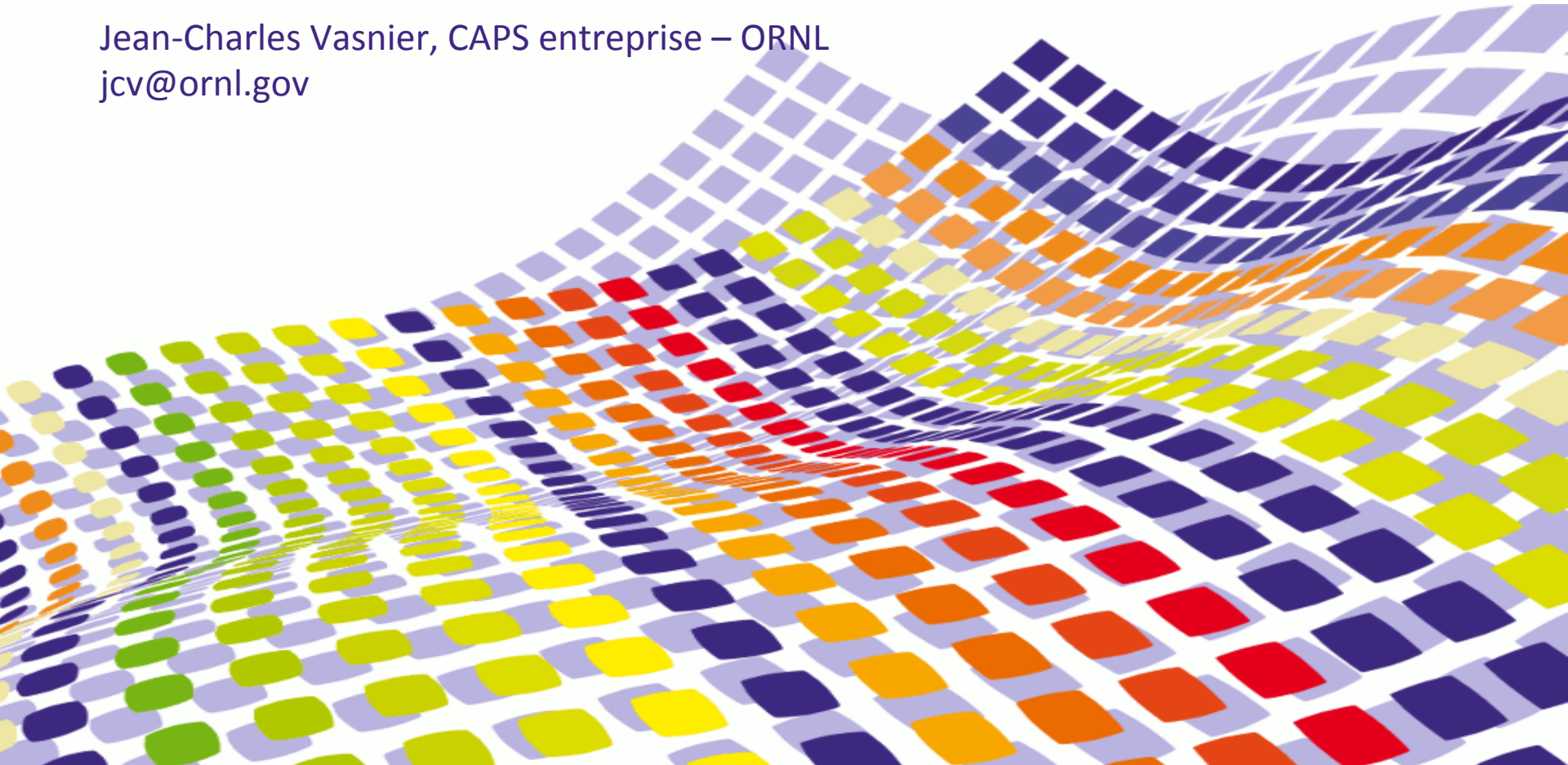
OpenACC using CAPS Compiler

CAPS OpenACC Compiler
XK6 Workshop – ORNL – October 9th 2012

Jean-Charles Vasnier, CAPS entreprise – ORNL
jcv@ornl.gov

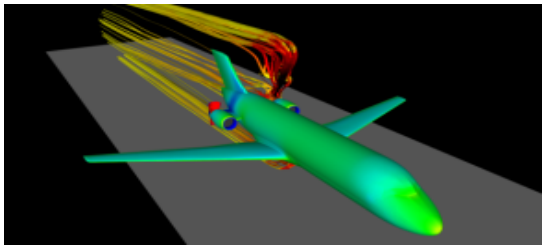
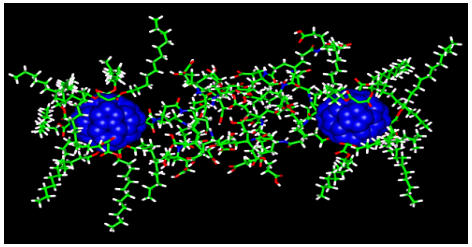


Agenda

- Introduction to GPU Computing
- Directive-based Programming
- OpenACC Standard
- CAPS OpenACC Compiler
- Porting DNADist Application
  - Simple Port with OpenACC
  - A Better Balancing of Computations
  - Optimizing Transfers
- Conclusion

Why Hybrid Computing?

- Modeling & Simulation are pervasive


- Precision is the key to success
- More precision implies more data to process
- Current technologies reached a limit in terms of frequency
  - One solution is to increase the number of cores
- Massively data-parallel architectures are increasingly used as accelerators
  - GPUs

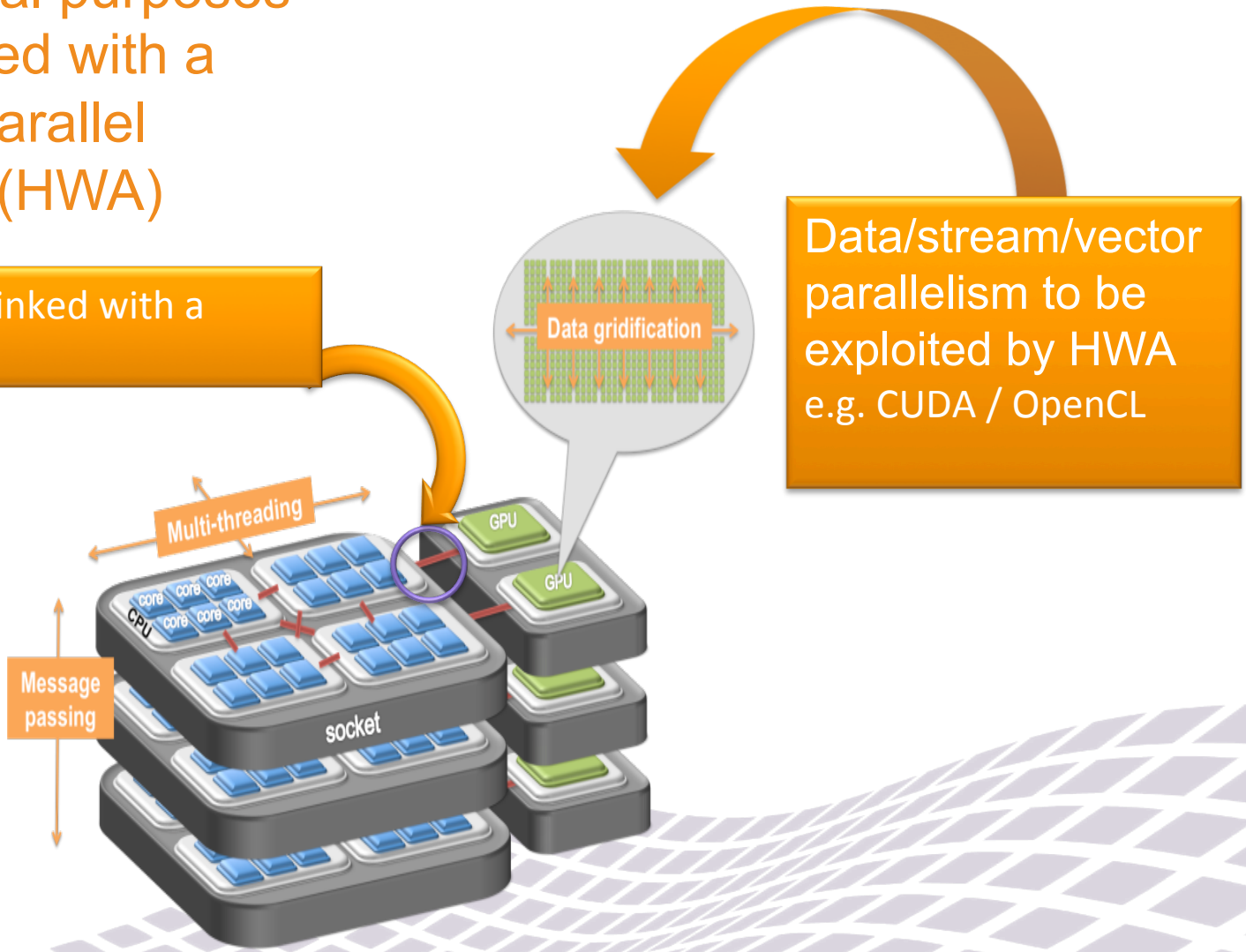
Introduction

- Programming many-core systems faces the following dilemma
  - The constraint of keeping a unique version of codes, preferably mono-language
    - Reduces maintenance cost
    - Preserves code assets
    - Less sensitive to fast moving hardware targets
    - Codes last several generations of hardware architecture
  - Achieve "portable" performance
    - Multiple forms of parallelism cohabiting
      - Multiple devices (e.g. GPUs) with their own address space
      - Multiple threads inside a device
      - Vector/SIMD parallelism inside a thread
    - Massive parallelism
      - Tens of thousands of threads needed
- For legacy codes, directive-based approach may be an alternative

Heterogeneous Many-Cores

- Many general purposes cores coupled with a massively parallel accelerator (HWA)

CPU and HWA linked with a PCIx bus



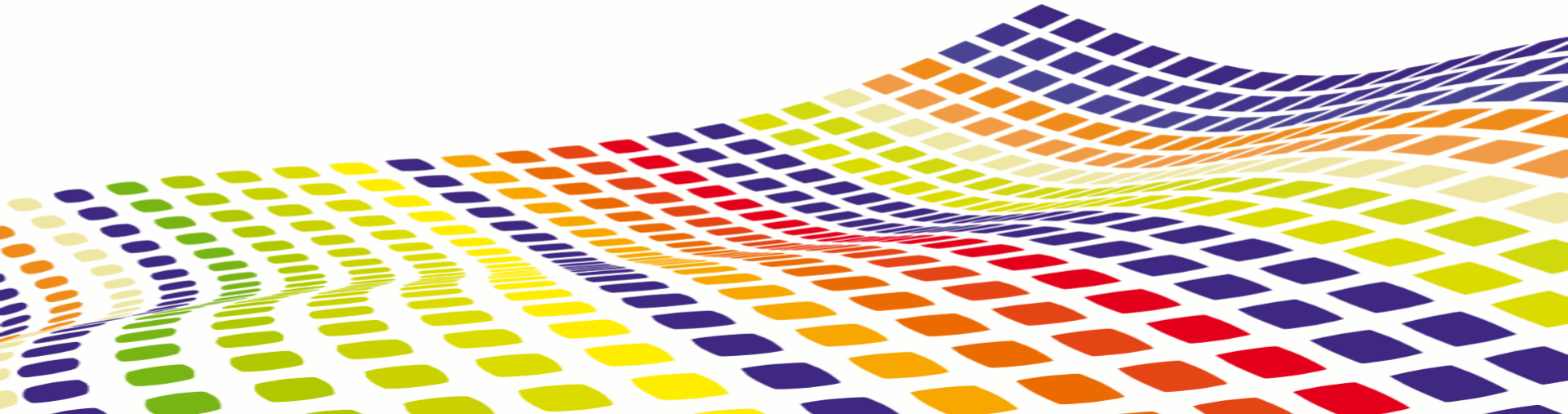
Profile of a Legacy Application

- Written in C/C++/Fortran
- Mix of user code and library calls
- Hotspots may or may not be parallel
- Lifetime in 10s of years
- Cannot be fully re-written
- Migration can be risky and mandatory

```
while(many){
  ...
  mylib1(A,B);
  ...
  myuserfunc1(B,A);
  ...
  mylib2(A,B);
  ...
  myuserfunc2(B,A);
  ...
}
```

OAK RIDGE National Laboratory CAPS

Directive-based Programming



Directive-based Programming

- Three ways of programming on GPU accelerators:

Libraries	Directives	Programming Languages
Ready-to-use Acceleration (e.g.: cuBLAS, cuFFT)	*Quickly Accelerate Existing Applications (e.g.: OpenACC, OpenHMPP)*	*Maximum Performance (e.g: CUDA, OpenCL, DirectCompute)*

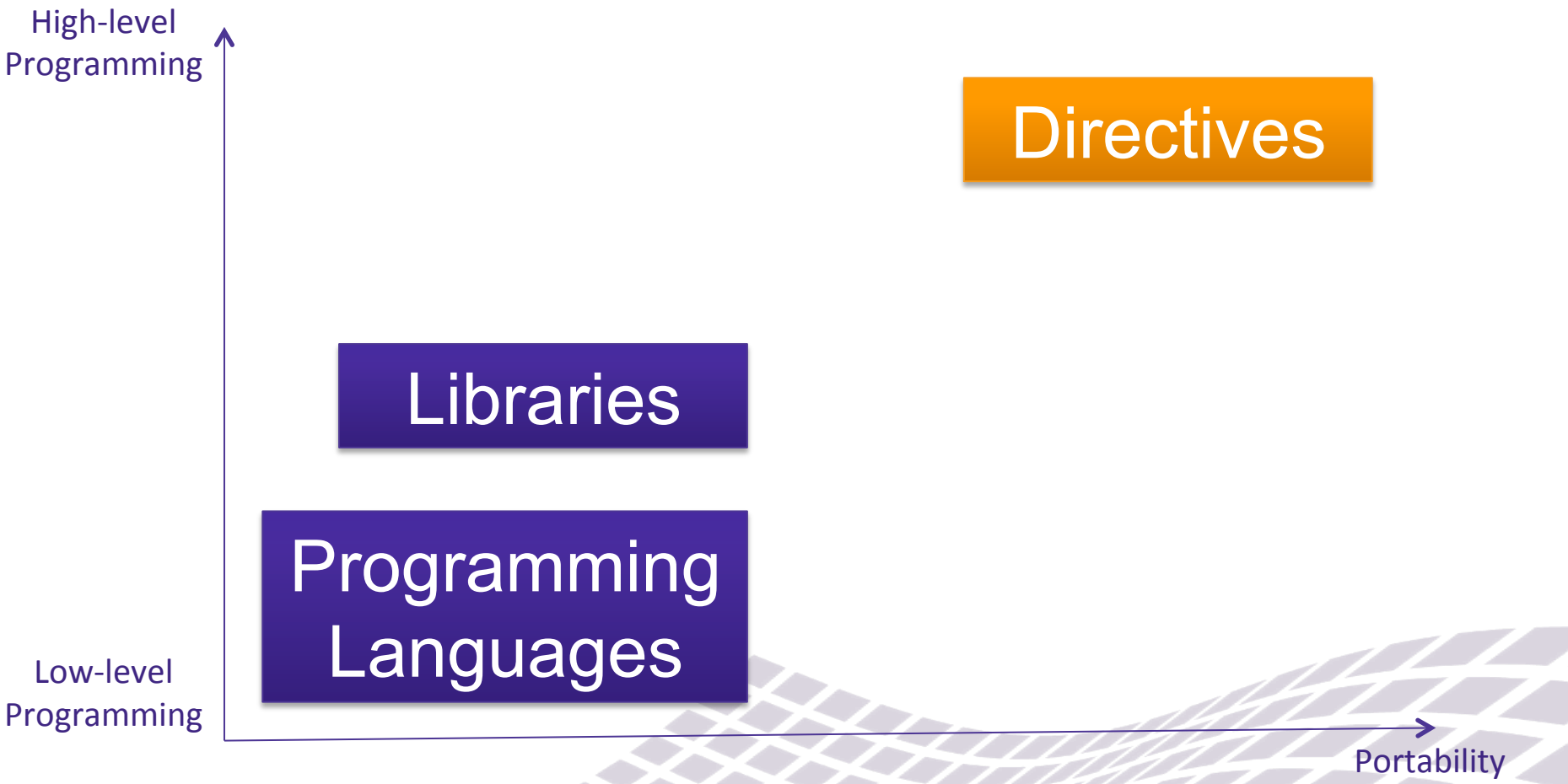
Computing onto GPUs

- Computing on a GPU requires either the use of:
  - An API library (CUDA,...)
  - or a directive set (OpenACC,…)

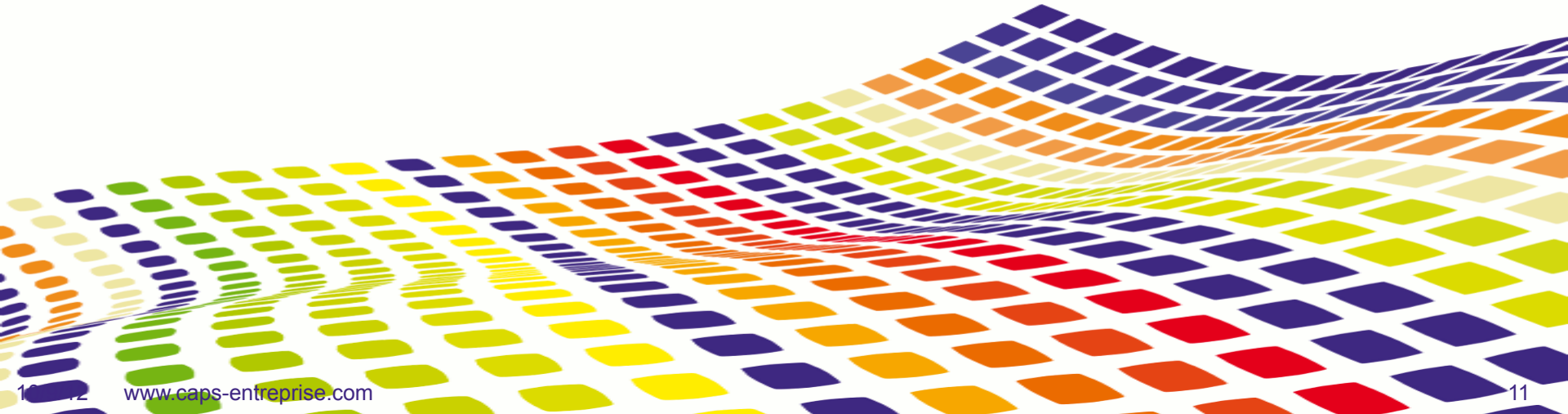
```
#include <cuda.h>
...
__global void myKernel(float * in, float * out)

int main void(){
...
cudaMalloc(&deviceIn_p, sizeInBytes);
cudaMalloc(&deviceOut_p, sizeInBytes);
...
cudaMemcpy(deviceIn, hostIn, sizeInBytes,...);
grid(...);
block(...);

myKernel<<< grid, block>>>(deviceIn_p, deviceOut_p);

cudaMemcpy(hostOut, deviceOut, sizeInBytes,...);
```

```
int main void(){
...

//parallelize what's following
#pragma acc kernels
compute(bigDataSet);

...

//but don't parallelize that
compute(tinyDataSet);
```

High-level and Portable



The OpenACC Standard



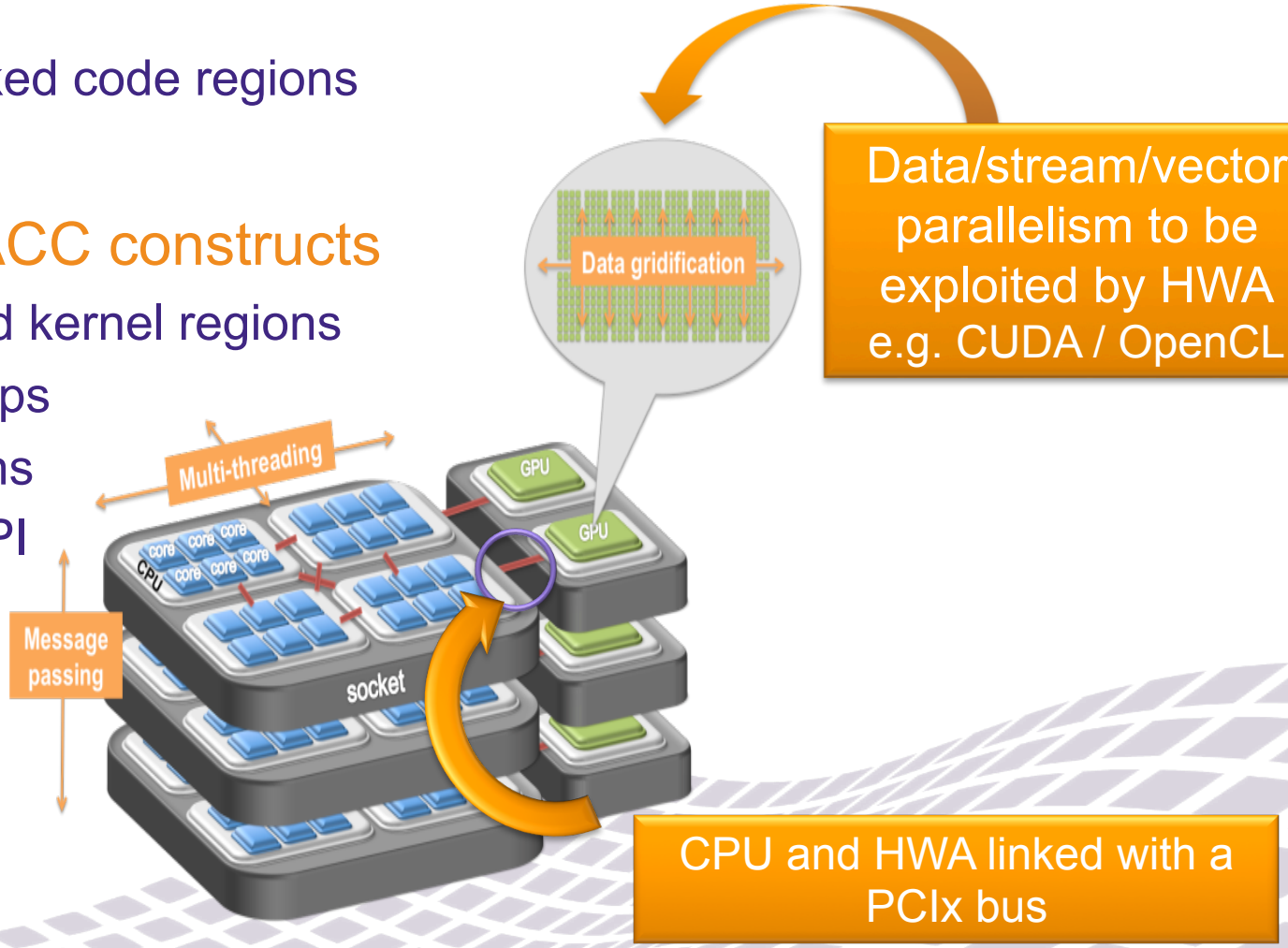
OpenACC Initiative

- A CAPS, CRAY, Nvidia and PGI initiative
- Open Standard
- A directive-based approach for programming heterogeneous many-core hardware for C and FORTRAN applications
- Available for implementation
  - CRAY Compiler, PGI Accelerator, CAPS OpenACC Compiler …

- Visit http://www.openacc-standard.com for more information

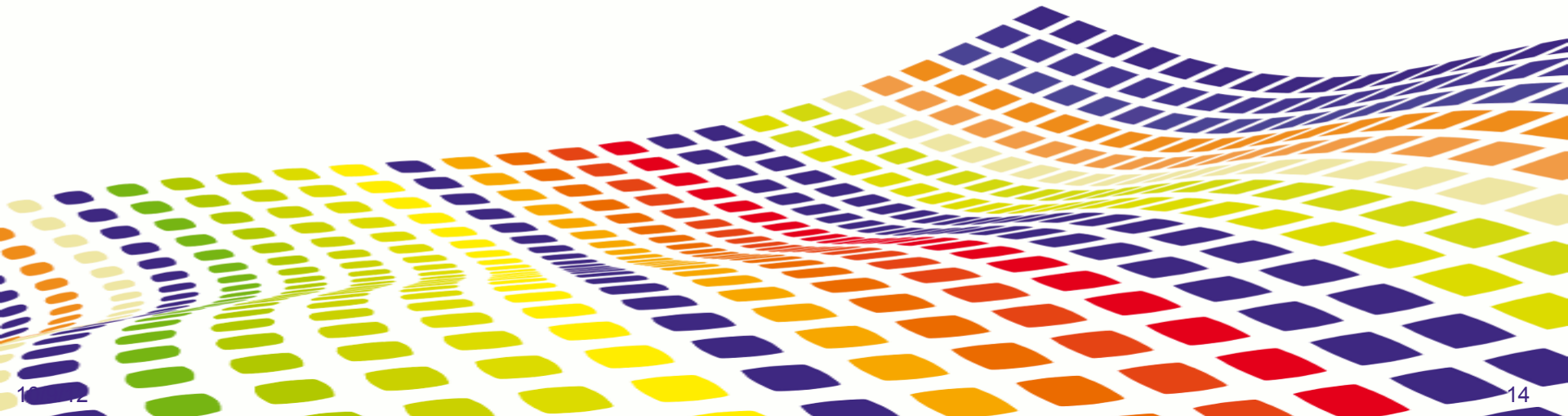
OpenACC Initiative

- Express data and computations to be executed on an accelerator
  - Using marked code regions
- Main OpenACC constructs
  - Parallel and kernel regions
  - Parallel loops
  - Data regions
  - Runtime API



OAK RIDGE National Laboratory CAPS

OpenACC Programming Model



Execution Model

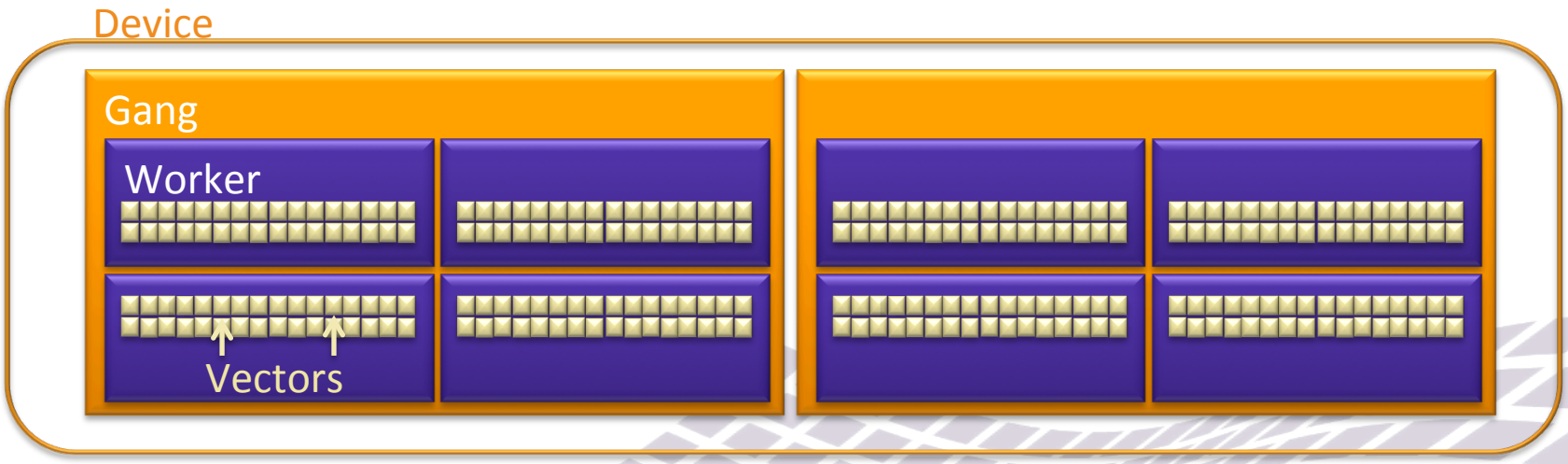
- Among a bulk of computations executed by the CPU, some regions can be offloaded to hardware accelerators

- Host is responsible for:
  - Allocating memory space on accelerator
  - Initiating data transfers
  - Lauching computations
  - Waiting for completion
  - Deallocating memory space

- Accelerators execute parallel regions:
  - Use work-sharing directive
  - Specify level of parallelization

OpenACC Kernel Execution Model

- Host-controlled execution
- Based on three parallelism levels
  - Gangs – coarse-grain
  - Workers – fine-grain
  - Vectors – Finest grain



Work Management: Parallel Construct

- Starts parallel execution on the accelerator
- A parallel construct is compiled into one kernel
- Creates a number of gangs and workers that remains constant for the parallel region

```
#pragma acc parallel […]
{
  …
  for(i=0; i < n; i++) {
    for(j=0; j < n; j++) {
      …
    }
  }
  …
}
```

```
$!acc parallel […]

  …
  DO i=0,n
    DO j=0,n
      …
    END DO
  END DO
  …
$!acc end parallel
```

Parallel Construct: Gangs and Workers

- The clauses:
  - *num_gangs*
  - *num_workers*
- Enables to specify the number of gangs and workers in the corresponding *parallel* section

```
#pragma acc parallel, num_gangs[32], num_workers[256]
{
  …
  for(i=0; i < n; i++) {
    for(j=0; j < n; j++) {
      …
    }
  }
  …
}
```

Work distribution over 32 gangs and 256 workers

Work Management: Kernels Construct

- Kernels construct is compiled into a sequence of accelerator kernels
  - Typically, each loop nest will be a distinct kernel
- The number of gangs and workers can be different for each kernel

```
#pragma acc kernels […]
{
  for(i=0; i < n; i++) {
    …
  }
  …
  for(j=0; j < n; j++) {
    …
  }
}
```

```
$!acc kernels […]

  DO i=1,n
    …
  END DO
  …
  DO j=1,n
    …
  END DO

$!acc end kernels
```

1st Kernel

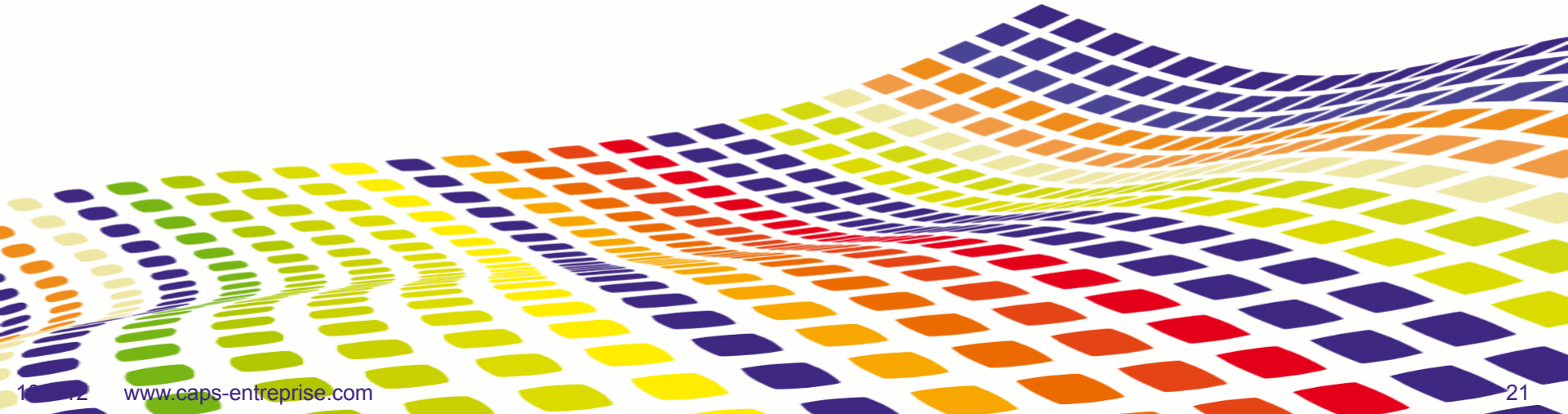
2nd Kernel

OpenACC *Kernels* Regions

- Identifies sections of code to be executed on the accelerator
- Parallel loops inside a *Kernels* region are turned into accelerator kernels
  - Such as CUDA or OpenCL kernels
  - Different loop nests may have different gridifications

```
#pragma acc kernels
{
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < n; ++j){
      for (int k = 0; k < n; ++k){
        B[i][j*k%n] = A[i][j*k%n];
      }
    }
  }
  ...
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < m; ++j){
      B[i][j] = i * j * A[i][j];
    }
  }
  ...
}
```

Loops



Kernel Optimization: Loop Construct

- *Loop* directive applies to a loop that immediately follow the directive
- Describes what kind of parallelism to use
  - According to the distribution among gangs, workers and vectors
  - If the iterations of the loop are data independent or sequential

```
#pragma acc kernels
{
  #pragma acc loop […]
  for(i=0; i<n; i++)
  {
    …
  }
}
```

```
$!acc kernels

  $!acc loop […]
  DO i=1,n
    …
  END DO
  $!acc end loop […]
$!acc end kernels
```

OpenACC *Loop independent* Directive

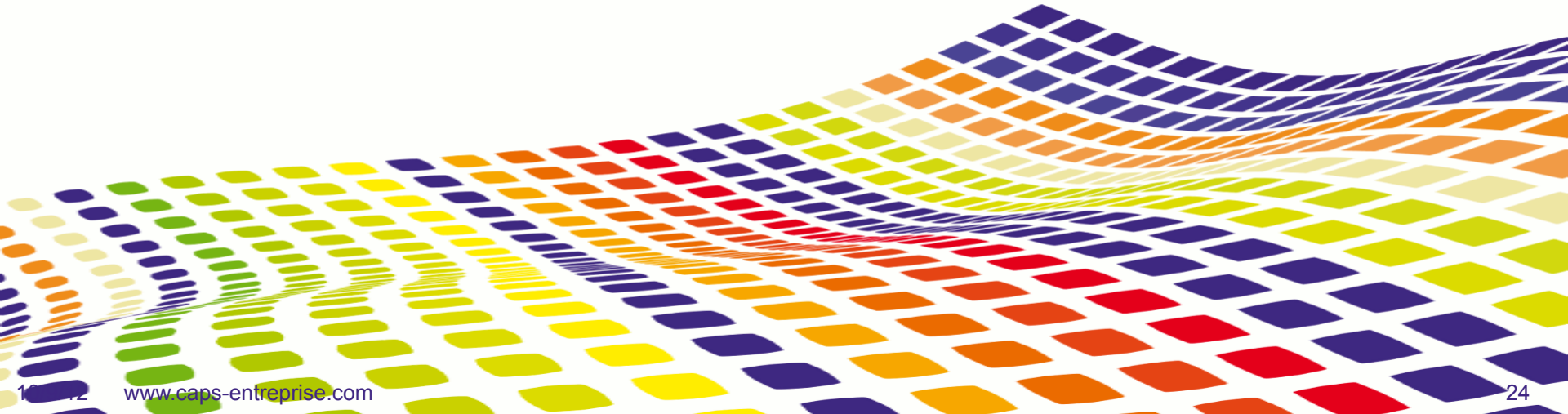
- Inserted inside *Kernels* regions
- Describes that a loop is data-independent
- Other clauses can declare loop-private variables or arrays, and reduction operations

```
#pragma acc loop independent
for (int i = 0; i < n; ++i){
  A[i*m+0] = 1;
  for (int j = 1; j < m; ++j){
    A[i*m+j] = i * j * A[i*m+j-1];
  }
}
```

Iteration s of variable *i* are data independent

Iterations of variable *j* are not data independent

Managing Data



Data Management: Data Constructs

- Defines scalars, arrays and subarrays to be allocated on the device memory for the duration of the region

```
#pragma acc data […]
{
   …
   #pragma acc kernels
   {
      …
   }
   …
}
```

```
$!acc data […]

   …
   $!acc kernels
   …
   $!acc end kernels
   …

$!acc end data
```

Transfers: Copy Clause

- Declares data that need to be copied from the host to the device when entering the data section
- These data are assigned values on the device that need to be copied back to the host when exiting the data section

```
#pragma acc data, copy (A)
{
  …
  #pragma acc kernels
  {
    …
  }
  …
}
```

```
$!acc data, copy (A)

  …
  $!acc kernels
  …
  $!acc end kernels
  …

$!acc end data
```

Transfers: Copyin/Copyout Clause

- Transfers are expensive
  - Data communications use the PCIe bus
- To optimize an application, useless transfers should be avoided

- *Copyin* clause
  - Declares data that need to be copied from the host to the device when entering the data section
- *Copyout* clause
  - Declares data that need to be copied from the device to the host when exiting data section

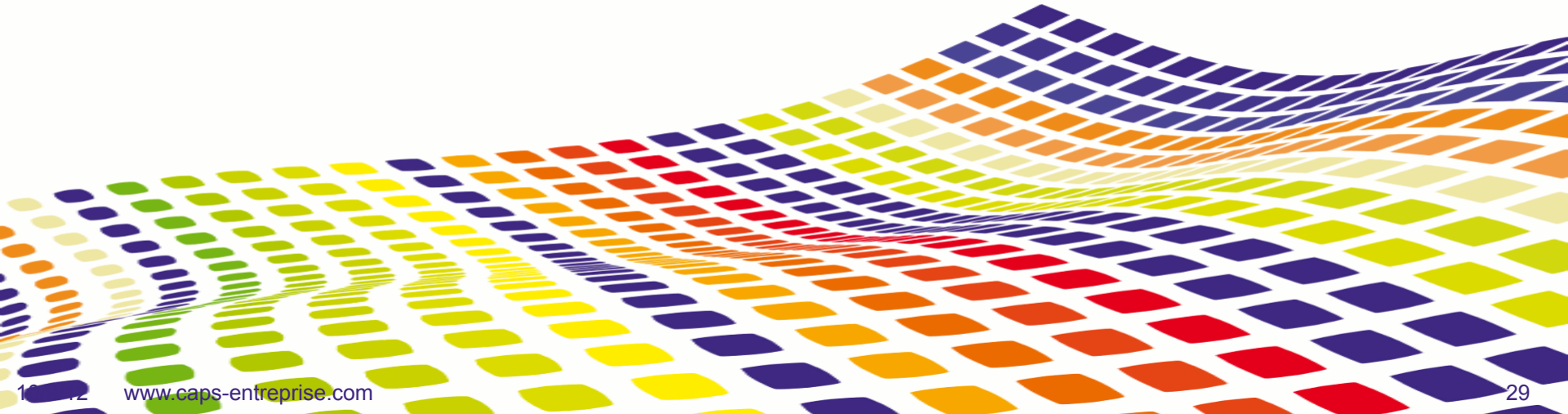
Transfers: Present_or_copy Clauses

- If data is already present, use value in the device memory
- If not:
  - Allocates data on device and copies the value from the host at region entry for *present_or_copy / present_or_copyin* clauses
  - Copies the value from the device to the host and deallocate memory at region exit for *present_or_copy / present_or_copyout* clauses
- May be shortened to *pcopy / pcopyin / pcopyout*

```
#pragma acc data, pcopy (A)
{
  …
  #pragma acc kernels
  {
    …
  }
  …
}
```

```
$!acc data, pcopy (A)

  …
  $!acc kernels
  …
  $!acc end kernels
  …

$!acc end data
```

How to Use an OpenACC Compiler



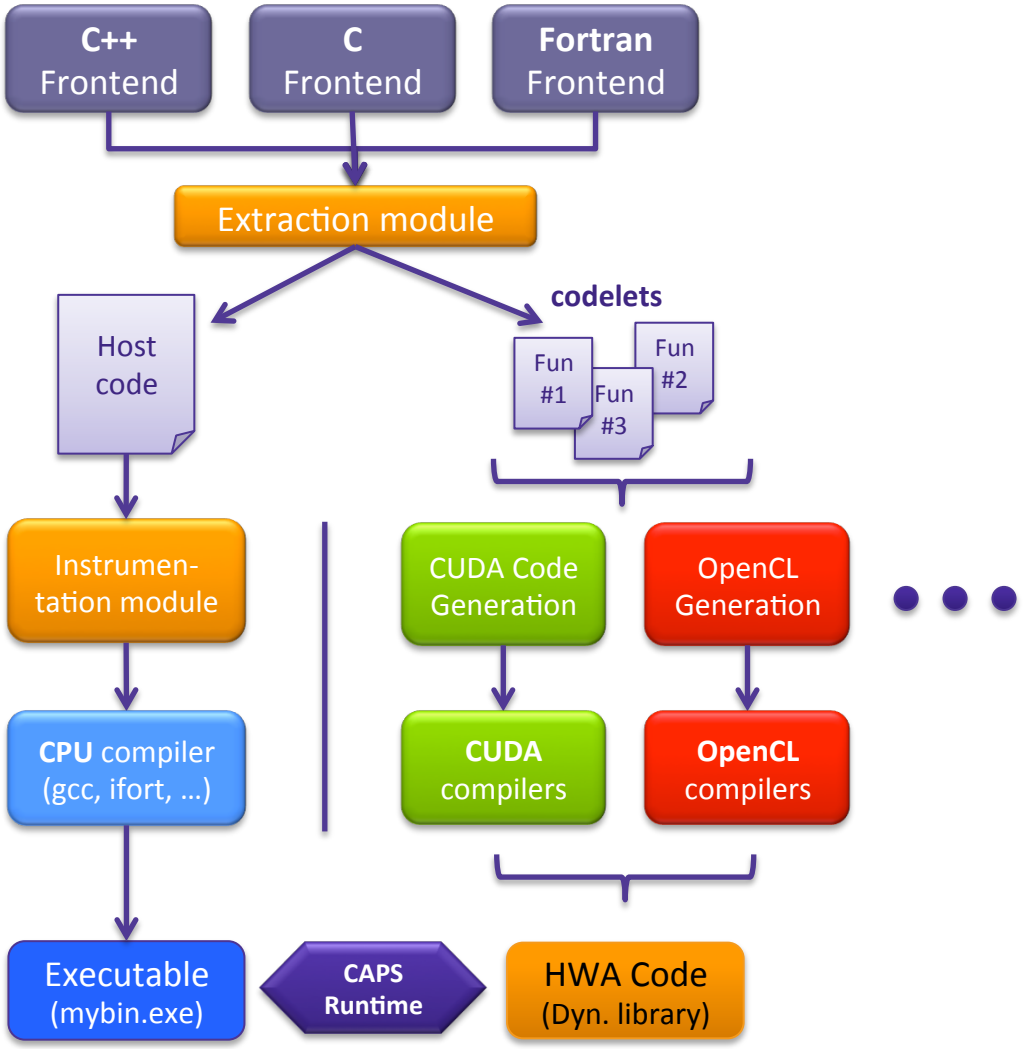
How to Use an OpenACC Compiler

- Here is an example with an OpenACC compiler from CAPS:

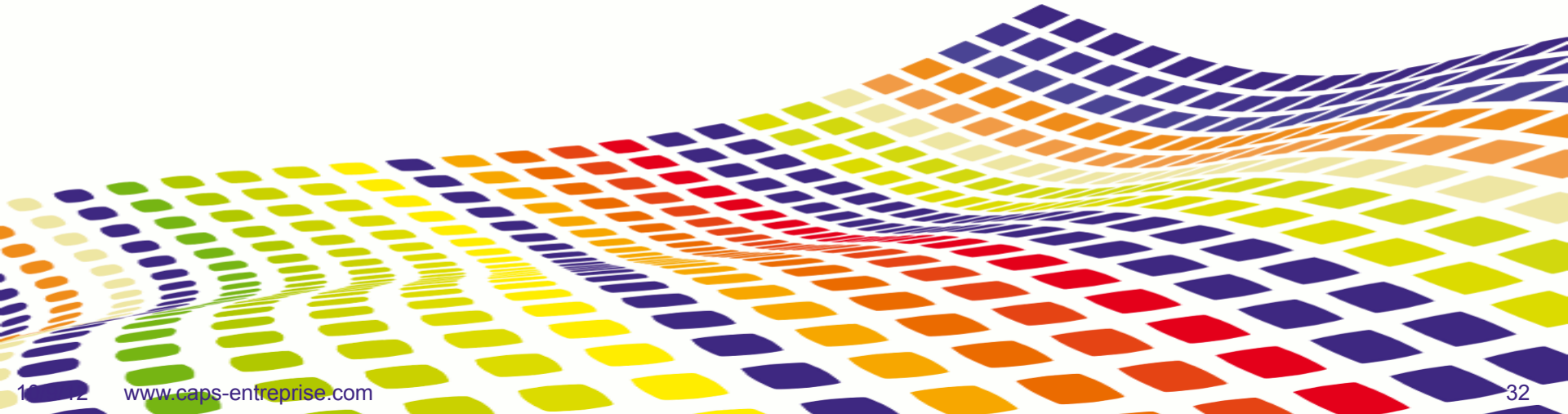
```
$ hmpp gcc myopenaccapp.c -o myopenaccapp.exe
```

- You can also use OpenACC compilers from:
  - PGI
  - CRAY

CAPS OpenACC Compiler flow



Porting the DNADist Application

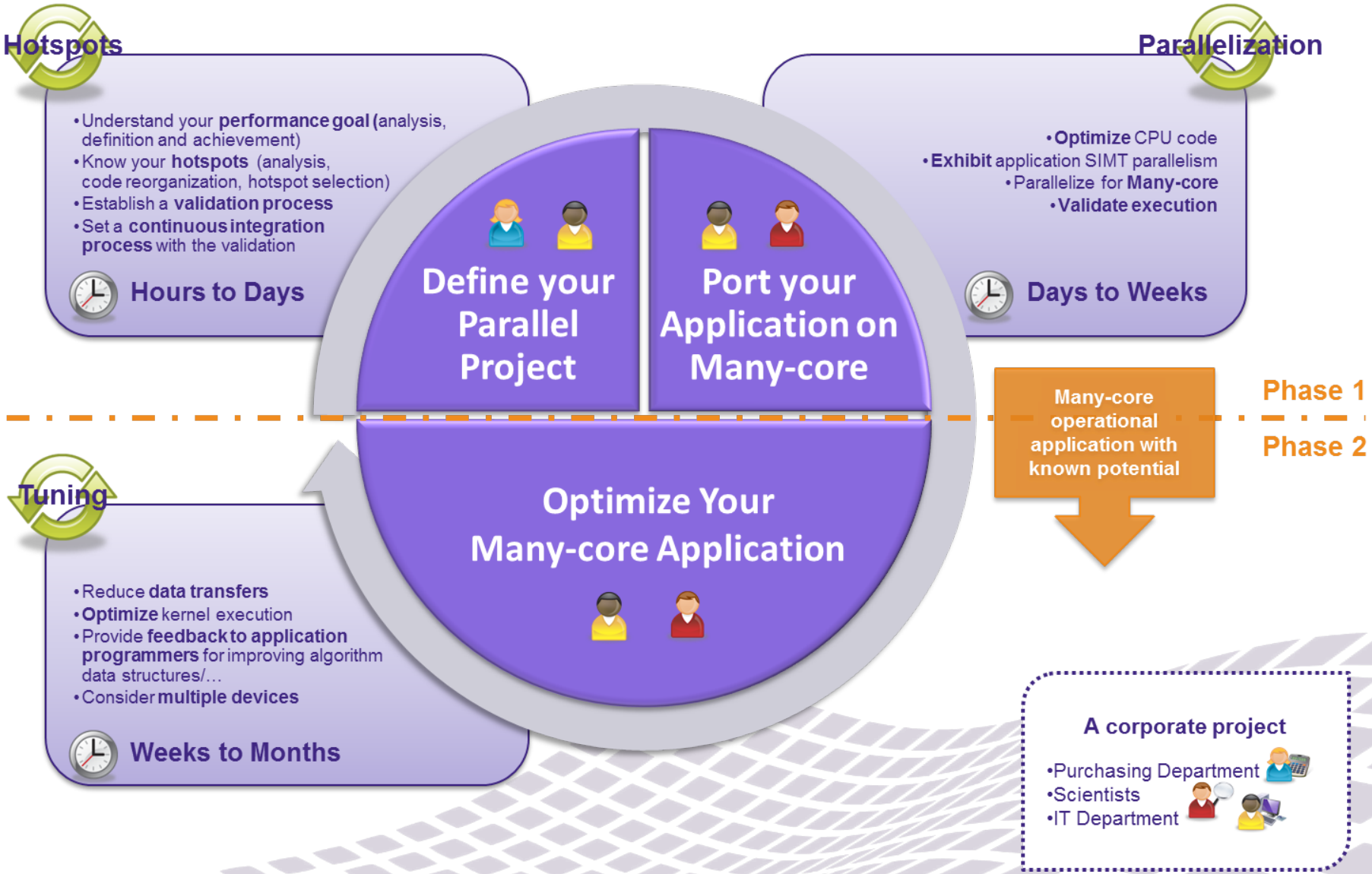


DNA Distance Application with OpenACC

- Biomedical application
  - Part of Phylip package
  - Main computation kernel takes as input a list of DNA sequences for several species
  - Code is based on an approximation using Newton-Raphson method (SP)
  - Produces a 2-dimension matrix of distances

- Comes in two versions
  - CPU optimized version
    - And its OpenMP implementation for 4 & 8 threads
    - x5 speedup with Gcc on Intel i7 CPU 920 @2.67GHz
      - http://itis.grid.sjtu.edu.cn/download/8.DNAdist_by_Yin.pdf
  - GPU-friendly version
    - Provided by Jiao Tong University
      - http://competencecenter.hmpp.org/category/hmpp-coc-asia/
    - Ported to GPU with OpenACC on a **Nvidia Tesla c2070**

DNADist: Migration Steps



Step #1: First Port

- This DNADist version mainly consists in one kernel

```
void makev_kernel ( int spp, int endsite,
    TYPE h_sitevalues[spp * endsite * 4],
    TYPE weightrat[endsite],
    TYPE ratxv, TYPE rat, TYPE xv,
    TYPE freqa, TYPE freqc, TYPE freqg,
    TYPE freqt, TYPE freqar, TYPE freqgr,
    TYPE freqcy, TYPE freqty, TYPE fracchange,
    TYPE h_vvs[spp * spp])
{


      for ( y = 0 ; y < spp ; y++) {
        for ( x = 0 ; x < spp ; x++) {
          //DNADist computations are here
          …
          sitevalue10 = h_sitevalues[i * 4  * spp + x];
          …
          tmp1= weightrat[i] * (zlzz  * (bb – aa) + zlxv * (cc – bb));
          …
          h_vvs[y * spp + x] = mid * fracchange;
          …
        }
      }

}
```

Step #1: First Port

- Offloading computations on the GPU

```
void makev_kernel ( int spp, int endsite,
    TYPE h_sitevalues[spp * endsite * 4],
    TYPE weightrat[endsite],
    TYPE ratxv, TYPE rat, TYPE xv,
    TYPE freqa, TYPE freqc, TYPE freqg,
    TYPE freqt, TYPE freqar, TYPE freqgr,
    TYPE freqcy, TYPE freqty, TYPE fracchange,
    TYPE h_vvs[spp * spp])
{
   #pragma acc kernels copy(h_vvs[0:spp * spp], h_sitevalues[0:spp * endsite * 4], weightrat[0:endsite])
   {
      #pragma acc loop independent
      for ( y = 0 ; y < spp ; y++) {
         for ( x = 0 ; x < spp ; x++) {
            //DNADist computations are here
            …
            sitevalue10 = h_sitevalues[i * 4  * spp + x];
            …
            tmp1= weightrat[i] * (zlzz  * (bb – aa) + zlxv * (cc – bb));
            …
            h_vvs[y * spp + x] = mid * fracchange;
            …
         }
      }
   }
}
```

Step #1: First Port

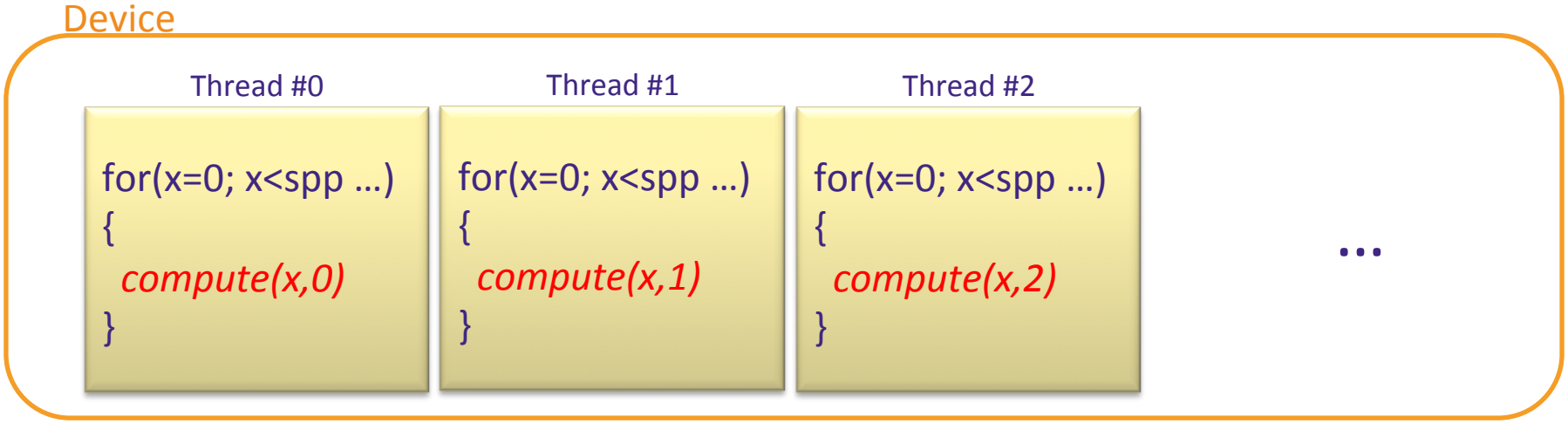
- Compilation with CAPS tools

```
$ hmpp  --nvcc-options -Xptxas=-v  gcc -std=c99 -O2 -Wall -lm  phylip.c seq.c dnadist.c -o DNAdist

hmppcg: [Message DPL3000] /tests/DNAdist/dnadist_kernel.c:57: Loop 'y' was gridified (1D)
…
ptxas info    : Compiling entry function '__hmpp_acc_region__f58ujb3h_loop1D_1' for 'sm_20'
ptxas info    : Function properties for __hmpp_acc_region__f58ujb3h_loop1D_1
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 55 registers, 112 bytes cmem[0], 24 bytes cmem[16]
```

- CAPS OpenACC Compiler creates CUDA code from the original C code
  - Without any tip given to the compiler, the generation is considerered as *naive*
  - At compile time, Nvidia CUDA compiler is called to generate GPU binary
  - x2 speedup compared to original CPU-optimized code
    - This is a slowdown compared to the OpenMP version

Gridification 1D

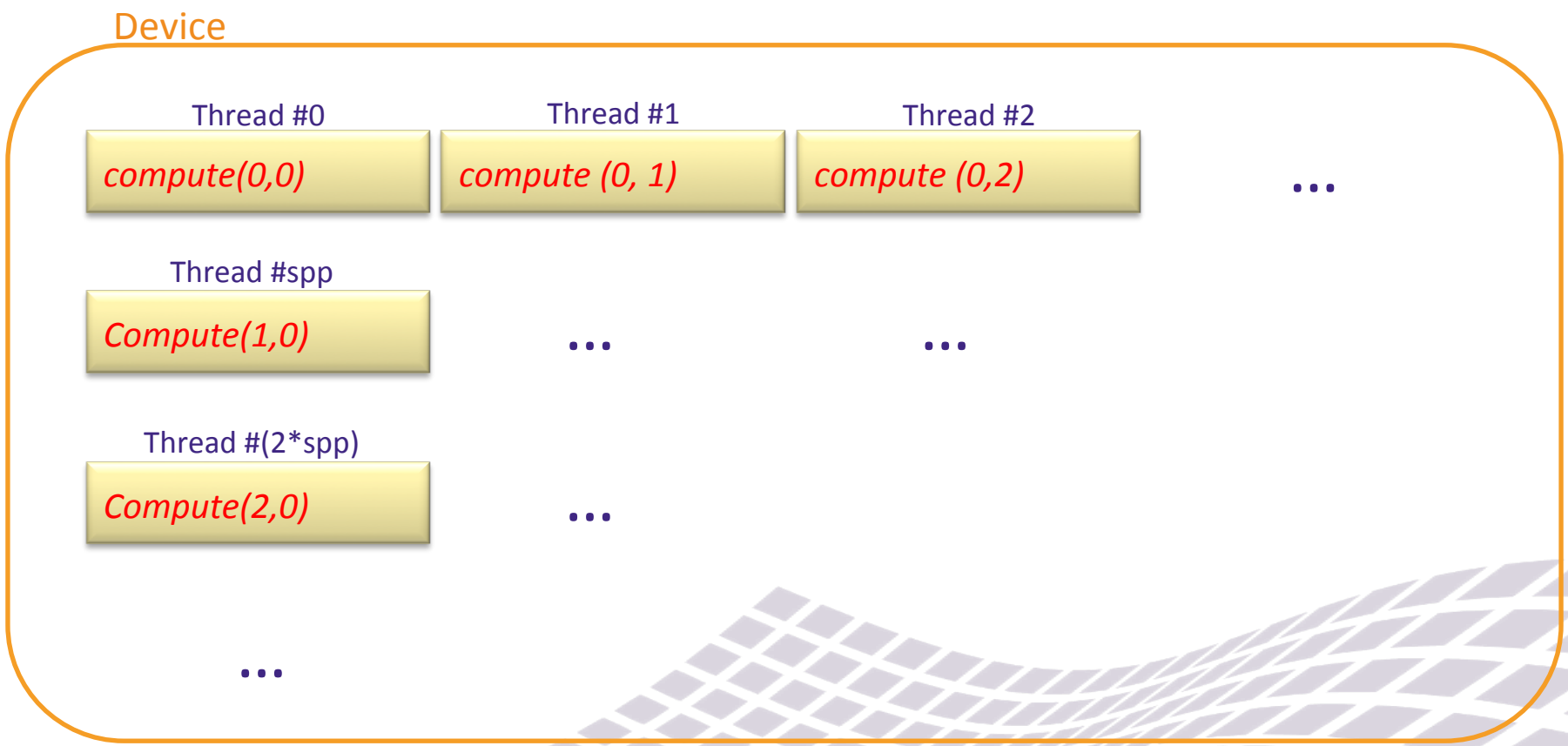
- What happened in the previous code generation



- Consequences
  - Sequential loop execution remains in threads
  - Amount of spawned threads does not fit the entire GPU cores (low occupancy)
    - Architecture is under-exploited which leads to poor performances

Gridification 2D

- How to increase occupancy and reduce sequential code



Step #2: Gridification Optimization

- The gridification may be optimized

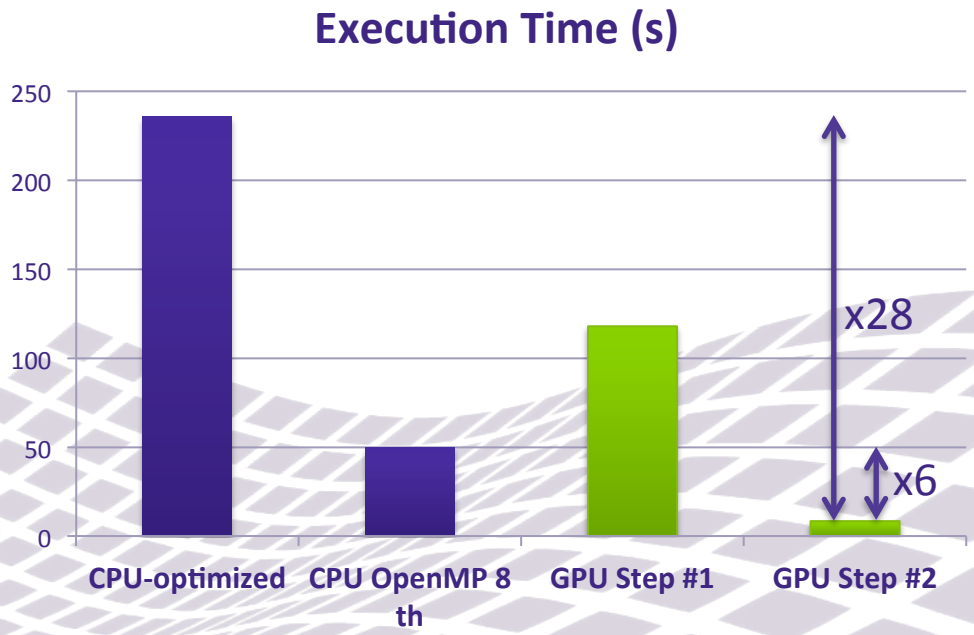
```
void makev_kernel ( int spp, int endsite,
    TYPE h_sitevalues[spp * endsite * 4],
    TYPE weightrat[endsite],
    ....
    TYPE freqcy, TYPE freqty, TYPE fracchange,
    TYPE h_vvs[spp * spp])
{
   #pragma acc kernels copy(h_vvs[0:spp * spp], h_sitevalues[0:spp * endsite * 4], weightrat[0:endsite])
   {
      #pragma acc loop independent
      for ( y = 0 ; y < spp ; y++) {
         #pragma acc loop independent
         for ( x = 0 ; x < spp ; x++) {
            //DNADist computations are here
            …
            sitevalue10 = h_sitevalues[i * 4  * spp + x];
            …
            tmp1= weightrat[i] * (zlzz  * (bb – aa) + zlxv * (cc – bb));
            …
            h_vvs[y * spp + x] = mid * fracchange;
            …
         }
      }
   }
}
```

Step #2: Gridification Optimization

- Feedback from the compiler

```
$ hmpp  --nvcc-options -Xptxas=-v gcc -std=c99 -O2 -Wall -lm  phylip.c seq.c dnadist.c -o DNAdist

hmppcg: [Message DPL3001] /tests/DNAdist/dnadist_kernel.c:57: Loops 'x' and 'y' were gridified (2D)
…
ptxas info    : Compiling entry function '__hmpp_acc_region__f58ujb3h_loop2D_1' for 'sm_20'
ptxas info    : Function properties for __hmpp_acc_region__f58ujb3h_loop2D_1
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 44 registers, 112 bytes cmem[0], 24 bytes cmem[16]
```

- x6 acceleration
  - Compared to OpenMP implementation
  - x28 speedup compared to original sequential CPU-optimized code



Step #3: Transfer Issue

- Code execution feedback

```
$ export HMPPRT_LOG_LEVEL=info
$ ./DNAdist
[    3.054960] ( 0) INFO : Enter    data (queue=none, location=dnadist.c:557)
[CALL kernel makev]: spp=762, endsite=4388
[    5.250125] ( 0) INFO : Enter    kernels (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[    5.250328] ( 0) INFO : Acquire  (target=cuda)

[    5.563183] ( 0) INFO : Allocate h_sitevalues[0:13374624] (element_size=4, memory_space=cudaglob, location=dnadist_kernel.c:24)
[    5.563461] ( 0) INFO : Upload   h_sitevalues[0:13374624] (element_size=4, queue=none, location=dnadist_kernel.c:24)
[    5.573226] ( 0) INFO : Allocate weightrat[0:4388] (element_size=4, memory_space=cudaglob, location=dnadist_kernel.c:24)
[    5.573453] ( 0) INFO : Upload   weightrat[0:4388] (element_size=4, queue=none, location=dnadist_kernel.c:24)
[    5.573498] ( 0) INFO : Allocate h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[    5.573499] ( 0) INFO : Upload h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)

[    5.574648] ( 0) INFO : Call     __hmpp_acc_region__7jpebxwe (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)

[   13.436533] ( 0) INFO : Download h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.436645] ( 0) INFO : Download   h_sitevalues[0:13374624] (element_size=4, queue=none, location=dnadist_kernel.c:24)
[   13.436837] ( 0) INFO : Download   weightrat[0:4388] (element_size=4, queue=none, location=dnadist_kernel.c:24)

[   13.437226] ( 0) INFO : Free     h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.437489] ( 0) INFO : Free     weightrat[0:4388] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.437708] ( 0) INFO : Free     h_sitevalues[0:13374624] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.439122] ( 0) INFO : Leave    kernels (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
```

Step #3: Transfer Optimizations

- Some transfers can be avoided

```
void makev_kernel ( int spp, int endsite,
    TYPE h_sitevalues[spp * endsite * 4],
    TYPE weightrat[endsite],
    ….
    TYPE freqcy, TYPE freqty, TYPE fracchange,
    TYPE h_vvs[spp * spp])
{
  #pragma acc kernels copyin( h_sitevalues[0:spp * endsite * 4], weightrat[0:endsite]) \
                      copyout(h_vvs[0:spp * spp])
  {
     #pragma acc loop independent
     for ( y = 0 ; y < spp ; y++) {
        #pragma acc loop independent
        for ( x = 0 ; x < spp ; x++) {
           …
           sitevalue10 = h_sitevalues[i * 4  * spp + x];
           …
           tmp1= weightrat[i] * (zlzz  * (bb – aa) + zlxv * (cc – bb));
           …
           h_vvs[y * spp + x] = mid * fracchange;
           …
        }
     }
  }
}
```

How to get Feedback at Execution Time?

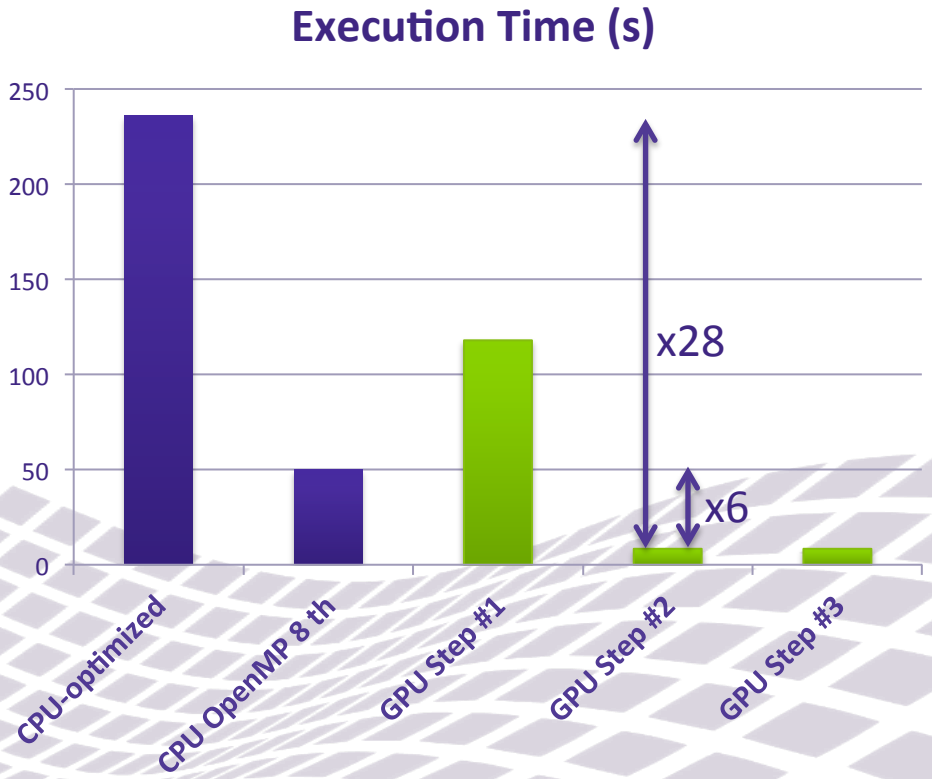
- Code execution feedback

```
$ ./DNAdist
[    3.054960] ( 0) INFO : Enter    data (queue=none, location=dnadist.c:557)
[CALL kernel makev]: spp=762, endsite=4388
[    5.250125] ( 0) INFO : Enter    kernels (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[    5.250328] ( 0) INFO : Acquire  (target=cuda)

[    5.563183] ( 0) INFO : Allocate h_sitevalues[0:13374624] (element_size=4, memory_space=cudaglob, location=dnadist_kernel.c:24)
[    5.563461] ( 0) INFO : Upload   h_sitevalues[0:13374624] (element_size=4, queue=none, location=dnadist_kernel.c:24)
[    5.573226] ( 0) INFO : Allocate weightrat[0:4388] (element_size=4, memory_space=cudaglob, location=dnadist_kernel.c:24)
[    5.573453] ( 0) INFO : Upload   weightrat[0:4388] (element_size=4, queue=none, location=dnadist_kernel.c:24)

[    5.573573] ( 0) INFO : Allocate h_vvs[0:580644] (element_size=4, memory_space=cudaglob, queue=none, location=dnadist_kernel.c:24)

[    5.574648] ( 0) INFO : Call     __hmpp_acc_region__7jpebxwe (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)

[   13.436533] ( 0) INFO : Download h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)

[   13.437226] ( 0) INFO : Free     h_vvs[0:580644] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.437489] ( 0) INFO : Free     weightrat[0:4388] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
[   13.437708] ( 0) INFO : Free     h_sitevalues[0:13374624] (element_size=4, queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)

[   13.439122] ( 0) INFO : Leave    kernels (queue=none, location=/tests/DNAdist/dnadist_kernel.c:24)
```

DNADist: Final Performances

- Final speedup is x28
  - With a Tesla C2070 / 448 CUDA Cores / 6 GB (8.38 s)
  - Including transfers (overall execution time)
  - Compared to the original sequential CPU-optimized code on a Intel Core i7 920 @2.67GHz (236 s)

- Only 3/500 lines of code modified
- Porting time is about one week
  - Starting from this GPU-friendly algorithm provided by Jiao Tong
  - Including project set-up, validation procedure, performance measure system etc.



DNADist: Conclusion

- Directive-based approaches are currently one of the most promising tracks for heterogeneous many-cores
  - Preserve code assets
  - Help separating parallelism description from implementation
  - Porting time is short
  - Create portable applications for fast-moving targets

- DNADist porting is an easy case
  - Small code (500 lines of code in the GPU-friendly version)
  - Few directives
  - Very good performance

➔ And CAPS OpenACC Compiler is the appropriate tool

What Does CAPS OpenACC Compiler Do?

- Generates
  - CUDA **or** OpenCL codes
  - From C **or** Fortran applications (soon C++)
  - For Linux (Windows coming soon)

- Implements the full OpenACC standard
  - Specification v1.0 (november 2011)

- Where to get the compiler

  ➔ www.caps-entreprise.com/purchase/price-list/
  - Starting at $199 (or 199 €)
  - Depending to :
    - Input languages
    - Targets
    - Node-locked/floating
    - Level of support

Going Further with GPUs

- In research and industry, applications are rich and complex
- These applications may need:
  - To integrate hardware-accelerated libraries
  - To integrate user's handwritten codes
  - Multiple GPUs parallelism
  - Directive-based kernel tuning

- The OpenACC does not provide this yet
  - But the consortium is working hard on it

- These features are already provided by the OpenHMPP directive-set
  - ➔ www.caps-entreprise.com/technology/hmpp/

Conclusion

- OpenACC also support asynchronous behavior
- Hardware information can be retrieved with the OpenACC runtime API
- Beware of compiler-dependent behaviors
- Fast development of high-level heterogenous applications
  - For C and FORTRAN code
- Explicit the calls to a hardware accelerator in your code
  - Whatever the target
    - Nvidia GPUs
    - AMD GPUs (CAPS)
    - X86 Intel Phi (CAPS, soon)

Accelerator Programming Model

Parallelization

Directive-based programming

GPGPU

Manycore programming

Hybrid Manycore Programming

HPC community

OpenACC

Petaflops

Parallel computing

HPC open standard

Multicore programming

Exaflops

NVIDIA Cuda

Code speedup

Hardware accelerators programming

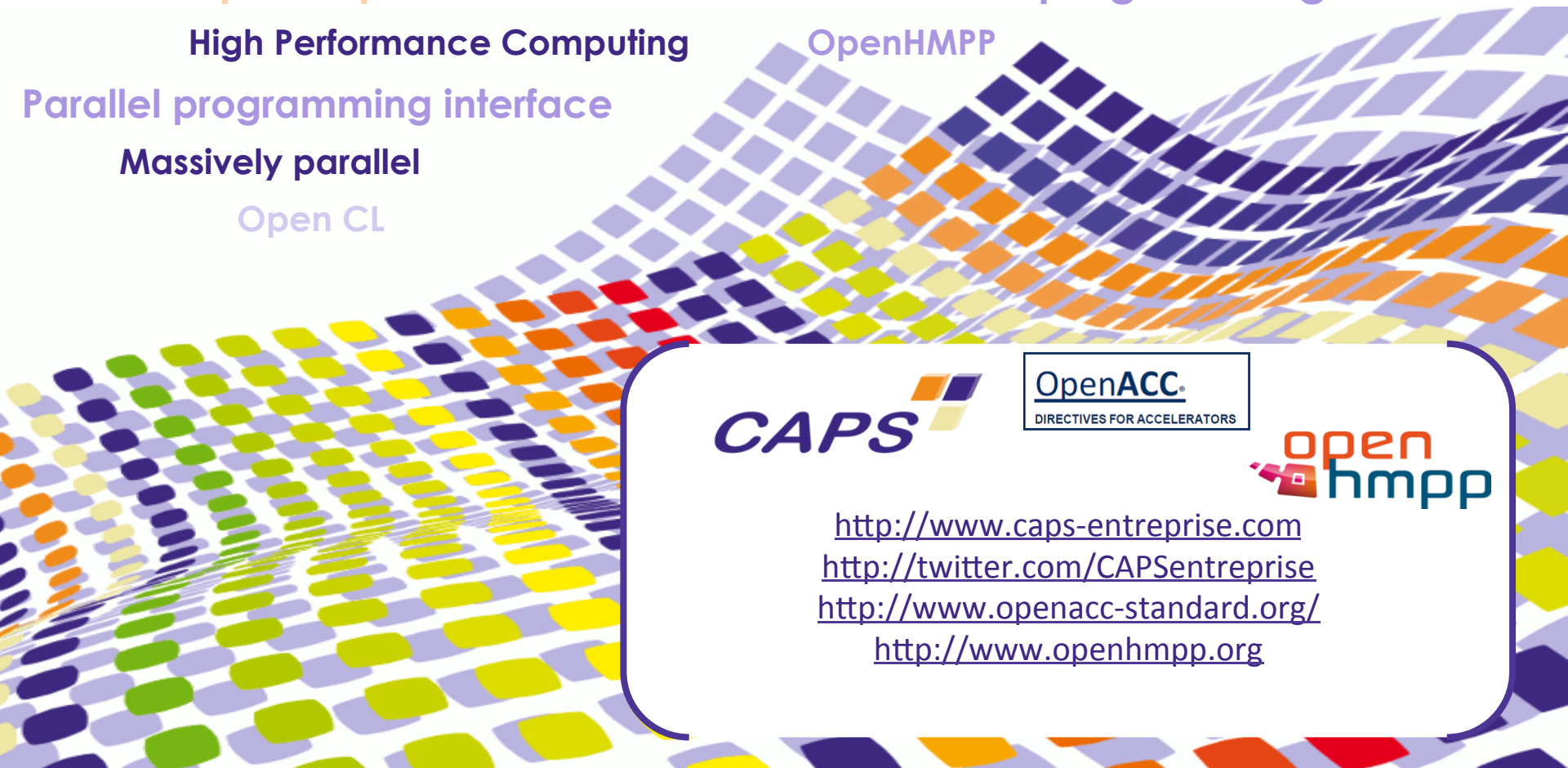
High Performance Computing

OpenHMPP

Parallel programming interface

Massively parallel

Open CL



http://www.caps-entreprise.com
http://twitter.com/CAPSentreprise
http://www.openacc-standard.org/
http://www.openhmpp.org