www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

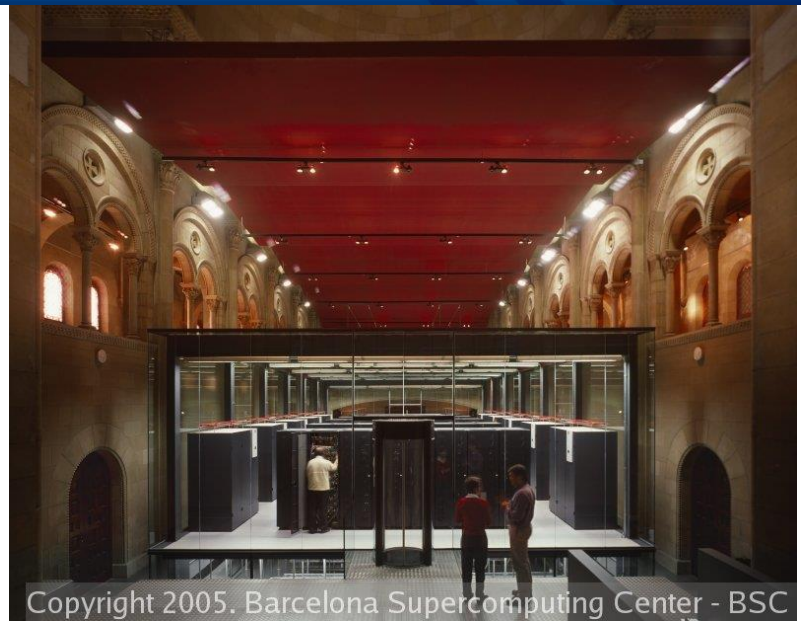
OmpSs + OpenACC

Multi-target Task-Based Programming Model Exploiting OpenACC GPU Kernel

Guray Ozen
guray.ozen@bsc.es

Exascale in BSC

- **Marenostrum 4** (13.7 Petaflops)
  - General purpose cluster (3400 nodes) with Intel Xeon
  - Emerging technologies clusters
    1. IBM Power9 – Nvidia GPU
    2. Intel Knights Landing (KNL) and Intel Knights Hill (KNH)
    3. 64 bit ARMv8 processors that Fujitsu



- **Research Lines**
  - **OmpSs** Parallel programming model
    - Simple data directionality annotations for tasks
    - Asynchronous data-flow, intelligence to the runtime
  - **BSCTools -** Performance analysis tools
    - Extrae, paraver and Dimemas
    - Performance analytics: intelligence, insight
  - **CUDA Center of Excellence**
    - PUMPS summer school 2010-2017, courses at BSC and UPC
  - **Mont-Blanc**
    - Exploring the potential of low-power GPU clusters as high-performance platforms

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Home of OmpSs Programming Model

OmpSs Programming Model

- Parallel Programming Model
  - Directive based to keep a serial version
  - Targeting: SMP, clusters and accelerator devices

- Experimental Platform
  - Mercurium Compiler (source-to-source) for Fortran/C/C++
  - Nanos Runtime
  - Applications

- **Forerunner** for OpenMP
  - “extending” OpenMP
  - “following” OpenMP

OmpSs Programming Model

Key concept

- **Single Program → Any target**
- **Sequential task based** program on **single address/name space + directionality annotations**
- Happens to execute parallel: Automatic run time computation of dependencies between tasks

Differentiation of OmpSs

- Dependences: Tasks instantiated but not ready.
  - Look ahead
    - Avoid stalling the main control flow when a computation depending on previous tasks is reached
    - Possibility to "see" the future searching for further potential concurrency
  - Dependences built from data access specification
- Locality aware
  - Without defining new concepts
- Homogenizing heterogeneity
  - Device specific tasks but homogeneous program logic

Task based concepts of OmpSs

- Minimalist set of concepts …

Key: OpenMP, influenced OpenMP, pushing, not yet

```
#pragma omp task [ in (array_spec, l_values...)]  [ out (...)]  [ inout (…, v[neigh[j]], j=0;n)]) \
                 [ concurrent  (…)] [commutative(...)]  [ priority(P) ] [ label(...) ] \
                 [ shared(...)][private(...)][firstprivate(...)][default(...)][untied] \
                 [final(expr)][if  (expression)] \
                 [reduction(identifier : list)] \
                 [resources(…)]
{code block or function}
```

```
#pragma omp taskwait [ { in | out | inout } (...) ] [noflush]
```

```
#pragma omp taskloop [grainsize(…) ] [num_tasks(…) [nogroup] [ in (...)] [reduction(identifier : list)]
{for_loop}
```

OpenMP compatibility

- Follow OpenMP syntax
  - For adopted OmpSs features
  - Adapt semantics for OpenMP features. Ensure High compatibility

```
#pragma omp parallel // ignore
```

```
#pragma omp for [ shared(...)][private(...)][firstprivate(...)][schedule_clause]  // ≈ taskloop
   {for_loop}
```

```
#pragma omp task [depend (type: list)]
```

OpenMP 4.5 GPU Offload Support

- **MACC Compiler**: Experimental branch supports **OpenMP 4.5 GPU Offload**
  - Relying on OmpSs task model and migrating OpenMP 4.5 directives with OmpSs
  - Generates CUDA/OpenCL codes
- **Key concepts:**
  - Propose clauses that improve kernel performance
  - Change in mentality … minor details make a difference

#pragma omp target device (acc) #pragma omp task #pragma omp teams distribute parallel for {for_loop}	#pragma omp target device (acc) #pragma omp task #pragma omp parallel for {for_loop}

```
#pragma omp taskwait [ on (...) ][noflush]
```

OmpSs GPU Support

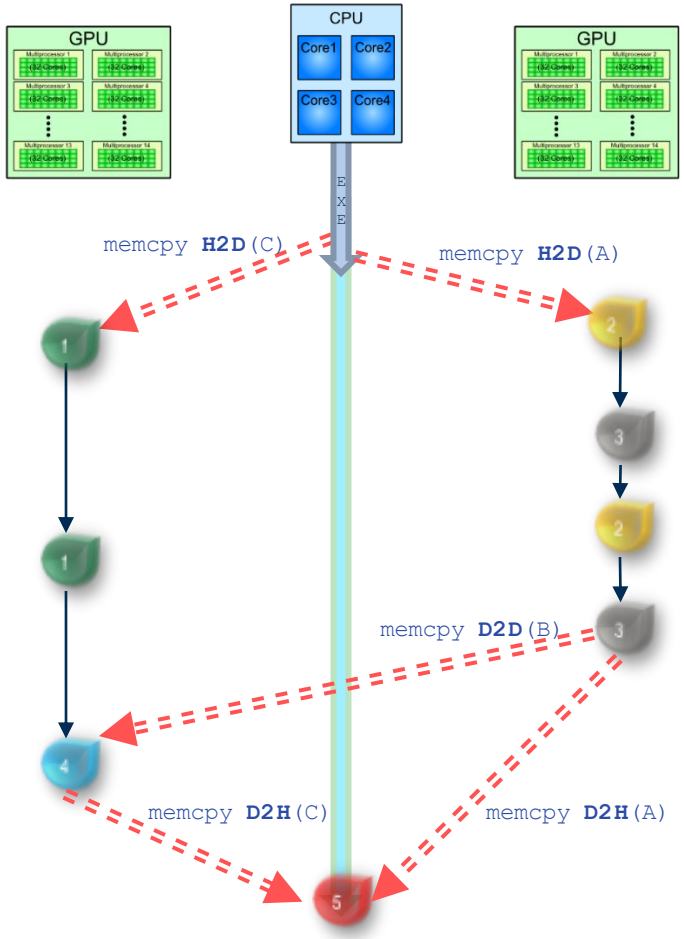
- Single address space program … executes in several non-coherent address spaces
  - Copy clauses:
    - ensure sequentially consistent copy accessible in the address space where task is going to be executed
  - Requires precise specification of data accessed (e.g. array sections)
  - Runtime offloads data and computation and manages consistency
- Kernel based programming
  - Separation of iteration space identification and loop body

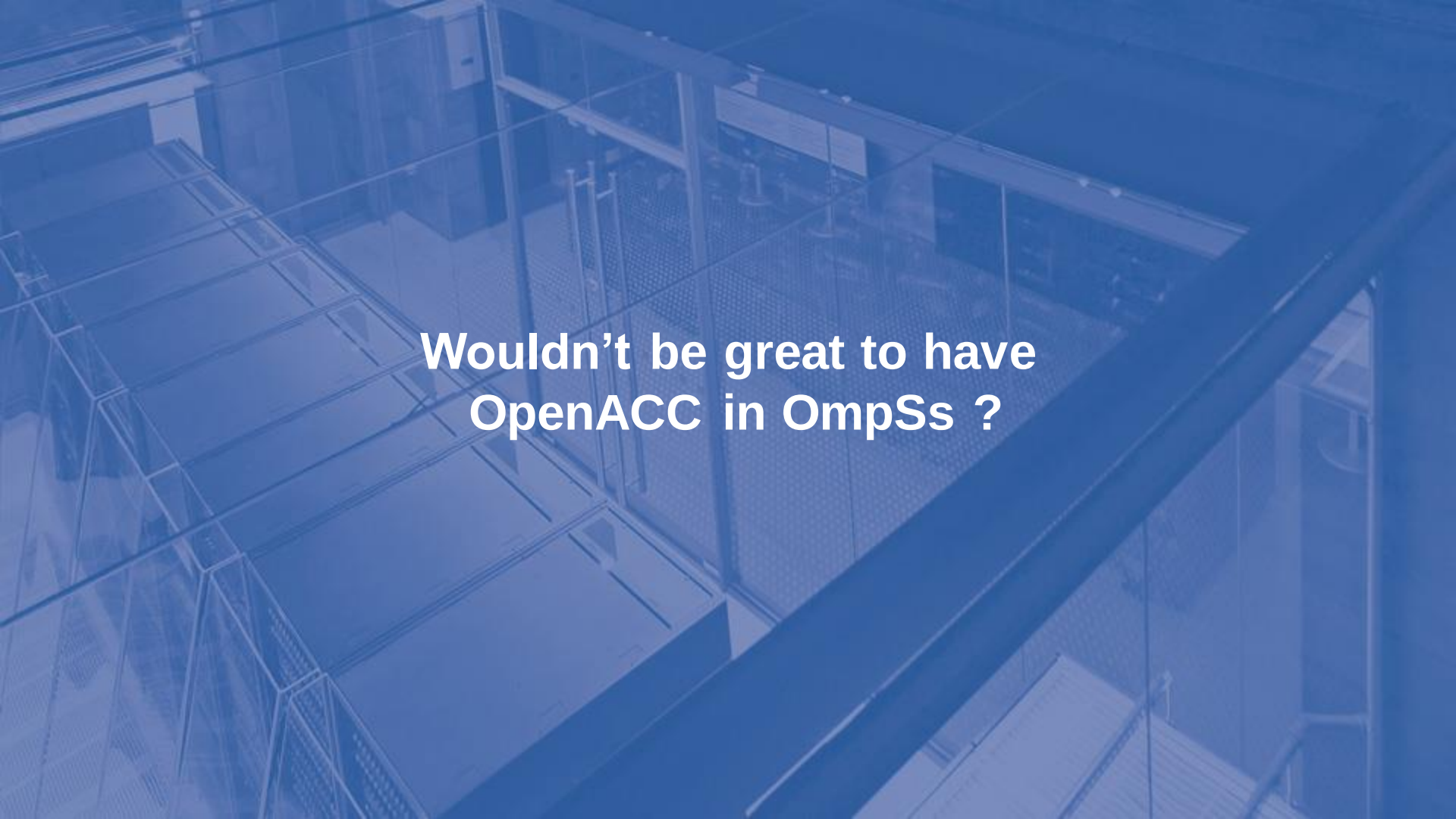
```
#pragma omp target device ({ smp | opencl | cuda }) \
                [ copy_deps | no_copy_deps ] [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] } \
                [ implements ( function_name )] \
                [shmem(...) ] \
                [ndrange (dim, g_array, l_array)]
```

```
#pragma omp taskwait [ on (...) ][noflush]
```

GPU Execution Model of OmpSs

```
#pragma omp target device(cuda) ndrange(1, N, 128)
#pragma omp task in(C) out(D)
__global__ MyFastKernel(double *C, double *D, int N) {
    <.. CUDA Kernel Codes ..>
}
int main(…) {
    double A[N], B[N], C[N] , D[N];
    for (int j=0; j<2; ++j) {
1      MyFastKernel( C, D, N) ;

2      #pragma omp target device(acc)
       #pragma omp task in(A) out(B)
       #pragma omp teams distribute parallel for
        for(i=0 ; i< N; ++i)
         <..Sequential Codes to generate CUDA..>

3      #pragma omp target device(acc)
       #pragma omp task inout(A,B)
       #pragma omp teams distribute parallel for
        for(i=0 ; i< N; ++i)
         <..Sequential Codes to generate CUDA..>
     }
4   #pragma omp target device(acc)
    #pragma omp task inout(C,B) in(D)
    #pragma omp teams distribute parallel for
     for(i=0 ; i< N; ++i)
      <..Sequential Codes to generate CUDA..>

5   #pragma omp target device(smp)
    #pragma omp task in(A, C)
      <..CPU codes / Print results to file ..>
    #pragma omp taskwait
}
```





Wouldn’t be great to have OpenACC in OmpSs ?

Idea & Motivation

- **Motivation**
  - **OpenACC** compilers deliver best performance by generating highly optimized GPU codes
  - **OmpSs** has powerful task support that allows to maintain entire application
    - Single address space -> any or multiple target
    - Potential ability to run same tasks onto hybrid GPU + CPU

- **Goal:** Make use of OpenACC GPU support with OmpSs task model

OpenACC Integration in OmpSs

- New device type for openacc is added

New Key: OpenACC

Start with **OmpSs,** Manage task dependency, data and multiple device
Parallelize with **OpenACC**…

#pragma omp target device (openacc) #pragma omp task [{ in | out | inout } (...)] #pragma acc kernels [*clause-list*] {code block}	#pragma omp target device (openacc) #pragma omp task [{ in | out | inout } (...)] #pragma acc parallel [*clause-list*] {code block}

Compilation Workflow

Device management is done by OmpSs passed to OpenACC

Streams are managed by OmpSs and passed to OpenACC

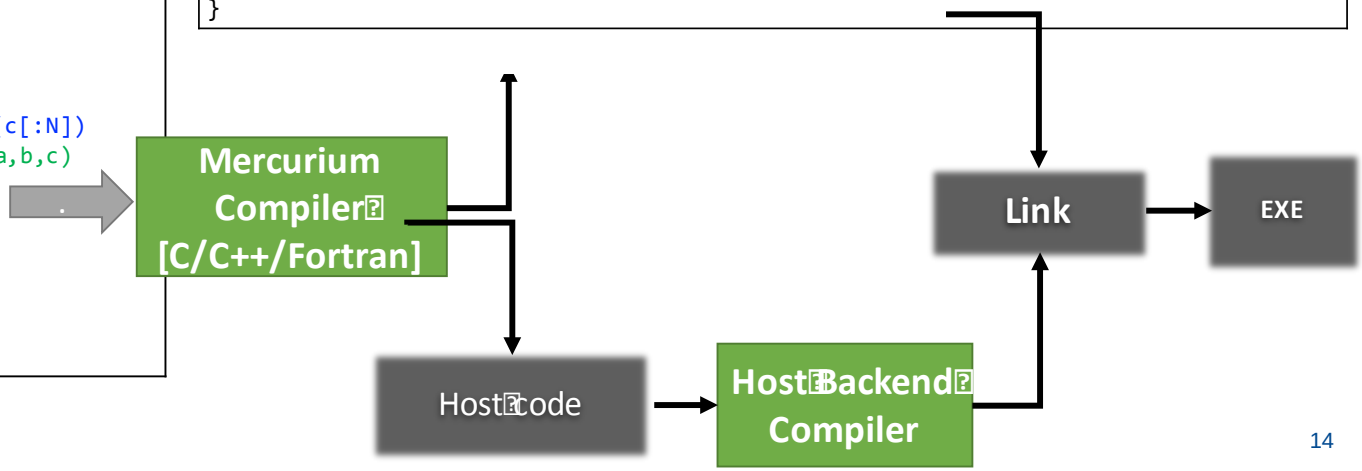
Each kernel is submitted asynchronously

Input Code

```
int main(int argc, char* argv) {
  double a[N], b,[N] c[N];

  #pragma omp target device (openacc)
  #pragma omp task in(a[:N],b[:N]) out(c[:N])
  #pragma acc parallel loop deviceptr(a,b,c)
  for (int i = 0; i < N; ++i) {
        c[i] = a[i] + b[i];
   }

  #pragma omp taskwait
  return 0;
}
```

OpenACC Code

```
#include <openacc.h>
#include <cuda_runtime.h>
extern "C" {
    extern int nanos_get_device_id_();
    extern cudaStream_t nanos_get_kernel_stream();
    extern unsigned int nanos_get_kernel_stream_id();
}
void oacc_ol_main_0_7_vecadd_unpacked (int* a, int* b, int* c, int N) {
   acc_set_device_num( nanos_get_device_id_() , acc_device_nvidia );
   acc_set_cuda_stream(nanos_get_kernel_stream_id(), nanos_get_kernel_stream());

   #pragma acc parallel loop deviceptr(a,b,c) async(nanos_get_kernel_stream_id())
   for (int i = 0; i < N; ++i) {
        c[i] = a[i] + b[i];
   }
}
```



Stream Benchmark

1st Style (single GPU)

- OmpSs creates single OpenACC task
- Single OpenACC task is run on single GPU

```
void triad(T* a, T* b, T* c, T scalar, int N){
  #pragma omp target device (openacc)
  #pragma omp task in(b[0:N], c[0:N]) out(a[0:N])
  #pragma acc parallel loop deviceptr(a,b,c)
  for (int i = 0; i < N; i++)
      a[i] = b[i]+scalar*c[i];
}
int  main(int argc, char const *argv[]) {
   ...
   copy(a, c, size);
   scale(b, c, size);
   add(a, b, c, scalar, size);
   triad(a, b, c, scalar, size);
}
```

device = Only openacc is requested
copy_deps = Copies dependencies to the target

Dependencies are specified

OmpSs manages data.
Symbols are passed deviceptr clause to inform OpenACC

Stream Benchmark

2nd Style (Multiple GPU)

- OmpSs creates multiple OpenACC tasks
- Multiple OpenACC tasks are run automatically on multiple GPU

```
void triad(T* a, T* b, T* c, T scalar, int N){
  #pragma omp target device (openacc)
  #pragma omp task in(b[0:N], c[0:N]) out(a[0:N])
  #pragma acc parallel loop deviceptr(a,b,c)
  for (int i = 0; i < N; i++)
      a[i] = b[i]+scalar*c[i];
}
int  main(int argc, char const *argv[]) {
   ...
   for (int i = 0; i < N; i += CHUNK) {
     copy(&a[i], &c[i], CHUNK);
     scale(&b[i], &c[i], CHUNK);
     add(&a[i], &b[i], &c[i], scalar, CHUNK);
     triad(&a[i], &b[i], &c[i], scalar CHUNK);
} }
```

device = openacc are requested
copy_deps = Copies dependencies to the target if it's required

Dependencies are specified

OmpSs manages data.
Symbols are passed deviceptr clause to inform OpenACC

Loop Blocking

Stream Benchmark

3rd Style (multiple GPU + multicore SMP)

- OmpSs compiler creates multiple OpenACC and SMP tasks
- Multiple tasks are run automatically on multiple GPU or CPU

```
void triad(T* a, T* b, T* c, T scalar, int N){
  #pragma omp target device (openacc, smp)
  #pragma omp task in(b[0:N], c[0:N]) out(a[0:N])
  #pragma acc parallel loop deviceptr(a,b,c)
  for (int i = 0; i < N; i++)
      a[i] = b[i]+scalar*c[i];
}
int  main(int argc, char const *argv[]) {
   ...
   for (int i = 0; i < N; i += CHUNK) {
     copy(&a[i], &c[i], CHUNK);
     scale(&b[i], &c[i], CHUNK);
     add(&a[i], &b[i], &c[i], scalar, CHUNK);
     triad(&a[i], &b[i], &c[i], scalar CHUNK);
} }
```

device = openacc and smp are requested
copy_deps = Copies dependencies to the target if it's required

Dependencies are specified

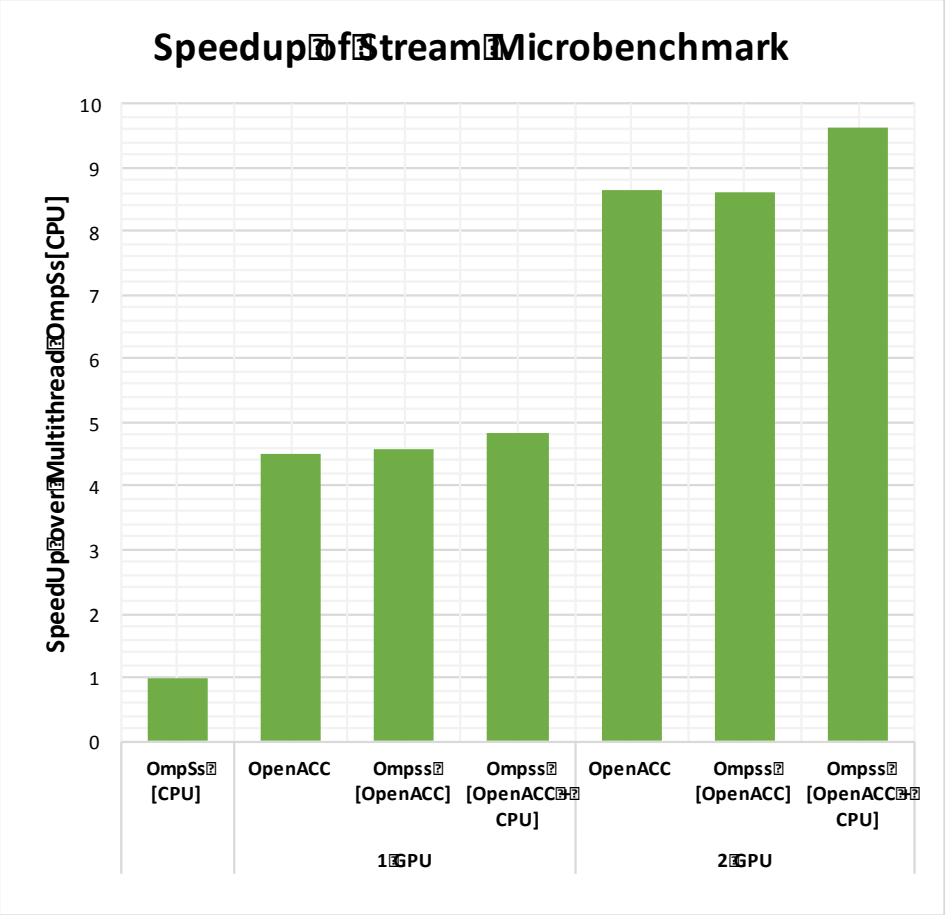
OmpSs manages data.
Symbols are passed deviceptr clause to inform OpenACC

Loop Blocking

Stream Benchmark

- **Configuration:**
  - i7-4820K 4 core
  - NVIDIA Tesla 2 x K 40
  - OpenMP = OmpSs 16.06
  - OpenACC = PGI OpenACC 16.9

- **Note:**
  - All executables use same GPU kernel
  - OpenACC is scaled manually across multiple GPUs modifying code



Future work

- Bring smoother OpenACC code generation by OmpSs

- Include new OpenACC directives
  - “routine”
  - “declare”
  - etc.

Conclusion

- OpenACC integration is feasible as OmpSs is flexible
- Using OpenACC + OmpSs delivers best performance
  - Two heads are better than one !!!

OpenACC become even easier with OmpSs

Guray Ozen
guray.ozen@bsc.es