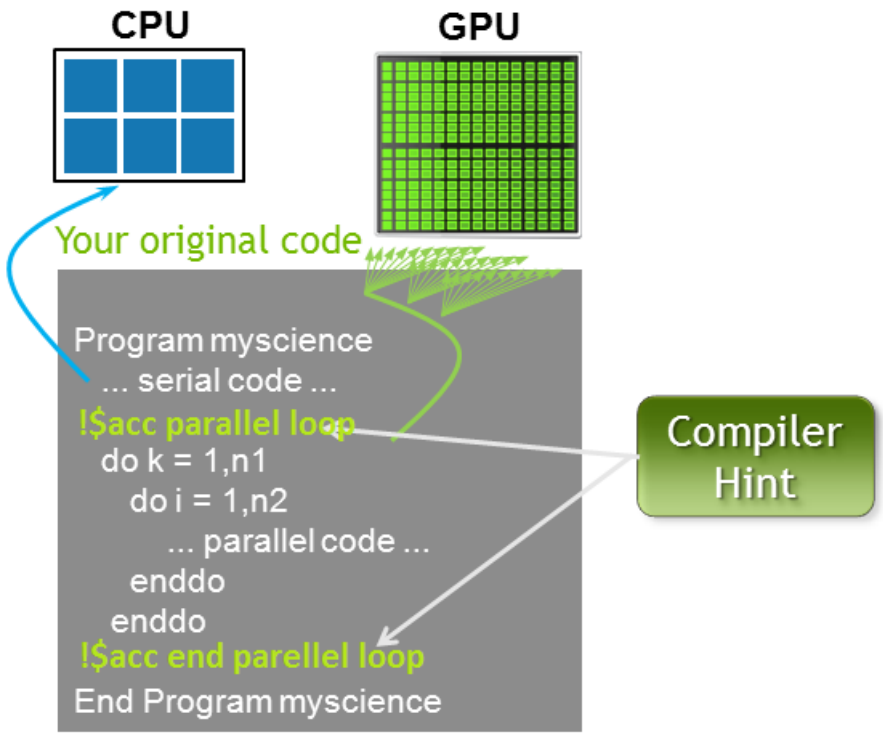
- hiCUDA (Toronto Univ., 2009)
- PGI Accelerator Model (Initial. rel. 2008)
- HMPP Directives (from CAPS)
- llc (Univ. la Laguna, 2010) / llcl (Univ. La Laguna, 2011)
- OMPCUDA (Univ. of Tokyo, 2010) based on OMNI OpenMP compiler (Univ. of Tsukuba)

OpenACC standard

- Combined effort from PGI, CAPS, Cray and others to produce an standard for GPU directives
- Members of the standard are also on the OpenMP comittee

OpenACC execution model

- Host-directed execution with attached GPU
  - Compute regions offloaded to the accelerator
  - Device execute parallel regions
- Host handle all the set-up operations
  - Allocate, initialise, copy, queue, wait, etc
- Hosts can queue operations



OpenACC standard

- C and Fortran API
  - #pragma acc in C
  - !$acc sentinels in Fortran
- Currently implemented by PGI, CAPS and Cray
- Open Source / experimental implementation (accULL)
  - Compilant with 1.0 rev. of the standard for C
  - **Support for CUDA and OpenCL devices**
  - https://bitbucket.org/ruyman/accull/downloads
    - Official release 0.2 (April, quite old)
    - Pre-release of 0.3 available!
  - Python compiler framework (yacf) and C++ Runtime System
  - *accULL: An OpenACC Implementation with CUDA and OpenCL Support* – EuroPar '12

Two key ideas

Offload Region

- Selects a piece of code to be offloaded to the device
- The host set-up the required parameter(s) and run the kernel(s)
- May wait or not for the kernel(s) to finish

Data Region

- Defines data required on the device for a future offload region
- User indicates the compiler/runtime which variables will be required in the future
- Can indicate variable directionality too

Very simple example

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

OpenMP

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
   #pragma omp parallel for shared(a, c)
    for (int i = 0; i < 20; i++)
       for (int j = 0; j < 30; i++) {
            // Do something with a and c!
       }
```

OpenACC Compute region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    #pragma acc kernels loop
     for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
             // Do something with a and c !
        }
}
```

Basic offload directive

OpenACC Compute region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    #pragma acc kernels loop
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

Basic offload directive

OpenACC Compute region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    #pragma acc kernels loop
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

Implicit data region

OpenACC Compute region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    #pragma acc kernels loop
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

Copy Inside

Copy Outside

OpenACC Compute region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
/* Repeat until accuracy is enough */
while (err >= TOL) {
    #pragma acc kernels loop
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

Copy Inside

Copy Outside

OpenACC Data region

```
/* Matrix initialization */
int a[20][30];
int c[10][15];
/* Support code */
....
#pragma acc data copy(a,c)
while (err >= TOL) {
    #pragma acc kernels loop
    for (int i = 0; i < 20; i++)
        for (int j = 0; j < 30; i++) {
            // Do something with a and c !
        }
}
```

Copy Inside

Copy Outside

Kernels

- An accelerator kernels construct surrounds loops to be executed on the accelerator, typically as a sequence of kernel operations.
- C

```
#pragma acc kernels [clause [[,] clause]…] new-line
{ structured block }
```

- Any data clause is allowed.
  - copy,copyin,copyout,...
- other Clauses
  - if(condition)
  - async(expression)

Loop

- A loop construct applies to the immediately following loop or nested loops, and describes the type of accelerator parallelism to use to execute the iterations of the loop
- C

```
#pragma acc loop [clause [[,] clause]…] new-line
{ loop nest }
```

- Clauses:
  - collapse(n)
  - seq
  - private(list) , firstprivate(list)
  - reduction(operator:list) (+,-,*,max,min...)

Kernels and Loop example

```
for (i = 0; i < N; i++) {

  for (j = 0; j < N; j++) {

    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

General rule: The more information you provide, the better

Kernels and Loop example

```
#pragma acc kernels loop
for (i = 0; i < N; i++) {

  for (j = 0; j < N; j++) {

    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

- Offload the code to the accelerator
- Mapping loop to HW architecture is up to the compiler

Kernels and Loop example

```
#pragma acc kernels loop
for (i = 0; i < N; i++) {
  #pragma acc loop
  for (j = 0; j < N; j++) {

    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

- Offload the code to the accelerator
- Both loops can be scheduled on the accelerator

Kernels and Loop example

```
#pragma acc kernels loop collapse(2)
for (i = 0; i < N; i++) {

  for (j = 0; j < N; j++) {

    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

- Offload the code to the accelerator
- Both loops can be scheduled on the accelerator

Kernels and Loop example

```
#pragma acc kernels loop independent
for (i = 0; i < N; i++) {
  #pragma acc loop independent
  for (j = 0; j < N; j++) {

    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

- Offload the code to the accelerator
- Both loops can be scheduled on the accelerator
- Forces compiler to detect iterations as independent

Kernels and Loop example

```
#pragma acc kernels loop independent
for (i = 0; i < N; i++) {
  #pragma acc loop independent
  for (j = 0; j < N; j++) {
    #pragma acc loop seq
    for (k = 0; k < N; k++)
     a[i][j] += b[i][k]*c[k][j]
  }
```

- The inner loop is marked as sequential

Compiler output

- Compilers typically shows information about the code translation in standard output
  - PGI: Compile with –Minfo , you'll see which loops are converted into kernels. Some loops may be converted into kernels, that does not mean they will run in parallel!!
  - Caps –v : Shows which loops from a nest will be executed in parallel
  - Cray: Shows general information when compiling
  - accULL : Lots of info about translation, stored in a log file by default, output to stdout if -v
- Use this information carefully to further optimize your code!

Data

- An accelerator data construct defines a region of the program within which data is accessible by the accelerator.
- C

```
#pragma acc data [clause [[,] clause]…] new-line
{ structured block }
```

- Any data clause is allowed.
  - copy, copyin, copyout,...
- other clauses
  - if(condition)
  - async(expression)

Data clauses

- copy
- copyin
- copyout
- present
- pcopy
- pcopyin
- pcopyout
- deviceptr

Data

```
int main(…) {

…

#pragma acc kernels loop copy(a[0:N])
  for (int i = 0; i < N; i++)
            a[i] = 1;
#pragma acc kernels loop pcopyin(a[0:N]) \
                         copyout(b[0:N])
  for (int i = 0; I < N; i++)
             b[i] = a[i]

…
```

Re-using existing CUDA code

- Using native CUDA calls (or whatever)

```
#pragma acc host_data use_device(a,b,c)
  cublasSgemm('n','n',m,n,k,1,a,lda,b,ldb,0,b,ldc);
```

- Using a device pointer in a loop

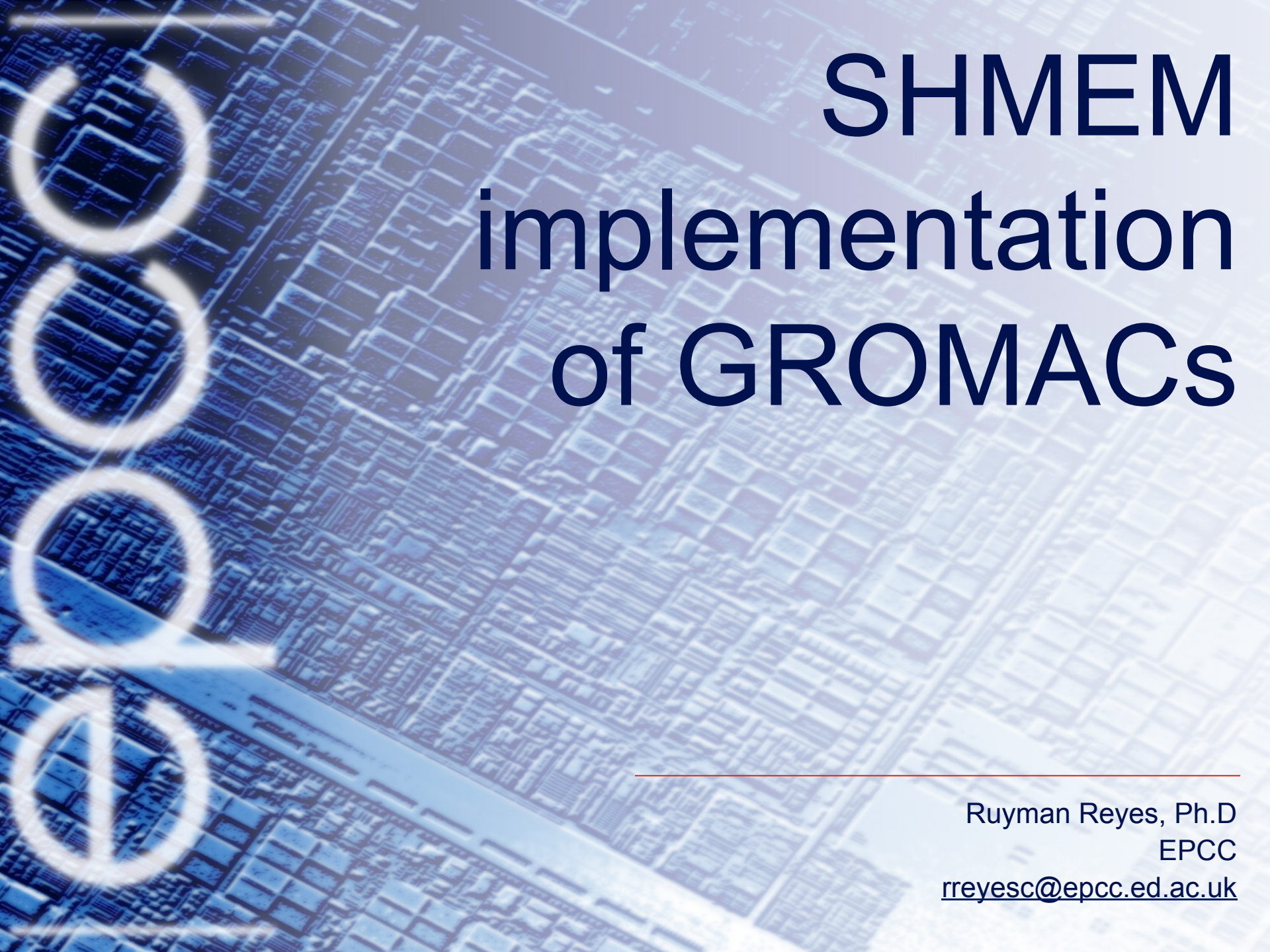
```
void * ptr = acc_malloc(size);
…
#pragma acc parallel loop deviceptr(ptr)
{
  ….
```

Other directives/features

- Ver. 1.0
  - Parallel directive (low-level kernel creation)
  - gang/worker/vector clauses (kernel tuning)
  - cache (memory access optimization within the device kernel)
- Ver 2.0 (Announced during ISC 2013)
  - atomic directive
  - runtime calls to handle memory (memcpy-to-device)
  - tile clause (apply loop tiling)
  - device_type (device-specific optimization)
- Support for Intel MIC architectures expected 2014?
  - Can be implemented with accULL !

Learning OpenACC

- One-day tutorial with excercises
  - https://bitbucket.org/ruyman/openacc-lab/overview
- OpenACC training resources
  - http://openacc.org/
    - Education section contains wide variety of resources
    - Talks from PGI and others are available there
- Stack Overflow – OpenACC tag
- EPCC Benchmarks
  - https://github.com/nickaj/epcc-openacc-benchmarks

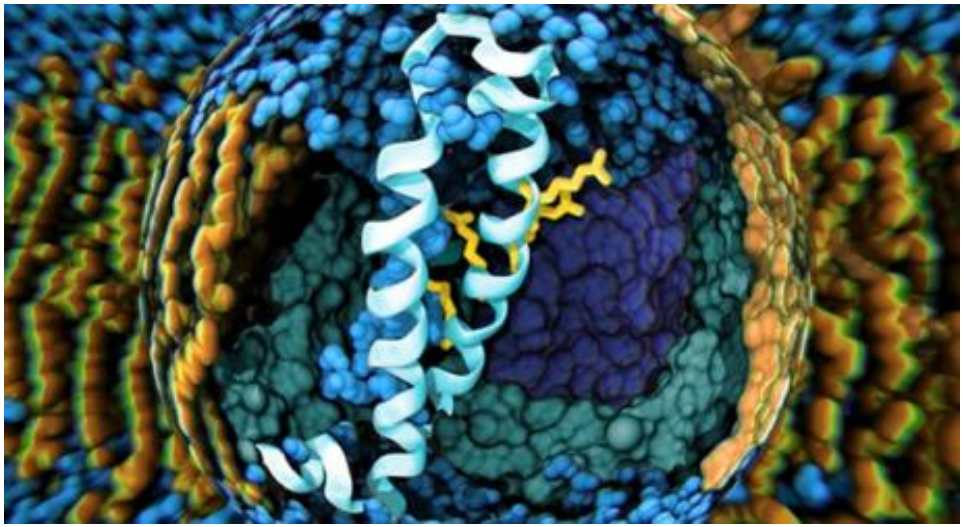


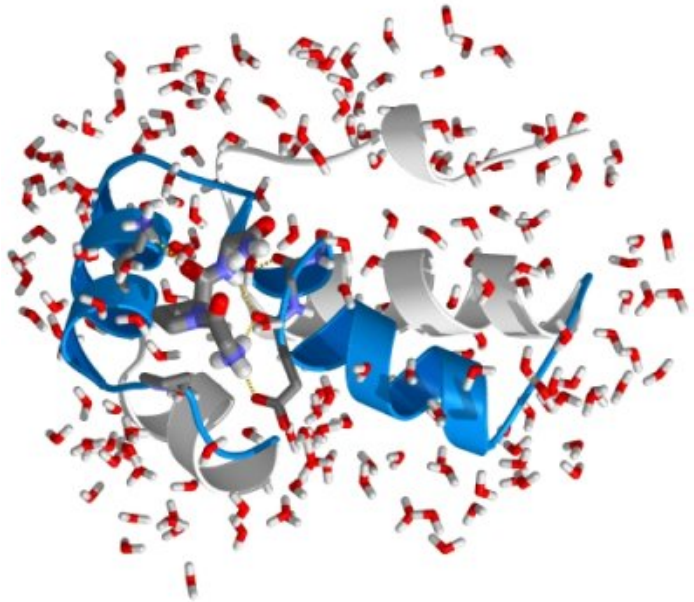
SHMEM implementation of GROMACs

Ruyman Reyes, Ph.D
EPCC
rreyesc@epcc.ed.ac.uk

GROMACs

- One of the leading biochemical MD simulation packages
- Widely used for simulation of biochemical systems
- Both PRACE pan-European HPC and CRESTA exascale projects have identified GROMACs as a key code for the future



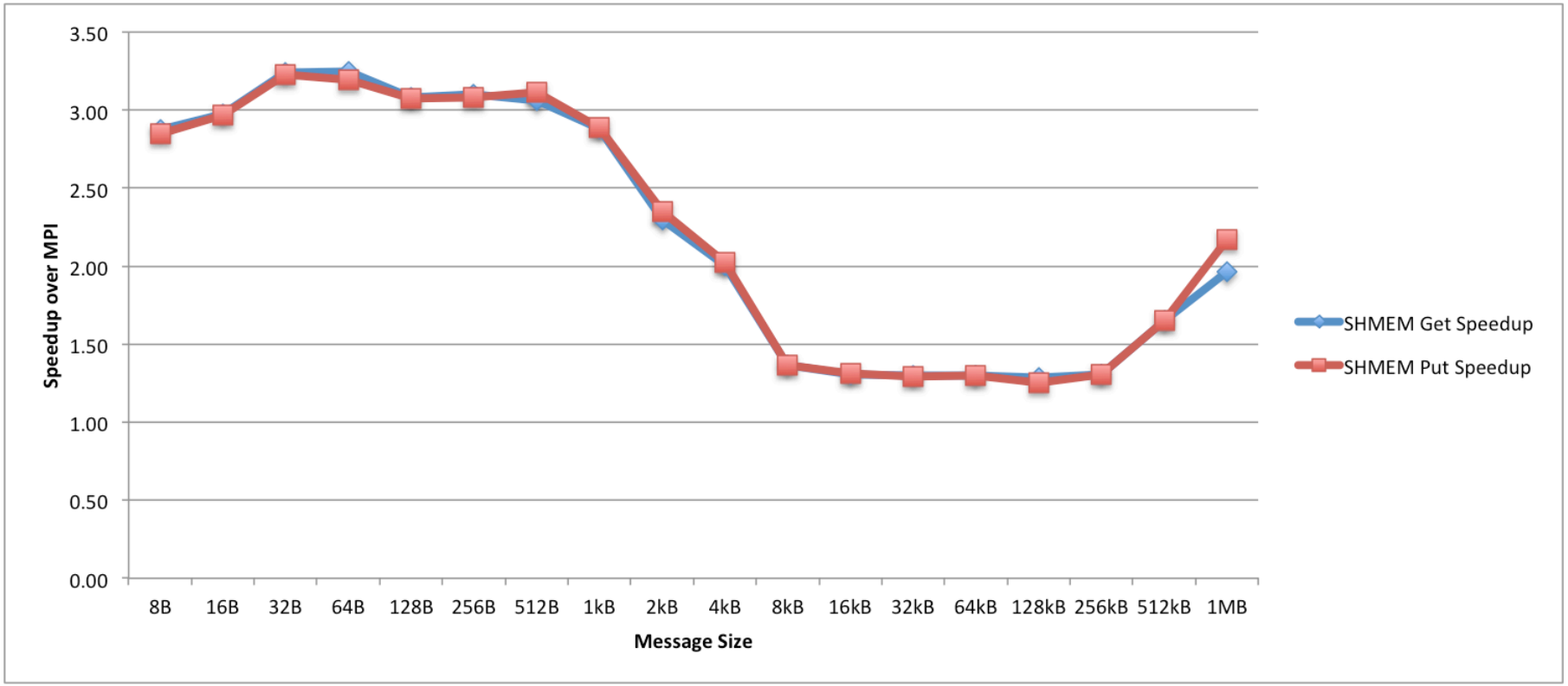


GROMACs - SHMEM

- 6-month project to implement comm. Layer of GROMACs using SHMEM
- Should improve performance in HECToR (UK main HPC resource)
  - Cray SHMEM implementation has very little overhead and good performance
  - Code is free of Cray SHMEM specifics where possible, uses ifdef when not possible
- It is currently a WIP – finishing end of August.

Why SHMEM?

- GROMACs uses SendRecv operations to communicate
- SHMEM Implementation provide up to 3.25 speedup



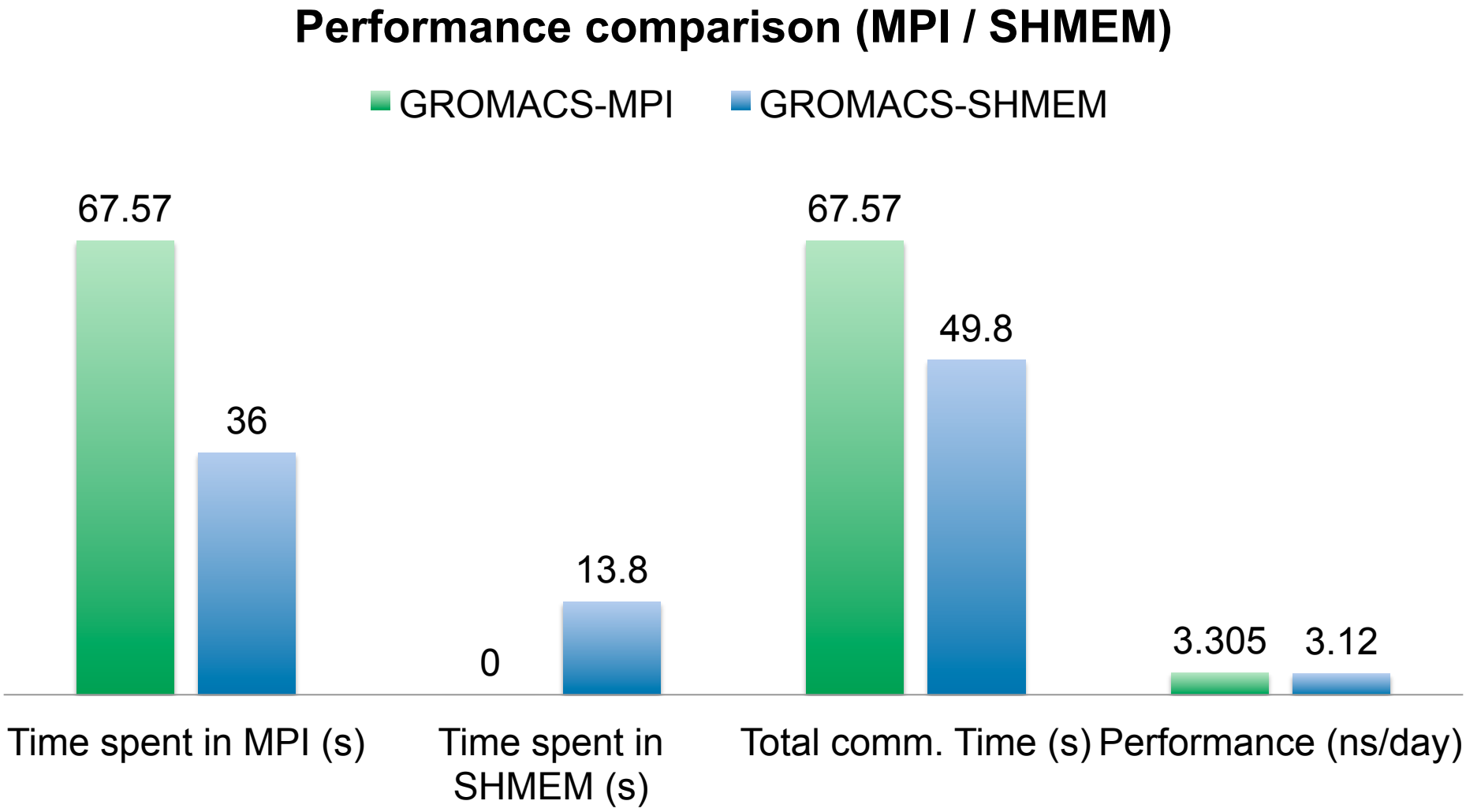
Progress so-far

- Startup and customization of building tools
  - Use GMX_SHMEM=ON when configuring to enable it
- Implementation of the Domain Decomposition
  - MPI_SendRecv in the DD have been replaced by PUT operations
  - Collectives replaced by SHMEM equivalents
- Implementation of the Particle Decomposition
  - MPI_SendRecv have been replaced by PUT operations
  - No collectives converted -> they are not critical in this part of the code
- PP -> PME communication
  - Still work in progress …

Problems so-far

- Lack of documentation / examples
  - Cray documentation is focused on routines
  - OpenSHMEM doc is limited (although useful)
- Restrictions on variables that can be communicated:
  - Only symmetric memory can be used in the SHMEM routines
  - This memory needs to be allocated with a specific routine (shmalloc, shrenew)
  - This routine contains an implicit barrier!
  - Many temporary buffers used in GROMACS – and they are not created by all the process at the same time!

Preliminary results



ADH test-case, 8 cores, 10000 iterations, 1 HECToR-node

Summary

- Still some work to do
- Most of the performance lost in the SHMEM implementation is due to excessive synchronization (due to temp. buffers)
- Promising reduction on the comm. Time
- Hard to get it to work!
  - Would be nice an OpenACC-like language to implement this ☺

Thanks / ありがとう

Ruyman Reyes, Ph.D
EPCC
rreyesc@epcc.ed.ac.uk