Addressing Heterogeneity in Manycore Applications

RTM Simulation Use Case

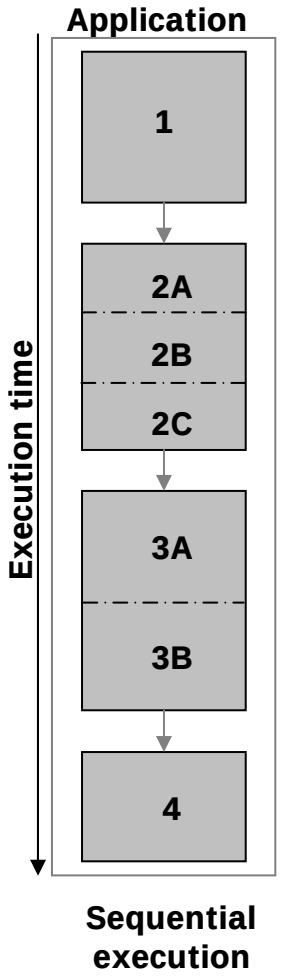
stephane.bihan@caps-entreprise.com

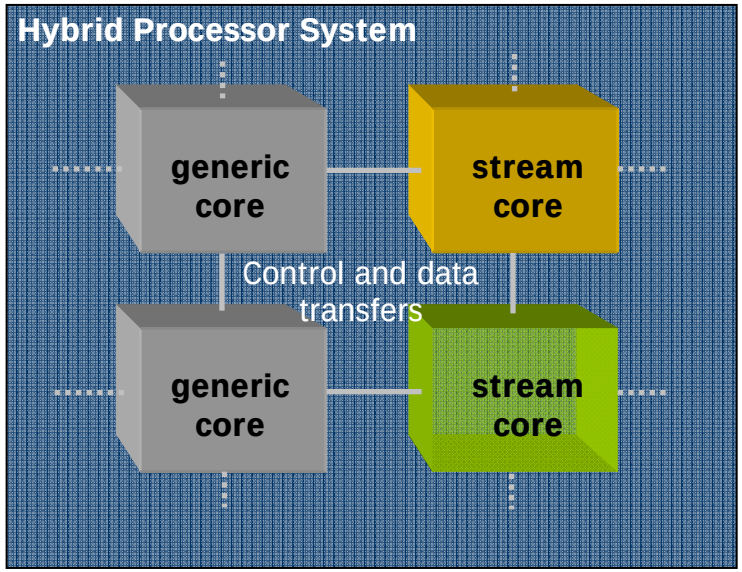
Oil&Gas HPC Workshop
Rice University, Houston, March 2008

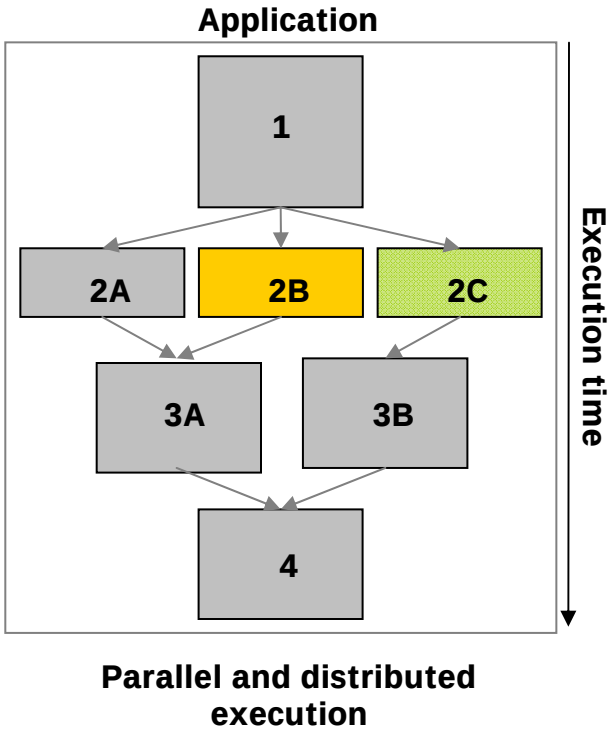
Introduction

- Hardware accelerators (HWAs) such as GPUs can bring very high application speedups but
  - No consensus on the programming model
    - Their usage brings complexity in the management of the application codes
  - No consensus on the architecture
    - Software moves slowly, hardware moves fastly
- There is a need for a *glue* between general purpose programming and HWA specific environments

Hybrid Application View

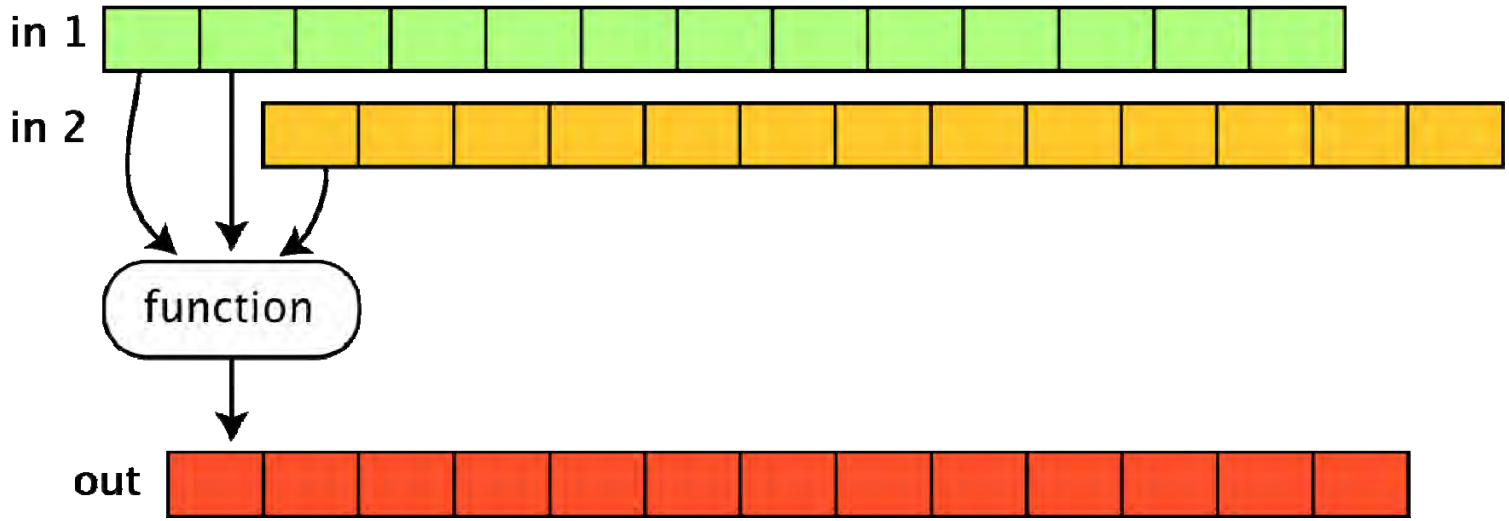






Stream Computing

- A similar computation is performed on a collection of data (*stream*)
  - There is no data dependence between the computation on different stream elements



Heterogeneous Platforms

- Homogeneous multi-core with already integrated specialized compute units (SSE, Cell)
- Use of external accelerators mainly interconnected with PCIe
  - NVIDIA® Tesla™
  - AMD-ATI GPUs
  - Various FPGAs: Celoxica, Mitrionics
  - ASICs: ClearSpeed
- Coming integrated hybrid systems
  - Intel Larrabee (QuickPath, PCIe)
  - AMD Fusion (HyperTransport™)

Existing Programming Tools

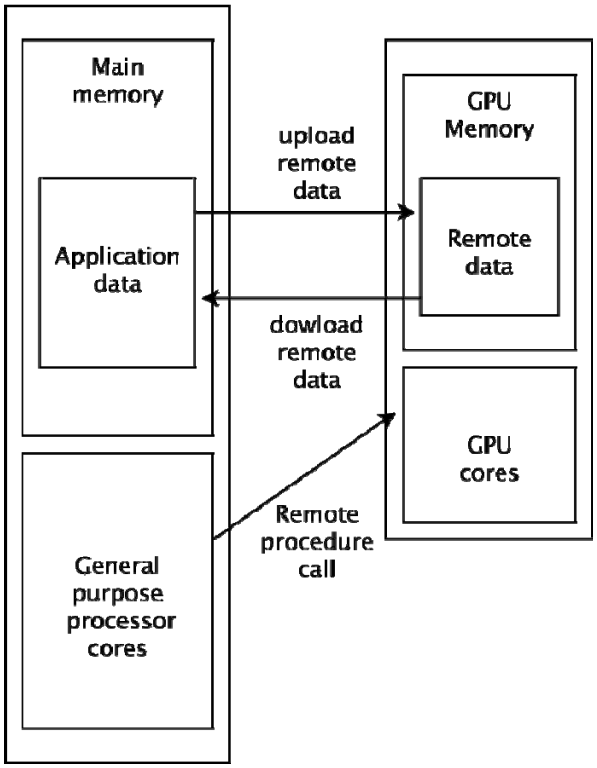
- Numerous HWA specific environments
  - NVIDIA ® CUDA™
  - AMD CAL (plus Brook, IL)
  - RapidMind
  - …
- No standard languages but extensions
- Address the programming of the stream cores
- Require new language or API training

HMPP™ Concepts

1. Expressing parallelism and communications in C and Fortran source
   - Through the use of directives “à la” OpenMP
   - The legacy code is preserved to ensure portability and default compilation and execution
2. Dealing with resource availability and scheduling
   - Scale to various platform configurations
   - Application should still run even if a core is not available or locked
3. Programming the HWAs
   - Insulate HWA specific computations
   - Use hardware vendor SDK
   - Domain specific code generators

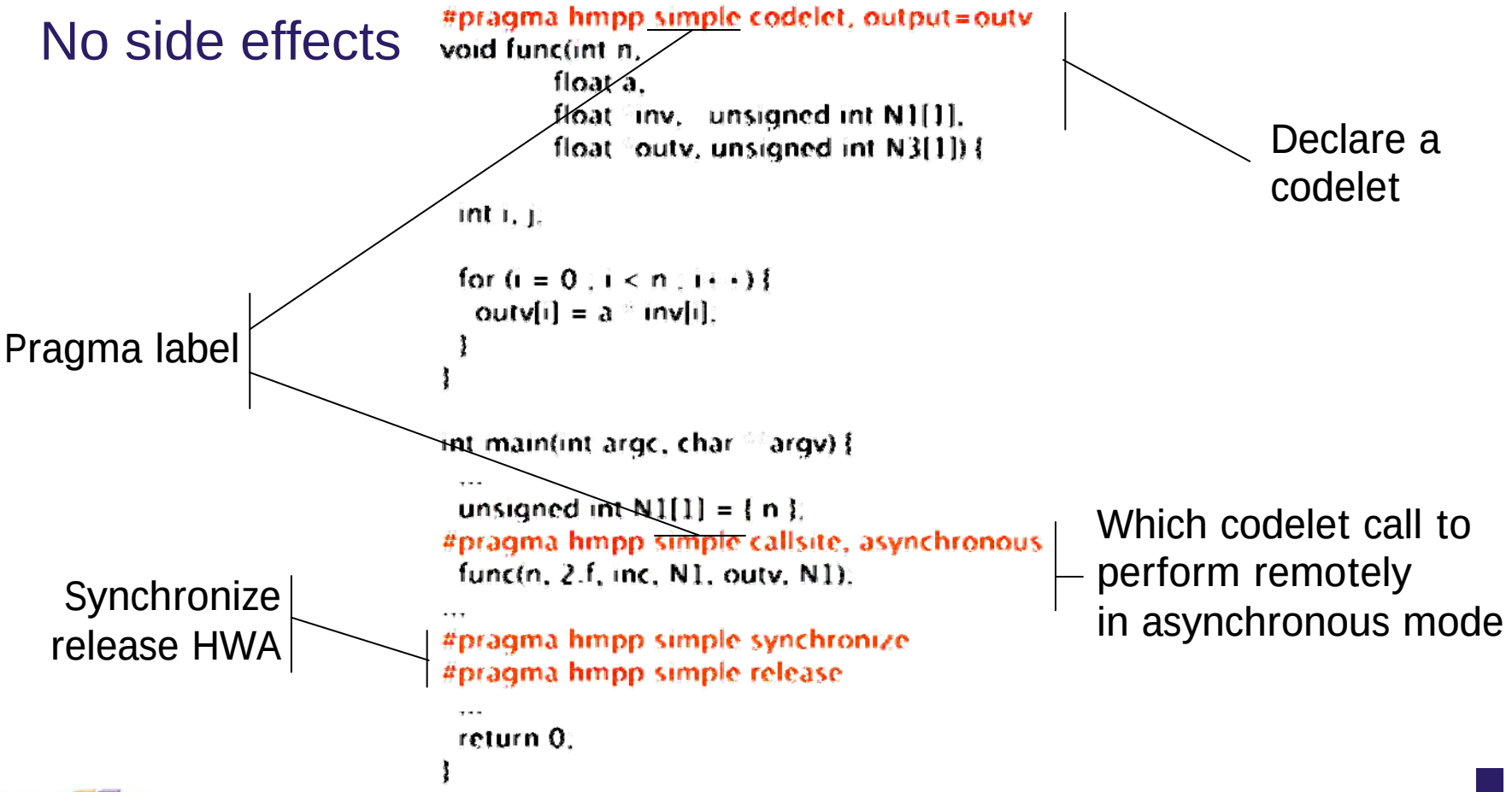
HMPP™ Directives

- Define hardware specific implementations of functions (codelets)
  - Can be specialized to the execution context (data size, …)
- Codelet execution
  - Synchronous, asynchronous properties
- Express data transfers
  - Pre loading of data
- Insert synchronization barriers
  - Host CPU will wait until remote computation is complete

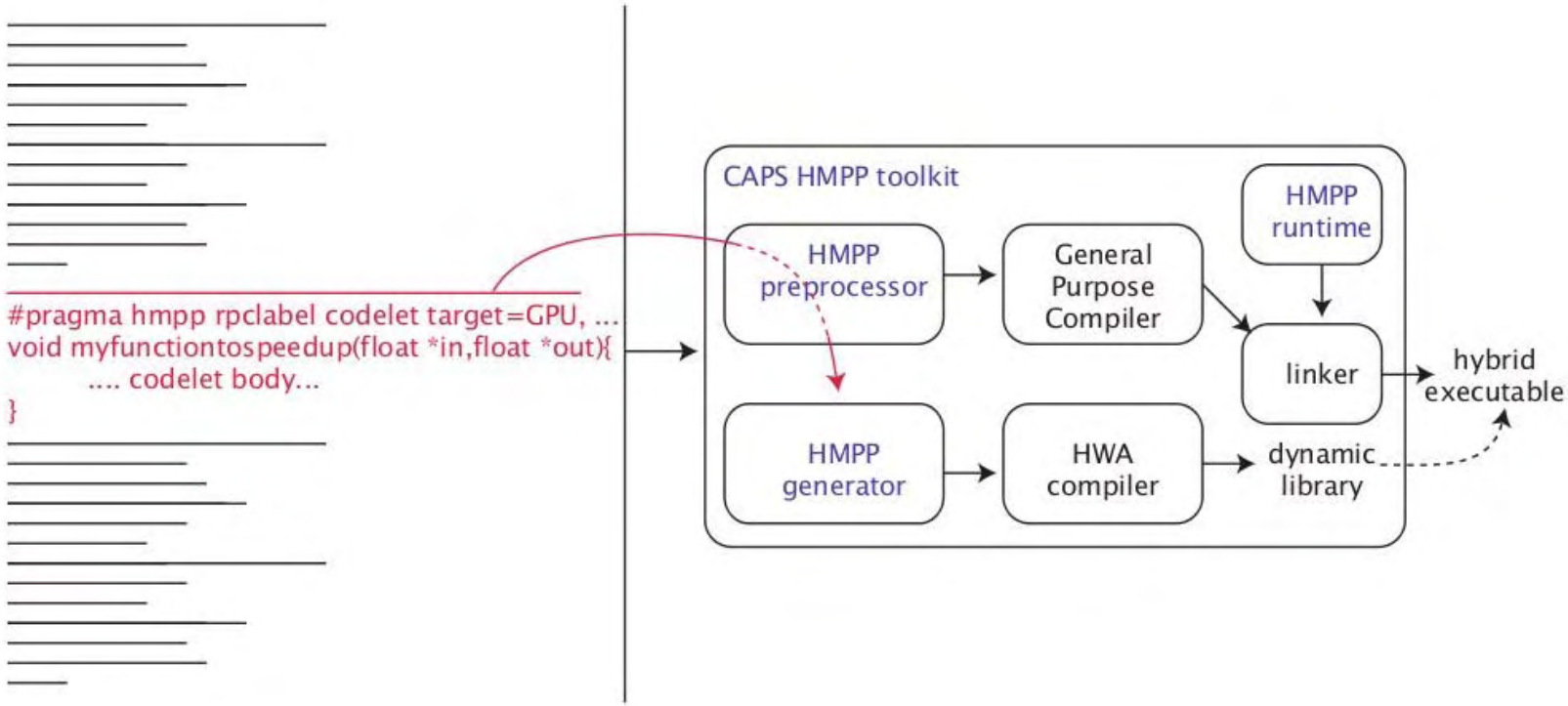


HMPP™ Codelet

- Pure function
- No side effects



HMPP™ Code Generation Flow



Codelet Execution Guard

- Codelet implementation may have restrictions
  - Performance
  - Data size, …

Guard to restrict HWA codelet usage

```
#pragma hmpp simple codelet, output=outv, &
#pragma hmpp simple param n = 1024, &
[#pragma hmpp simple cond = expr("n == 1024"), &]
#pragma hmpp simple target GPU
void func(int n,
          float a,
          float *inv,   unsigned int N1[1],
          float *outv, unsigned int N3[1]) {

  int i, j;

  for (i = 0 ; i < n ; i++) {
    outv[i] = a * inv[i];
  }
}

int main(int argc, char **argv) {
  ...
  unsigned int N1[1] = { n };
#pragma hmpp simple callsite
  func(n, 2.f, inc, N1, outv, N1);
  ...
  return 0;
}
```

Communication Directives

- Data transfers strongly impact performance
  - Data prefetching
  - Avoid reloading data

```
#pragma hmpp simple codelet, output=outv
void func(int n,
          float a,
          float *inv,   unsigned int N1[1],
          float *outv, unsigned int N3[1]) {

  int i, j;

  for (i = 0 ; i < n ; i++) {
    outv[i] = a * inv[i];
  }
}

int main(int argc, char **argv) {
  ...
  inc[...] = ...;
#pragma hmpp simple advancedload, calleeArg=inv, &
#pragma hmpp simple const, asynchronous
  ...
  unsigned int N1[1] = { n };
#pragma hmpp simple callsite, advancedload, calleeArg=inv
  func(n, 2.f, inc, N1, outv, N1);
  ...
  return 0;
}
```

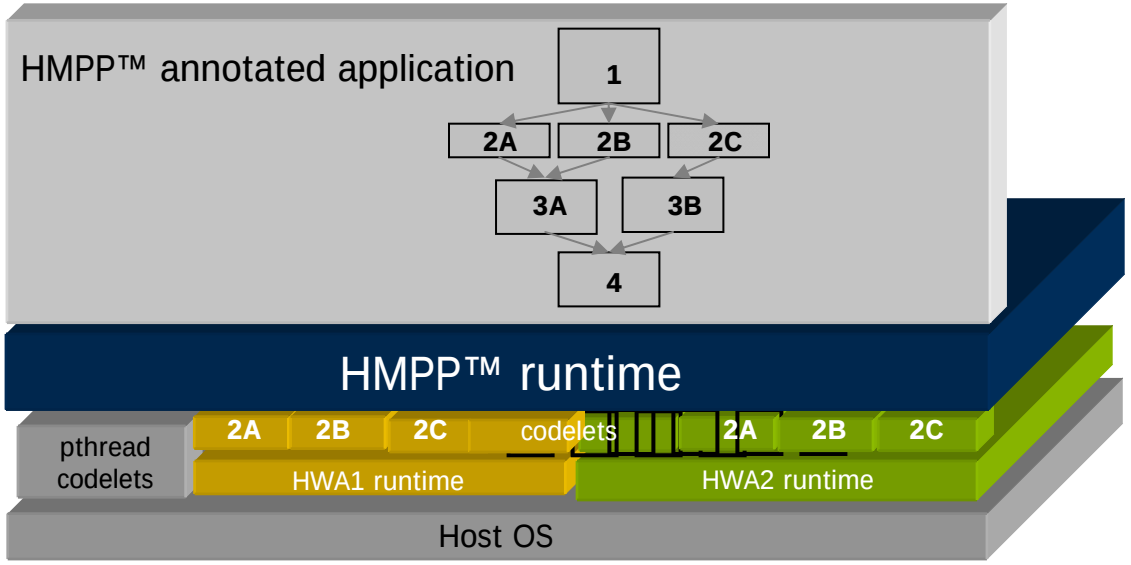
Where to prefetch

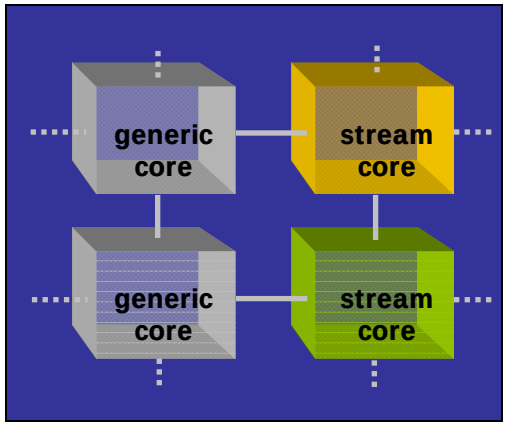
Argument is prefetched

Codelet Code Generation

- Stream computing specific approach
  - C and Fortran inputs
  - Code parallelisation machine independent directives
  - CUDA, SSE, CAL (soon) target languages
- Intend to cover most frequent cases
- Generate all runtime support routines
  - Communication, …

HMPP™ Application View



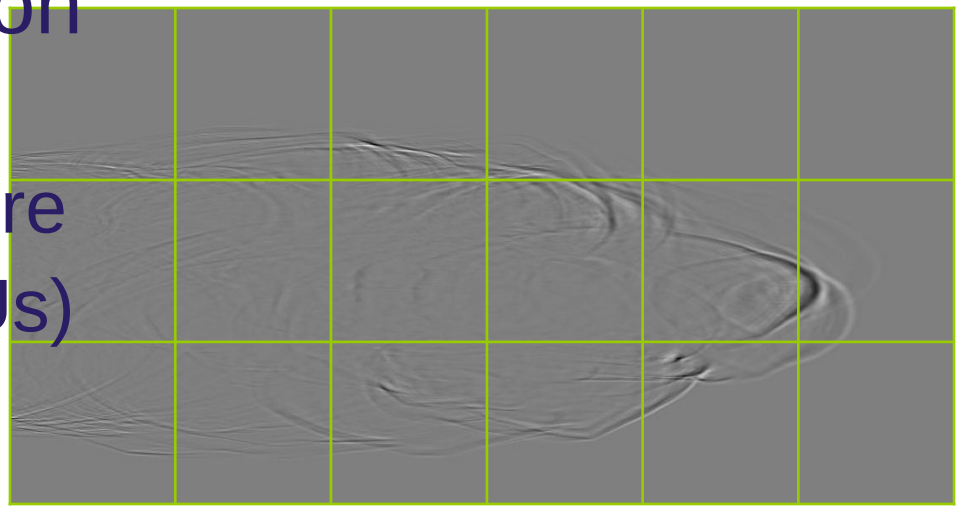


HMPP™ Key Points

- Not disruptive: keep the application code portable
- A glue between general purpose programming and hardware specific environments
- No new programming language or model: a standard way « à la » OpenMP
- Leverage the hardware vendor tools
- Application scaling to various system configurations
- Provide hardware interoperability

Use Example: RTM

- Parallelized with MPI
- Each sub-domain processed by a GPU
  - Modeling / Migration
  - Only border nodes are transfered
- Cluster configuration
  - 5 nodes
  - Bi-socket quadricore
  - Tesla S870 (4GPUs) PCIe gen2



Current Status

- For a simple 2D test case using snapshoting on a workstation with Quadro FX5600
  - 8x for the computational parts
  - About 1/3 of the application time is disk I/O
- Key optimizations
  - CUDA kernels carefully tuned
    - Data alignment, coalescing
  - Data transfers must be asynchronously overlapped
- Next work
  - 3D case using 2D operations

Conclusion

- Hybrid computing offers cheap high performance per Watt
- Code portability and resource management are main issues
- HMPP approach is a first step toward a complete heterogeneous programming workbench