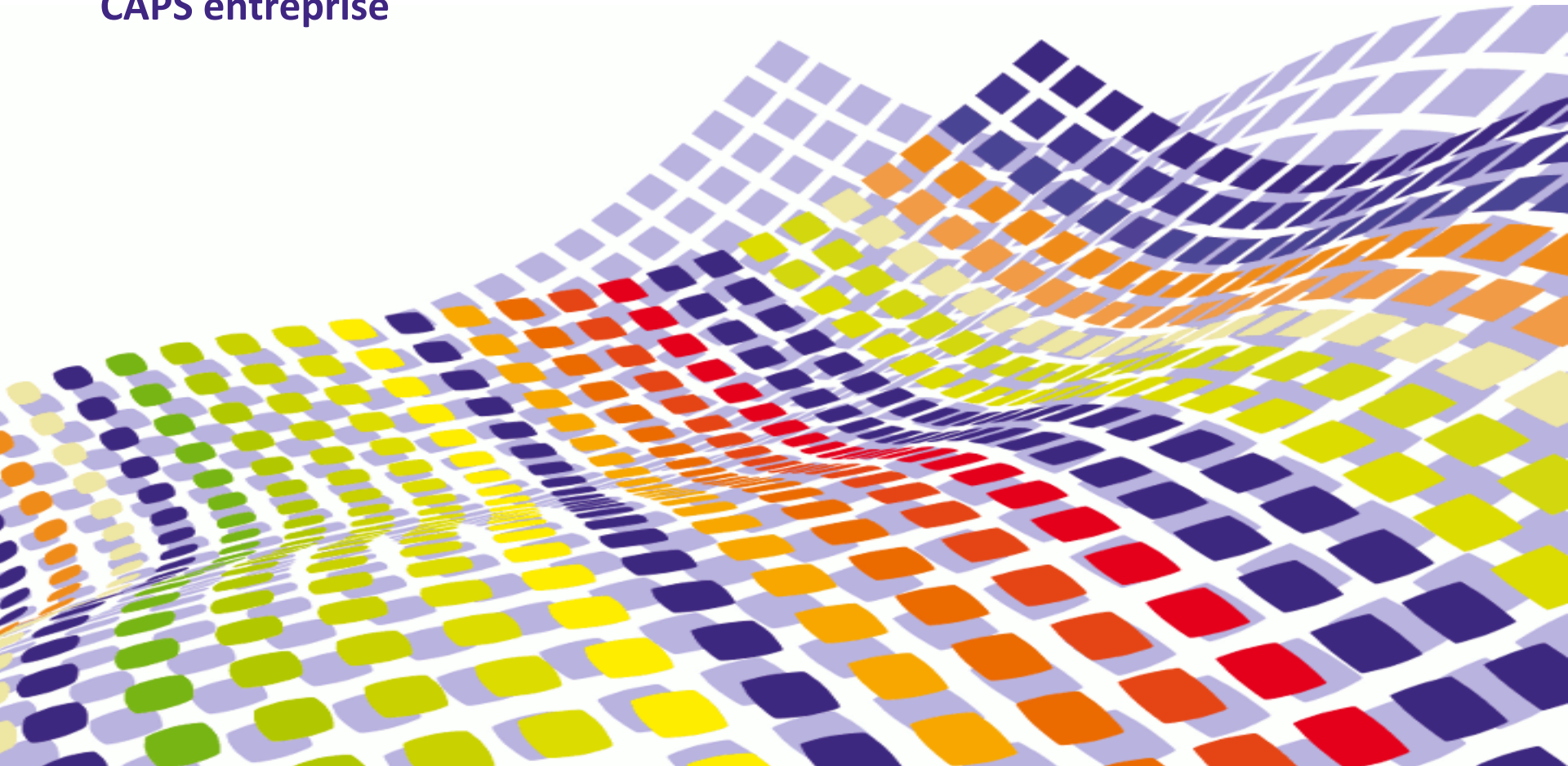
Porting DNADist to GPUs

With OpenACC and CAPS OpenACC Compiler 3.2.1

CAPS entreprise



Agenda

CAPS

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

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);
```

OpenACC Standard

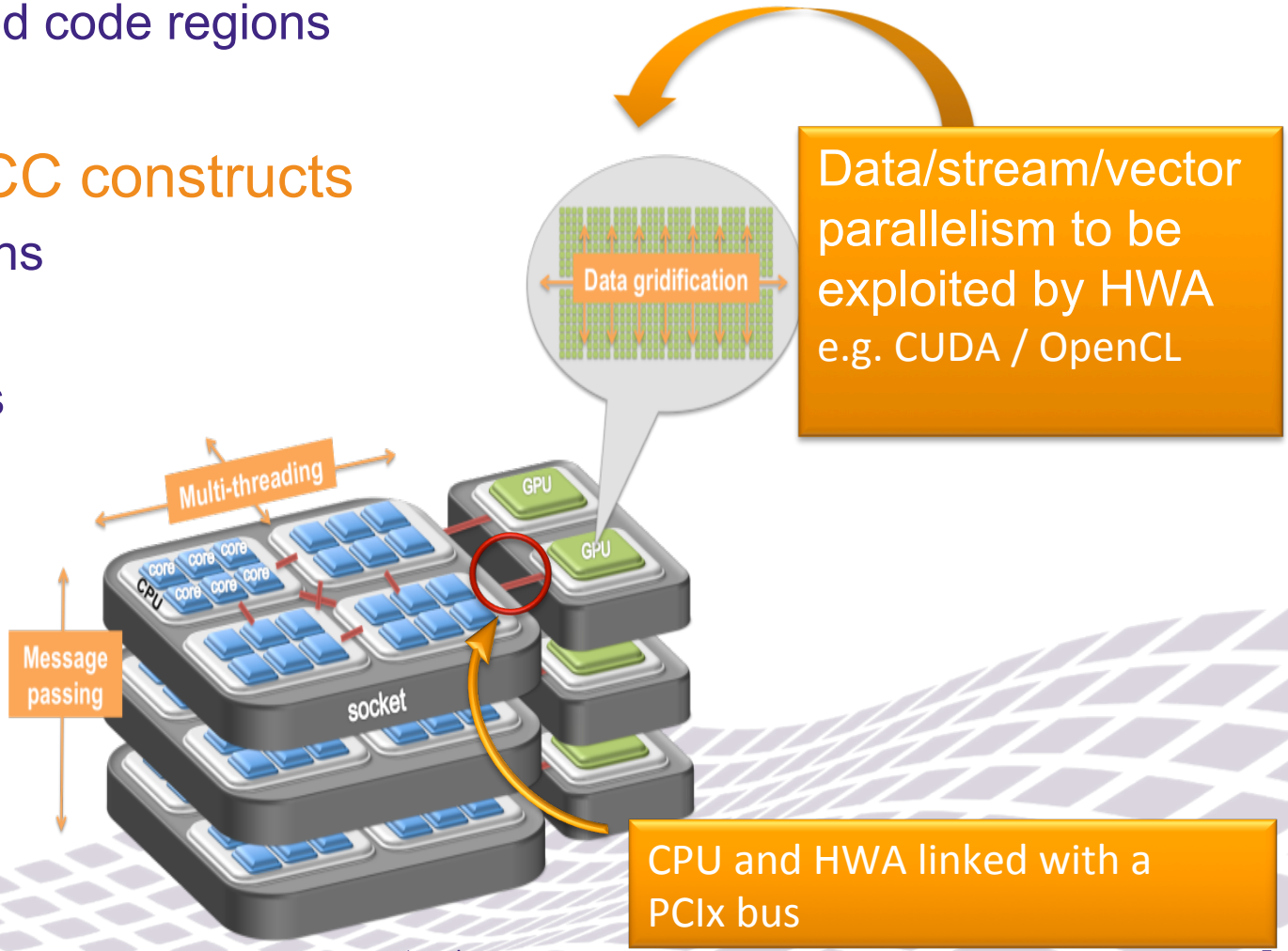
- A CAPS, CRAY, Nvidia and PGI initiative
- Open Standard
- A directive-based approach for programming heterogeneous many-core hardware

- Available for implementation
  - As CRAY, PGI and CAPS ones
  - CAPS OpenACC Compiler ➔ released in April 2012
    - Satisfies the OpenACC Test Suite provided by University of Houston

- http://www.openacc-standard.com

OpenACC Concepts

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



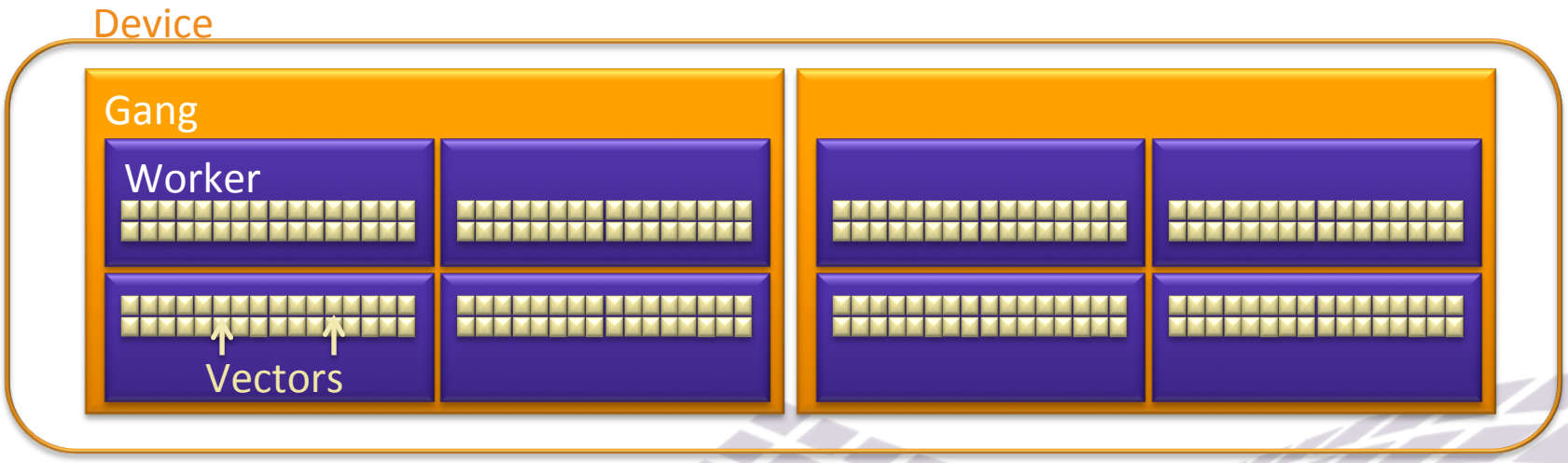
OpenACC *Kernels* Regions

- 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];
    }
  }
  ...
}
```

OpenACC Kernel Execution Model

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



Nvidia CUDA Architecture

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){
  B[i][0] = 1;
  for (int j = 1; j < m; ++j){
    B[i][j] = i * j * A[i][j-1];
  }
}
```

Iteration s of variable *i* are data independent

Iterations of variable *j* are not data independent

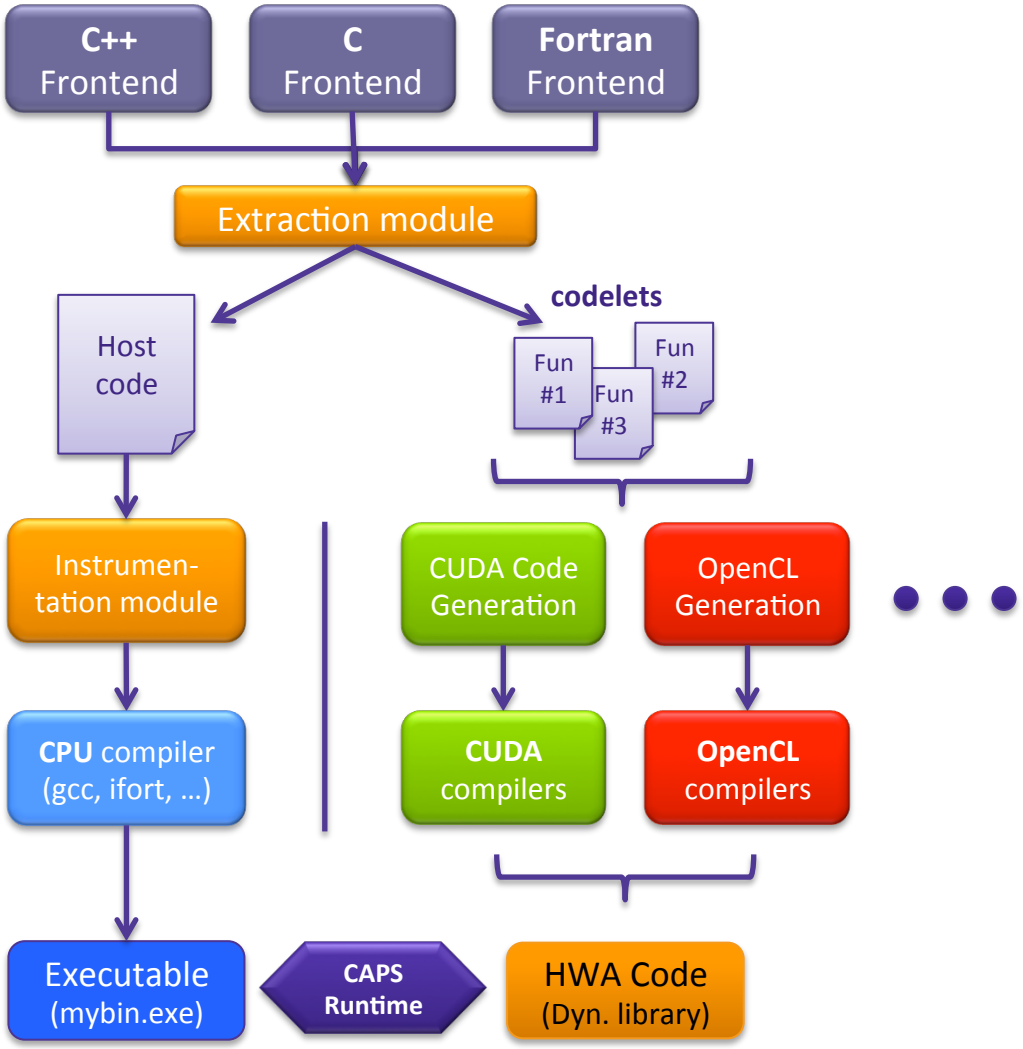
OpenACC *Data* Regions

- *Data* regions define scalars, arrays and sub-arrays
  - To be allocated in the device memory for the duration of the region
  - To be explicitly managed using transfer clauses or directives

- Optimizing transfers consists in:
  - Transferring data
    - From CPU to GPU when entering a data region
    - From GPU to CPU when exiting a data region
  - Launching several kernels
    - That can reuse the data inside this data region

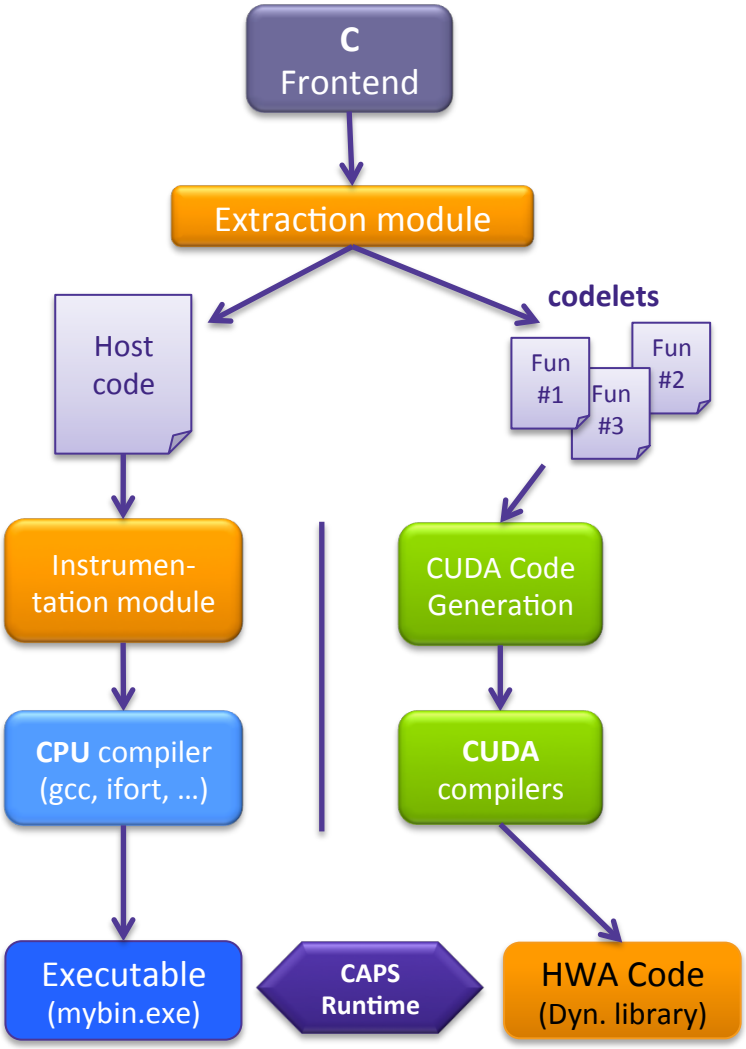
- *Kernels* regions implicitly define *Data* regions

CAPS OpenACC Compiler flow

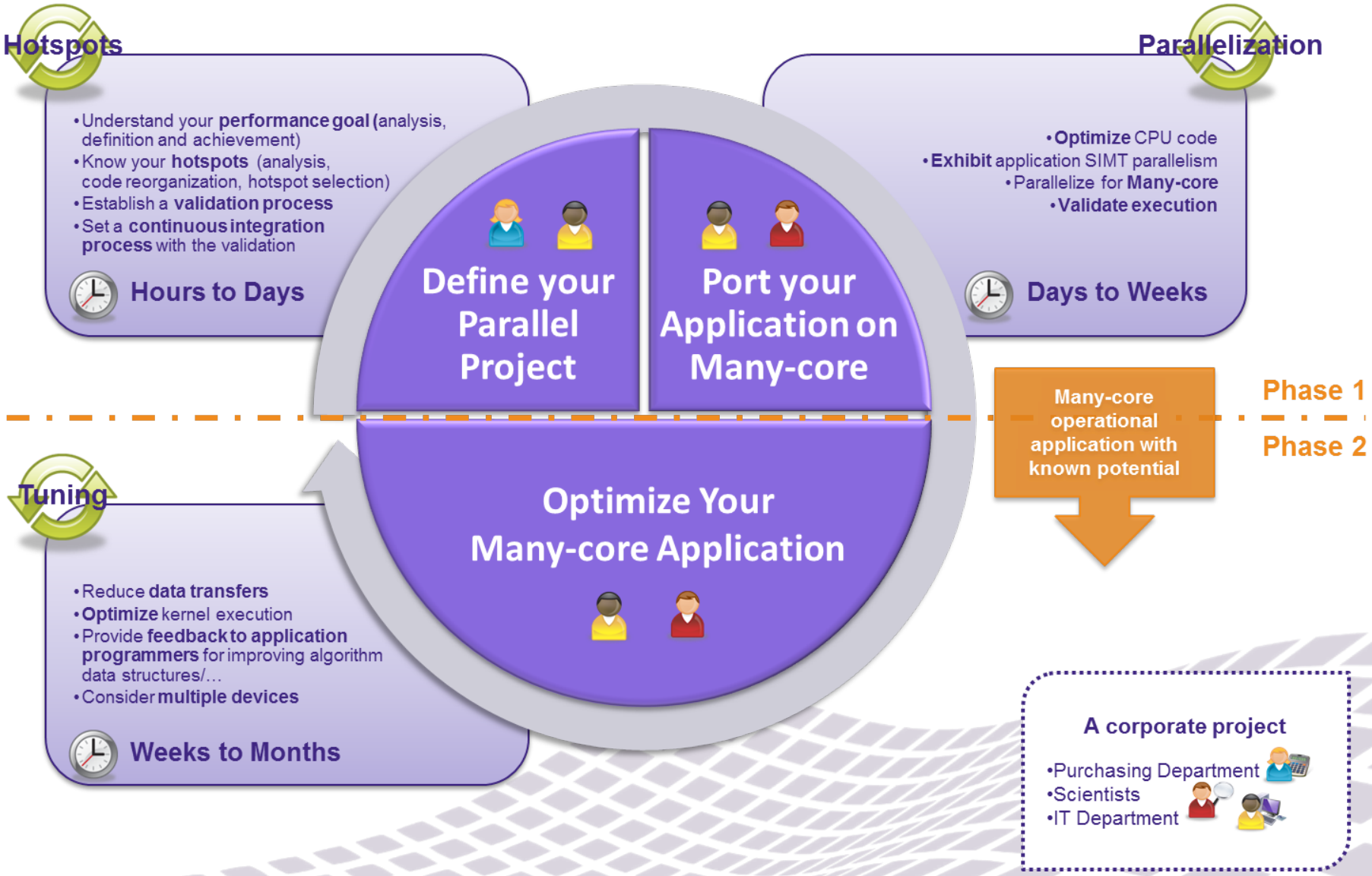


CAPS OpenACC Compiler flow for DNADist

```
$ hmpp gcc DNADist.c -o DNADist
```



HydroC Migration Steps



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**

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

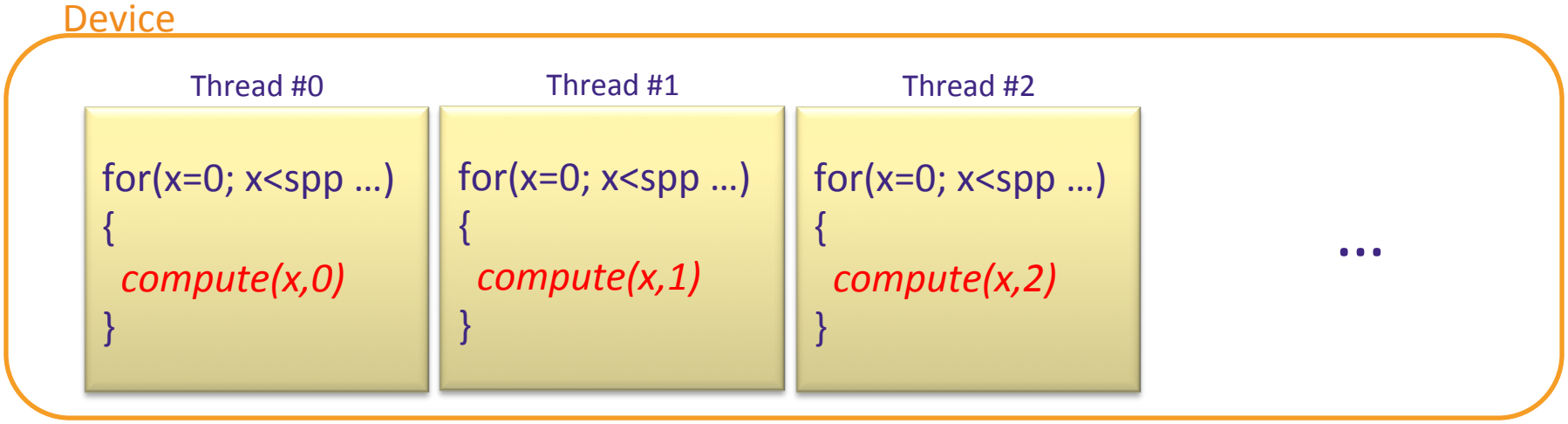
- Compilation with CAPS tools

```
$ 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

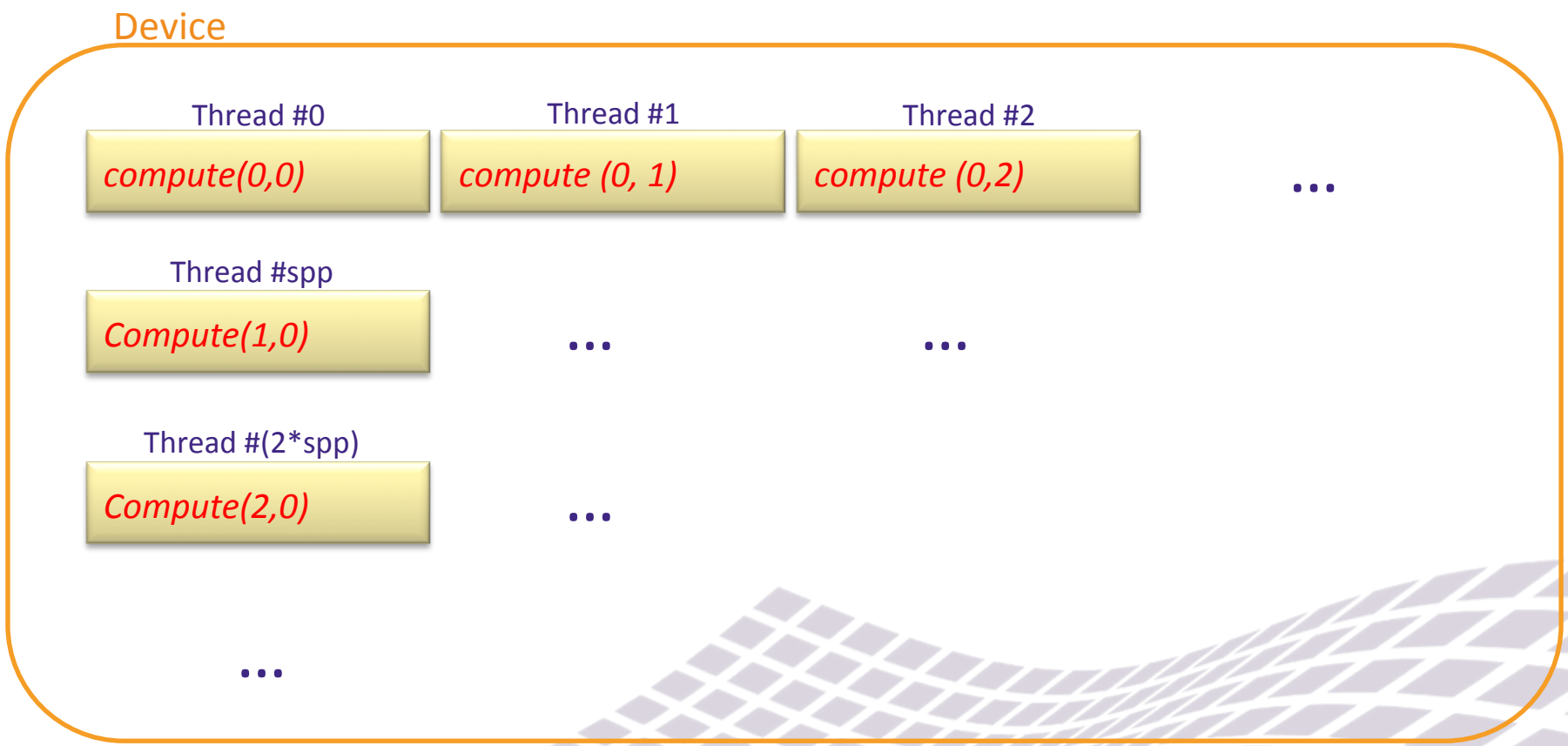
- What happened in the previous code generation



- 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

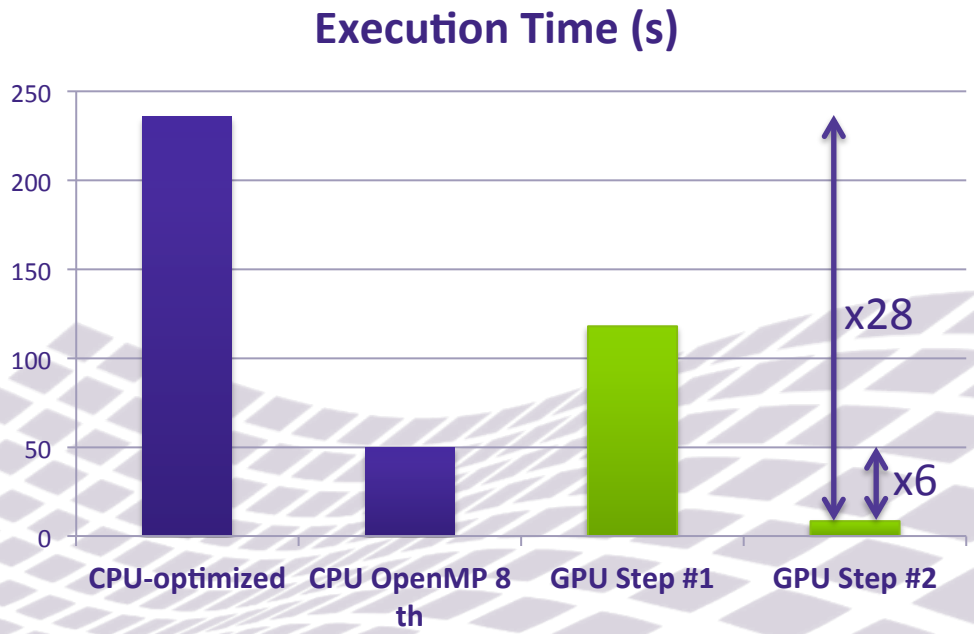
```
void makev_kernel ( int spp, int endsite,
    TYPE h_sitevalues[spp * endsite * 4],
    TYPE weightrat[endsite],
    ….
    TYPE freqcy, TYPE freqty, TYPE fracchange,
    TYPE h_vvs[spp * spp])
{
   #pragma acc kernels copy(h_vvs[0:spp * spp], h_sitevalues[0:spp * endsite * 4], weightrat[0:endsite])
   {
      #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?

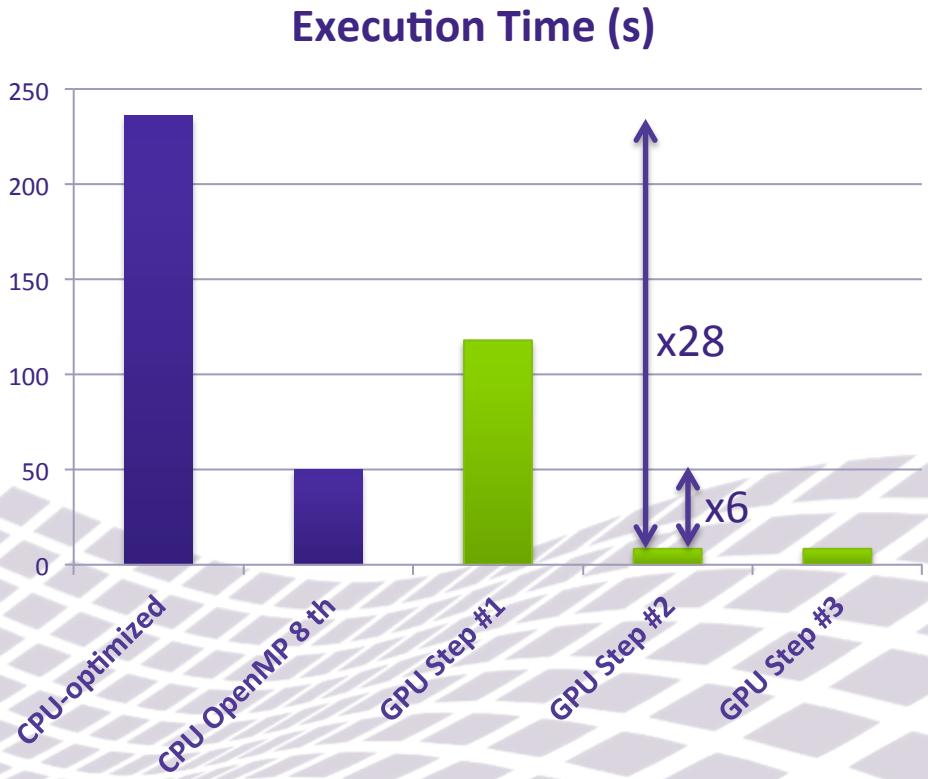
- Code execution feedback

```
$ ./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)
```

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.



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
  - 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/
  - $199 (or 199 €)

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 HMPP Workbench
  - ➔ www.caps-entreprise.com/technology/hmpp/

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

To see this webinar again:

http://www.caps-entreprise.com/webinars

webinars@caps-entreprise.com