Workshop on Models, Algorithms and Methodologies for Hierarchical Parallelism in new HPC Systems

Virtualizing CUDA enabled GPGPUs on ARM clusters

R. Montella[1], G. Giunta[1], G. Laccetti[2], M. Lapegna[2]

C. Ferraro[1], C. Palmieri[1], V. Pelliccia[1]

[1]Department of Science and Technologies
University of Napoli Parthenope
http://hpsc.uniparthenope.it

[2]Department of Mathematics and Applications
University of Napoli Federico II
http://dma.unina.it

{ raffaele.montella, giulio.giunta, carmine.ferraro, carlo.palmieri, valentina.pelliccia }
@uniparthenope.it

{ giuliano.laccetti, marco.lapegna }
@unina.it

Generic Virtualization Service

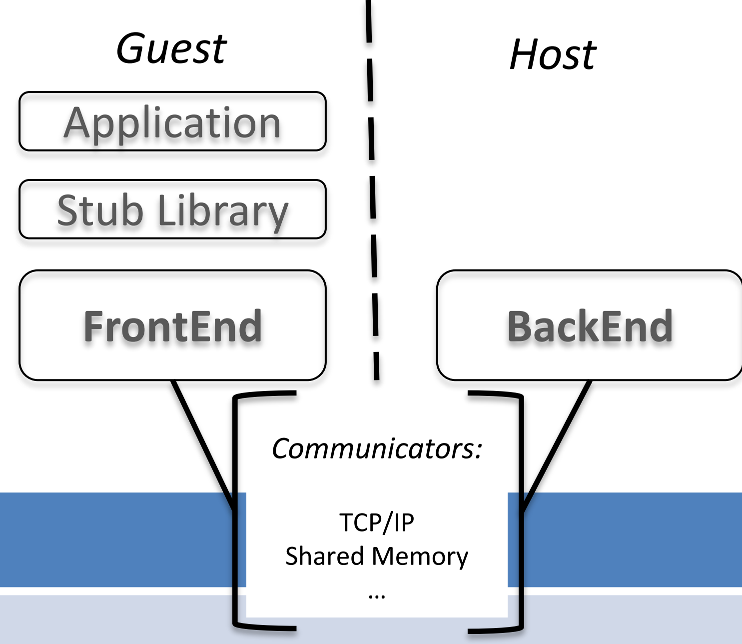
(since March 2010)

- Framework for split-driver based abstraction components
- Plug-in architecture
- **Independent from:**
  - Hypervisor (or no-hypervisor)
  - Communication
  - Target of virtualization
  - **Architecture!**
- **High performance:**
  - Enabling transparent virtualization
  - With overall performances better or not too far from un-virtualized resources

GVirtuS

http://hpsc.uniparthenope.it/

The Communicator



- Provides a high performance communication between virtual machines and their hosts.
- The choice of the hypervisor deeply affects the efficiency of the communication.

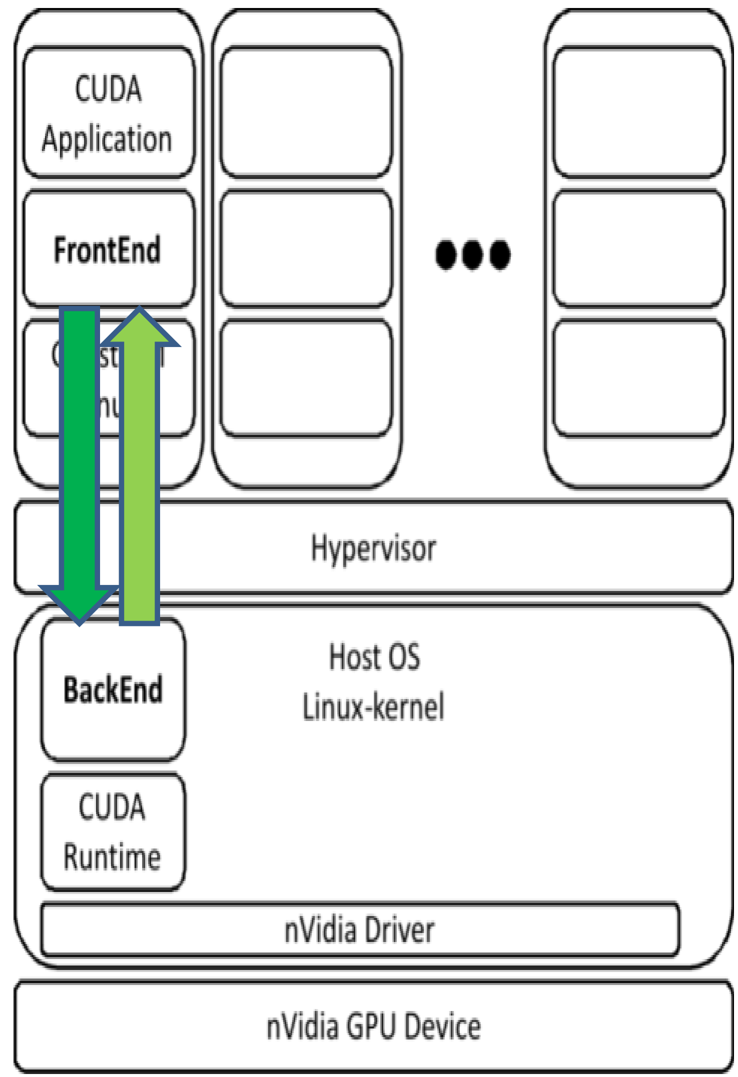
Hypervisor	FE/BE comm	Notes
No hypervisor	Unix Sockets	Used for testing purposes
Generic	TCP/IP	• Communication testing purposes • **Remote / Distributed virtualized resources (i.e. GPUs)** • **High Performance Internet of Things**
Xen	XenLoop	• runs directly on the top of the hardware through a custom Linux kernel • provides a communication library between guest and host machines • implements low latency and wide bandwidth TCP/IP and UDP connections • app transparent and offers an automatic discovery of the supported VMs
VMware	Virtual Machine Communication Interface (VMCI)	• commercial hypervisor running at the application level • provides a datagram API to exchange small messages • a shared memory API to share data • an access control API to control which resources a virtual machine can access • and a discovery service for publishing and retrieving resources

An application: Virtualizing gpGPUs

- **GPUs**
  - Hypervisor independent
  - Communicator independent
  - **GPU independent**
  - **Programming model independent**

GVirtuS – pre OpenCL

- Full support to
  - CUDA 3.x drivers
  - CUDA 3.x runtime
- Partially supporting (more work is needed)
  - OpenGL integration
- **Limitations:**
  No support after CUDA 3.x because NVIDIA issues

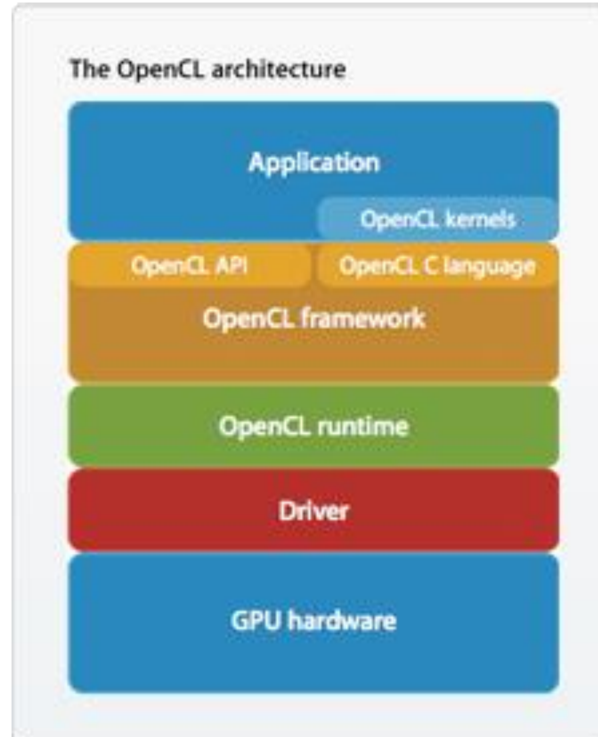


GVirtuS - OpenCL

- Currently ongoing!

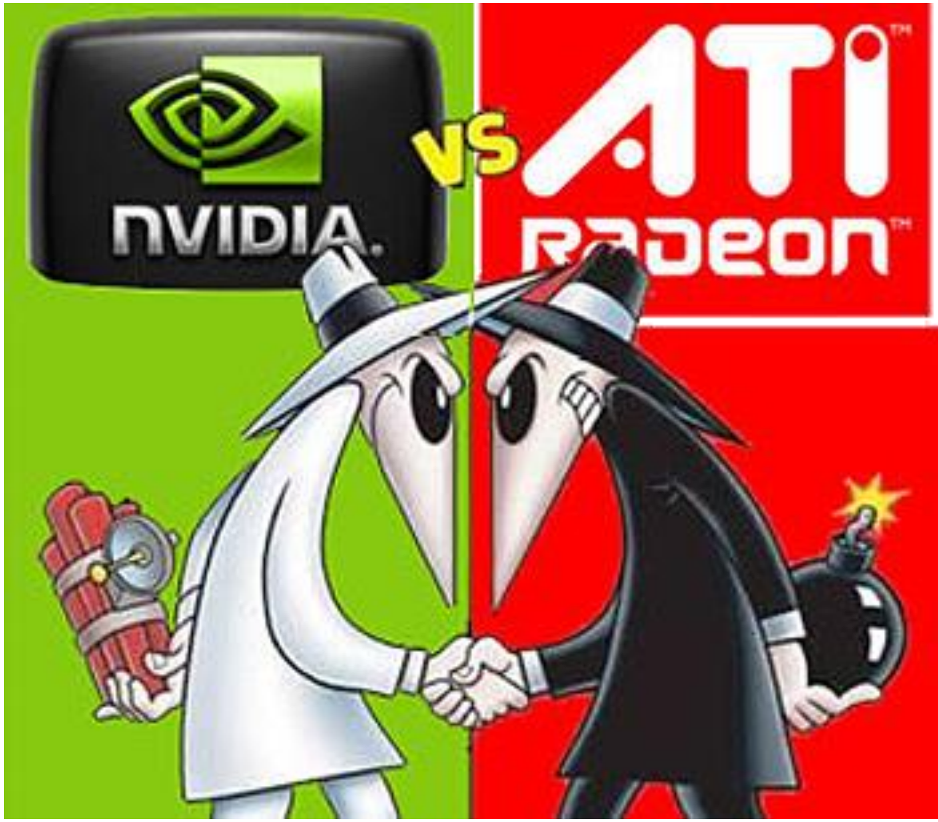
- Benchmarks available:
  - Matrix Multiply
  - Vector Product
  - Histogram

- Why OpenCL instead of CUDA:
  - Open Platform
  - Hardware independent (CPUs, GPUs)
  - Widely supported

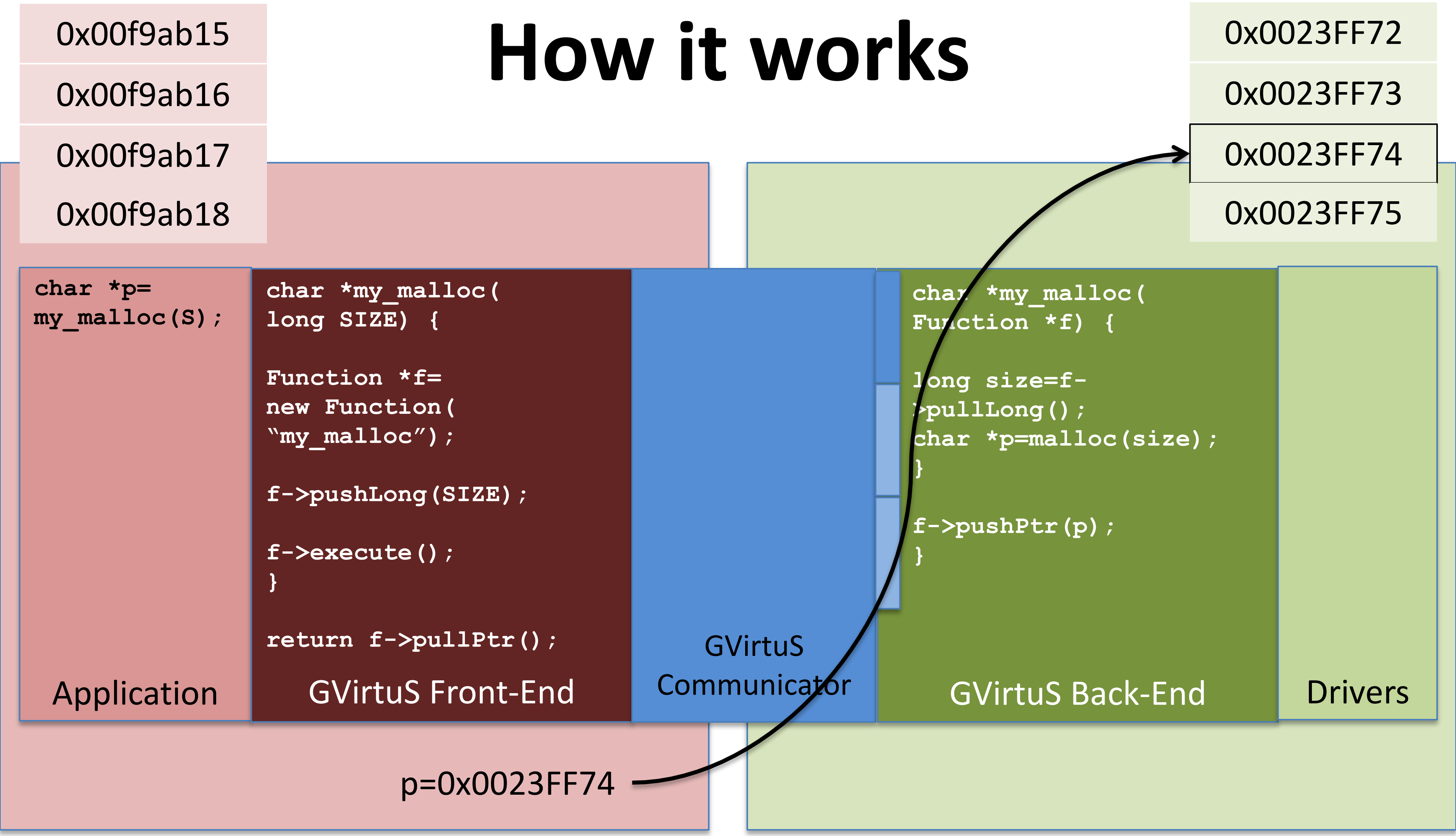


But the world evolved differently…

- OpenCL didn’t behaved as expected…
- …people uses CUDA…
- …we have to be back to the CUDA plugin.

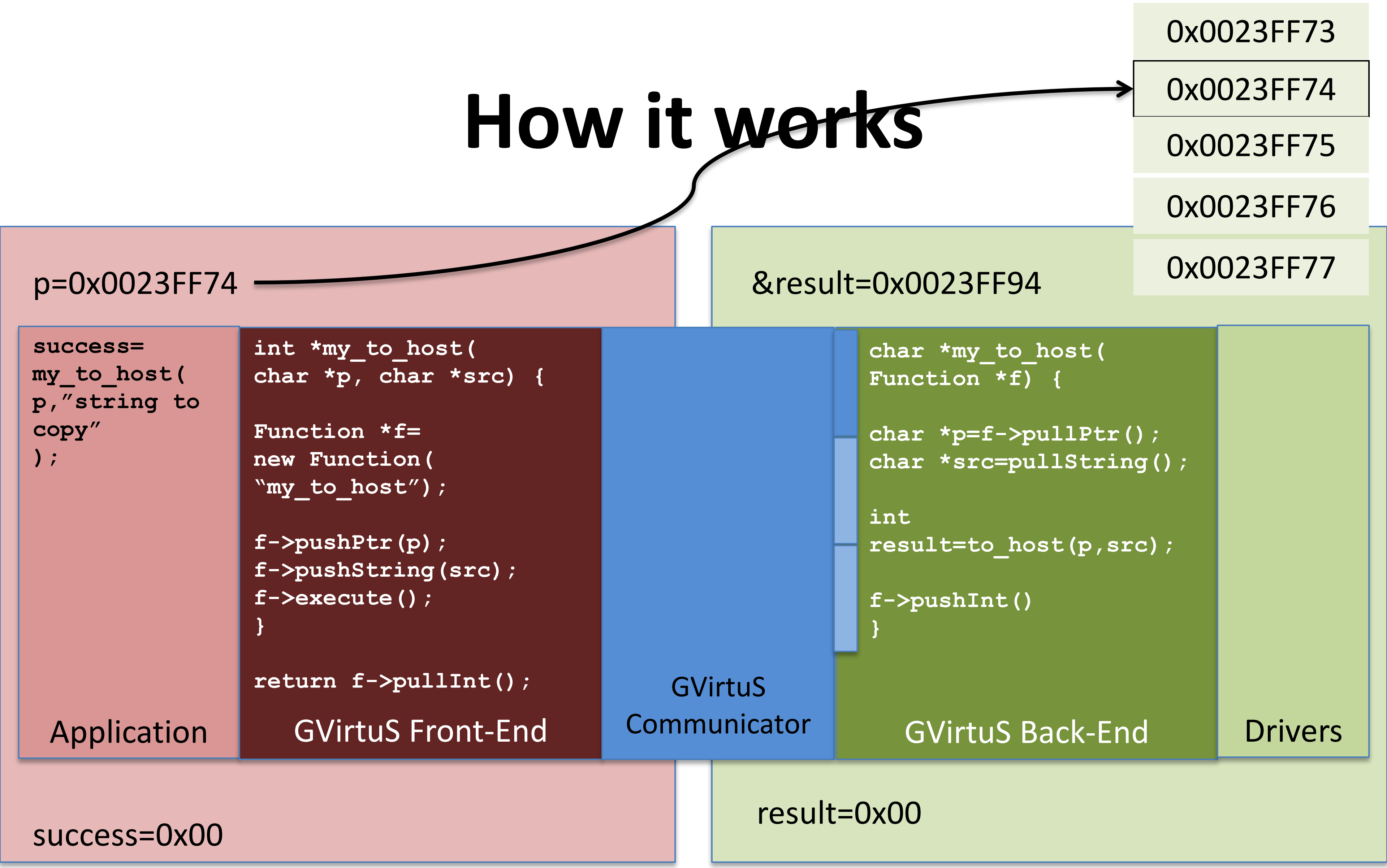


How it works



- Simplified example: allocate memory on the host

How it works



- Simplified example: accessing host memory

~~memcpy(p, "string to copy", 15)~~

GVirtuS and CUDA!

Consumer – The CUDA app

```
#include <stdio.h>
#include <cuda.h>
int main(void) {
    int n;
    cudaGetDeviceCount(&n);
    printf("Number    CUDA GPU(s): %d\n", n);
    return 0;
}
```

GVirtuS Frontend

```
cudaError_t cudaGetDeviceCount(int *count) {
    Frontend *f = Frontend::GetFrontend();
    f->AddHostPointerForArguments(count);
    f->Execute("cudaGetDeviceCount");
    if(f->Success())
        *count =
          *(f->GetOutputHostPointer<int>());
    return f->GetExitCode();
}
```

Producer – The CUDA device host

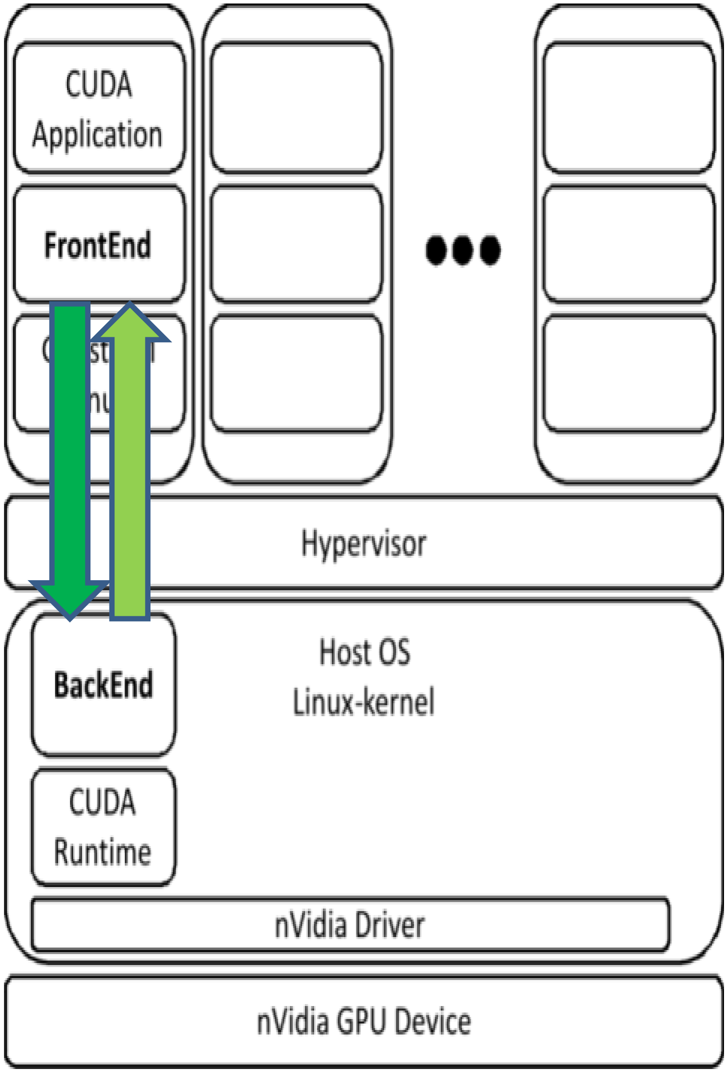
GVirtuS Backend

Process Handler

```
Result *handleGetDeviceCount(
  CudaRtHandler * pThis,
  Buffer *input_buffer) {
    int *count = input_buffer->Assign<int>();
    cudaError_t exit_code;
    exit_code = cudaGetDeviceCount(count);
    Buffer *out = new Buffer();
    out->Add(count);
    return new Result(exit_code, out);
}
```

GVirtuS – post OpenCL

- On going support to
  - CUDA 6.x drivers
  - CUDA 6.x runtime
- Partially supporting (more work is needed)
  - OpenGL integration
- **Limitations:**
  For now many, but challenging!



Scenarios and prototypal applications

- Development workstation/1
  - i7-940 2*NVIDIA Tesla C1060

- Development workstation/2
  - i7-940 2*NVIDIA Titan X

- The AWS EC2 GPU machine
  - g2.2xlarge 1*NVIDIA K250
- ARM based computing node/1
  - Udoo Cortex A9 32bit - nogpu

- ARM based computing node/2
  - NVIDTA Jetson Cortex A15 64bit – gpu/nogpu

Development Workstation

asyncAPI	WORKS
cdpSimplePrint	WORKS
cdpSimpleQuicksort	requires GPU devices with compute SM 3.5 or higher
clock	WORKS
cppIntegration	WORKS
cppOverload	requires GPU devices with compute SM 2.0 or higher
cudaOpenMP	ERRORS
inlinePTX	WORKS
matrixMul	WORKS
matrixMulCUBLAS	Comparing CUBLAS Matrix Multiply with CPU results: FAIL
matrixMulDrv	WORKS
simpleAssert	requires a GPU with compute capability 2.0 or later
simpleAtomicIntrinsics	WORKS
simpleCallback	ERRORS
simpleCubemapTexture	requires SM 2.0 or higher
simpleIPC	ERRORS
simpleLayeredTexture	simpleLayeredTexture requires SM >= 2.0
simpleMPI	WORKS

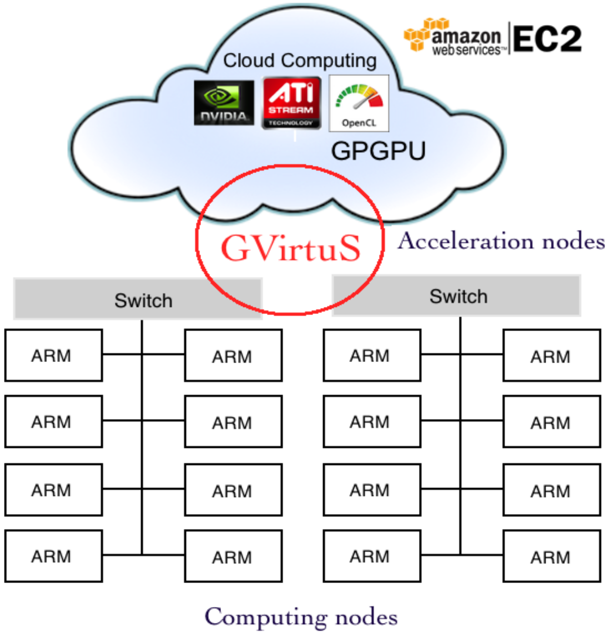
simpleMultiCopy	WORKS
simpleMultiGPU	WORKS
simpleOccupancy	cudaOccupancyMaxActiveBlocksPerMultiprocessor
simpleP2P	Two or more GPUs with SM 2.0 or higher capability are required
simplePitchLinearTexture	*** Error: cudaMemcpy2D() not yet implemented!
simplePrintf	WORKS
simpleSeparateCompilation	requires a GPU with compute capability 2.0
simpleStreams	WORKS
simpleSurfaceWrite	__cudaRegisterSurface
simpleTemplates	WORKS
simpleTexture	simpleTexture completed, returned ERROR!
simpleTextureDrv	WORKS
simpleVoteIntrinsics	WORKS
simpleZeroCopy	Device 0 does not support mapping CPU host memory!
template	WORKS
template_runtime	WORKS
UnifiedMemoryStreams	requires Compute Capability of SM 3.0
vectorAdd	WORKS
vectorAddDrv	WORKS

- Current GVirtuS CUDA 6.5 results running 0_Simple SDK examples.

Test	With GVirtuS	Without GVirtuS
Matrix Multiplication	0.139 sec	0.092 sec
Vector Addition	0.063 sec	0.059 sec
Sorting Networks	8.787 sec	8.539 sec

- Backend and Frontend on x86_64 (TcpIp)

AWS EC2 K250 GPU



Test	With GVirtuS	Without GVirtuS
Matrix Multiplication	38.236 sec	0.098 sec
Vector Addition	3.298 sec	0.057 sec
Sorting Networks	2min24.648 sec	8.482 sec

Backend: EC2 K250
Frontend: X86_64

Test	With GVirtuS	Without GVirtuS
Matrix Multiplication	50.607 sec	N/A (Can't run)
Vector Addition	3.368 sec	N/A (Can't run)
Sorting Networks	4min17.547 sec	N/A (Can't run)

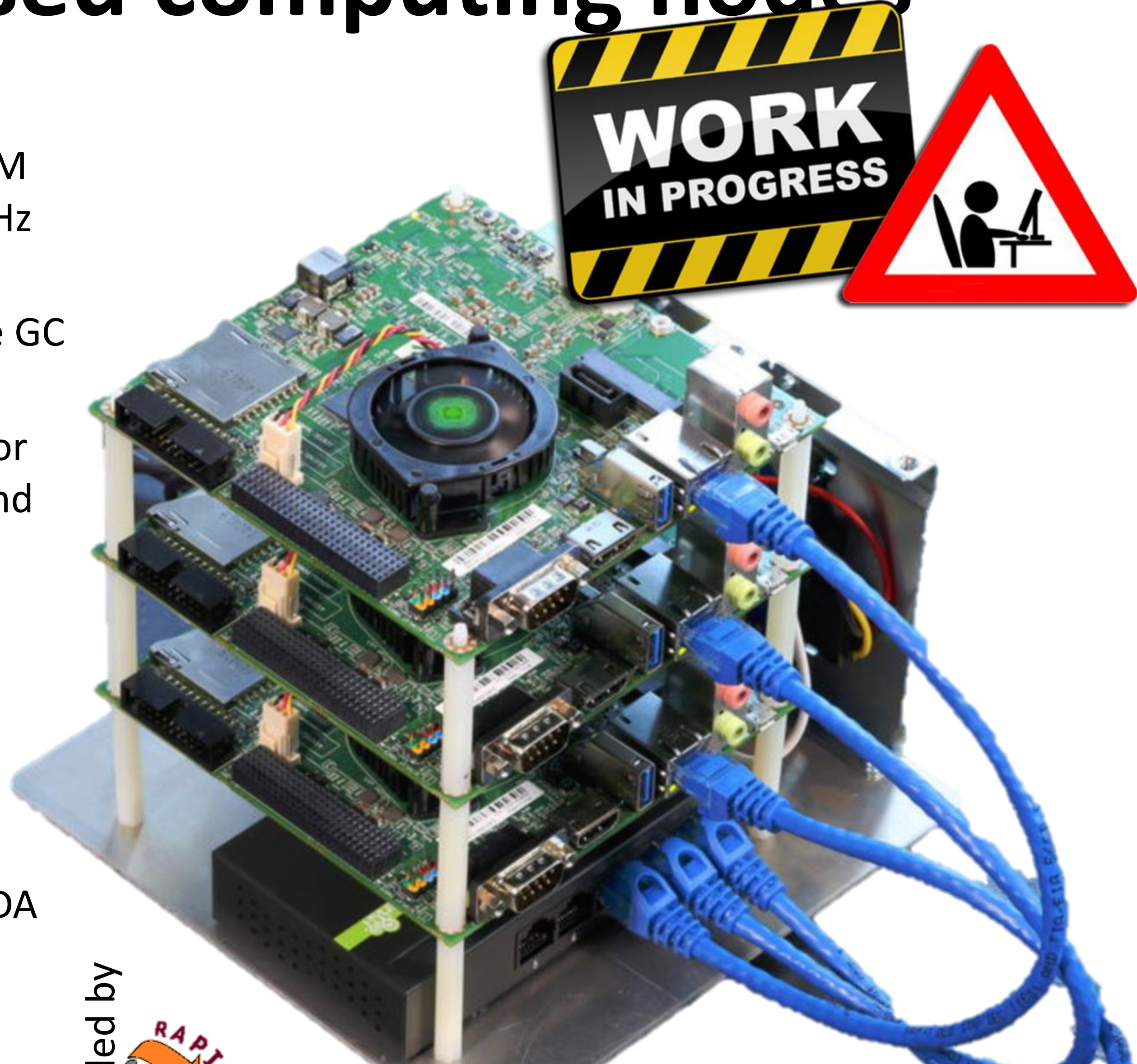
Backend: EC2 K250
Frontend: ARM Cortex A9

- Elastic GPU sharing.

ARM based computing nodes

- UDOO
  - CPU Freescale i.MX 6 ARM Cortex-A9 Quad core 1GHz
  - GPU Vivante GC 2000 + Vivante GC 355 + Vivante GC 320
  - Integrated accelerators for 2D, OpenGL® ES2.0 3D and OpenVG™
  - RAM DDR3 1GB

- NVIDIA Jetson
  - Tegra K1 SOC
  - Kepler GPU with 192 CUDA cores
  - 4-Plus-1 quad-core ARM Cortex A15 CPU
  - 2 GB x16 memory 64 bit



Funded by
RAPID
Heterogeneous Secure Multi-level Remote Acceleration Service for Low-Power Integrated Systems and Devices

A real life application:

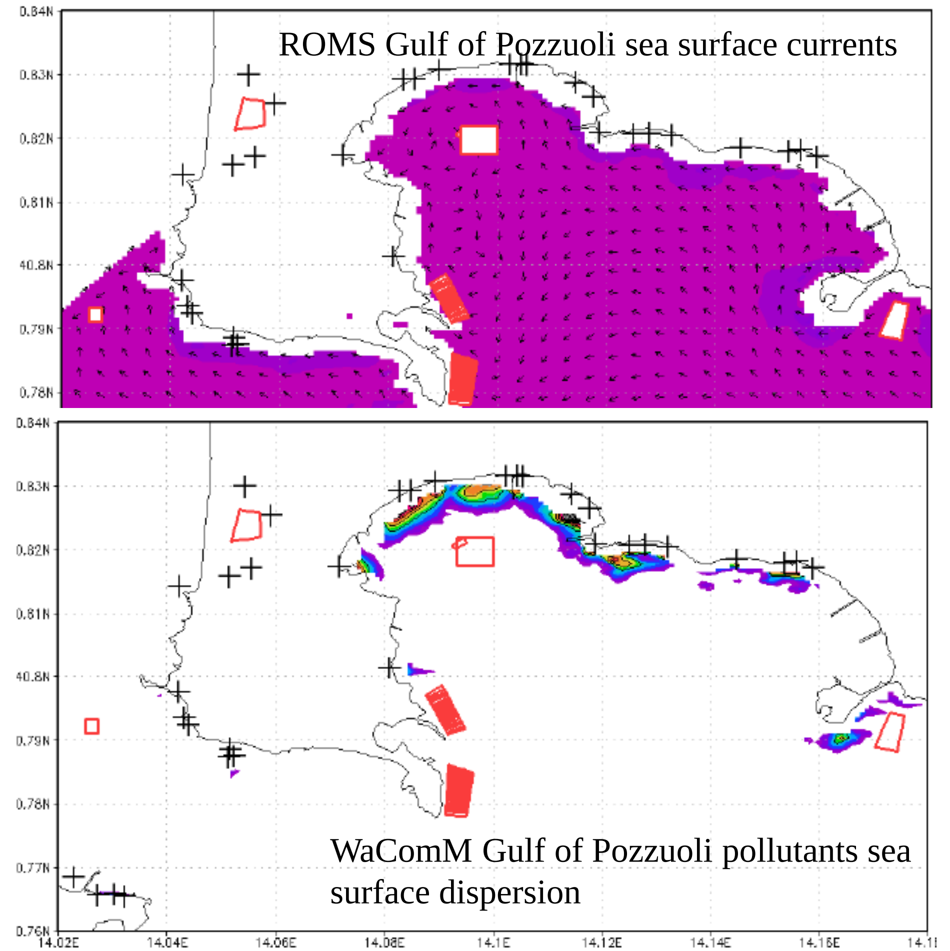
Coupling EulerianLagrangian models for offshore and coastal pollution tracing

WaComM (*Water Community Model*)

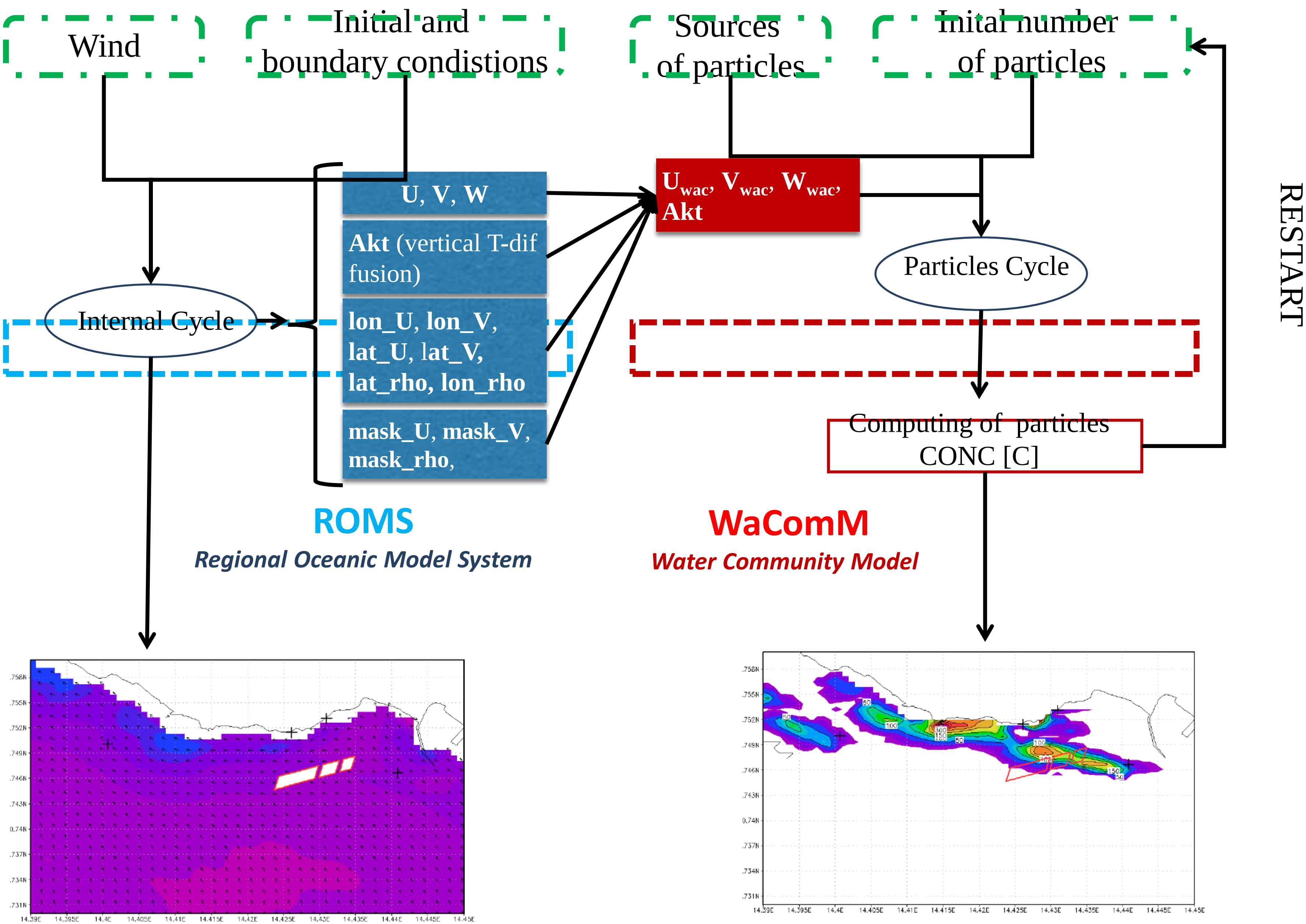
- Lagrangian 3D model
- simulate the transport and dispersion of pollutants spilled out into offshore ocean currents.

- Evolution of the *Lagrangian Assessment for Marine Pollution 3D* (LAMP3D) model.
- Restarts
- Shared memory / CUDA Hybrid parallel

- Hybrid approach (Eulerian-Lagrangian models)
- Coastal dynamics of the Bay of Naples
- Forecast the impact of pollutants spilled out from both natural and anthropic sources in high density areas of mussel culture.



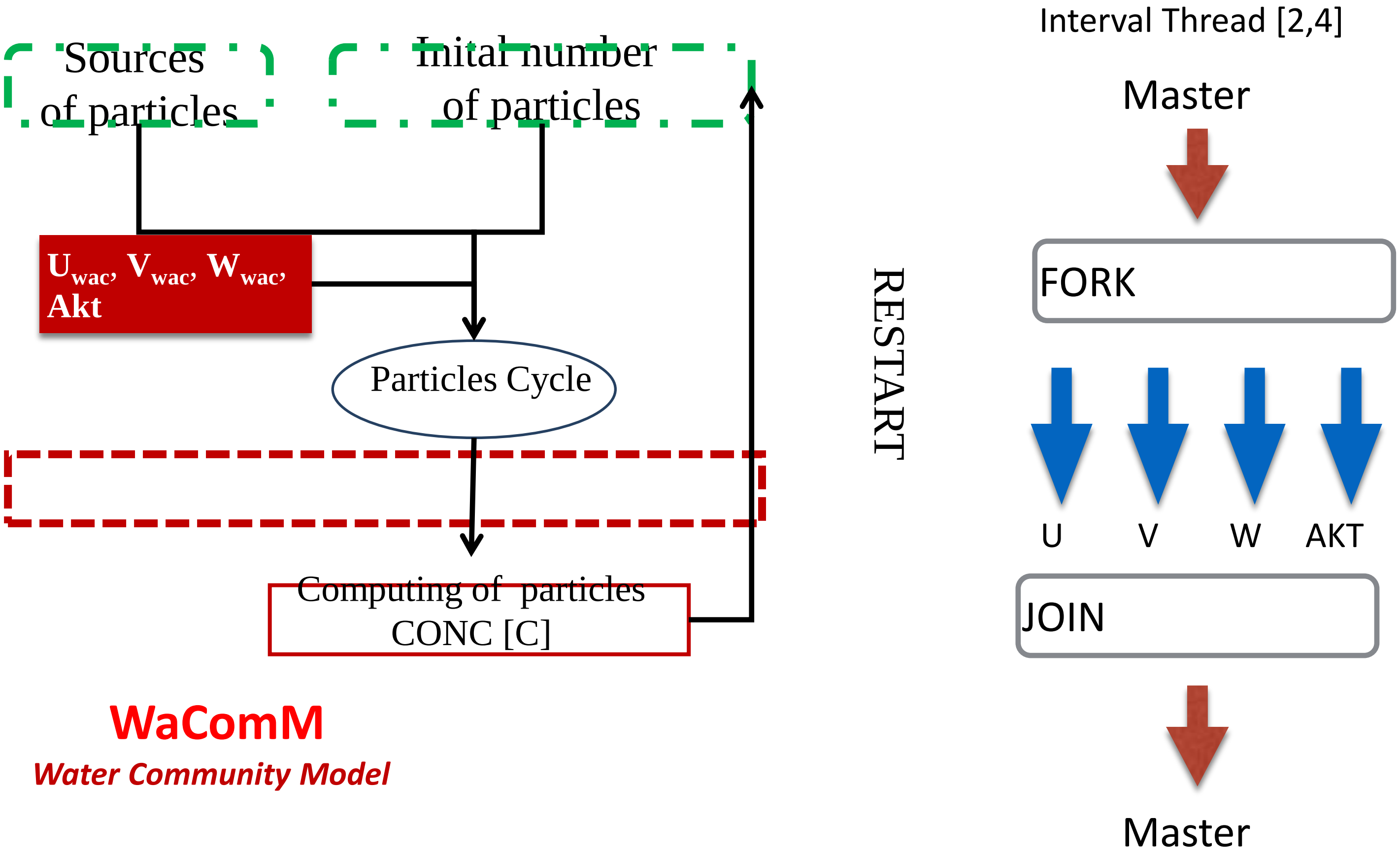
D. Di Luccio, A. Riccio, P. Troiano and R. Montella
University of Napoli Parthenope



ROMS Torre del Greco (NA) sea surface currents

WaComM Torre del Greco (NA) pollutants sea surface dispersion

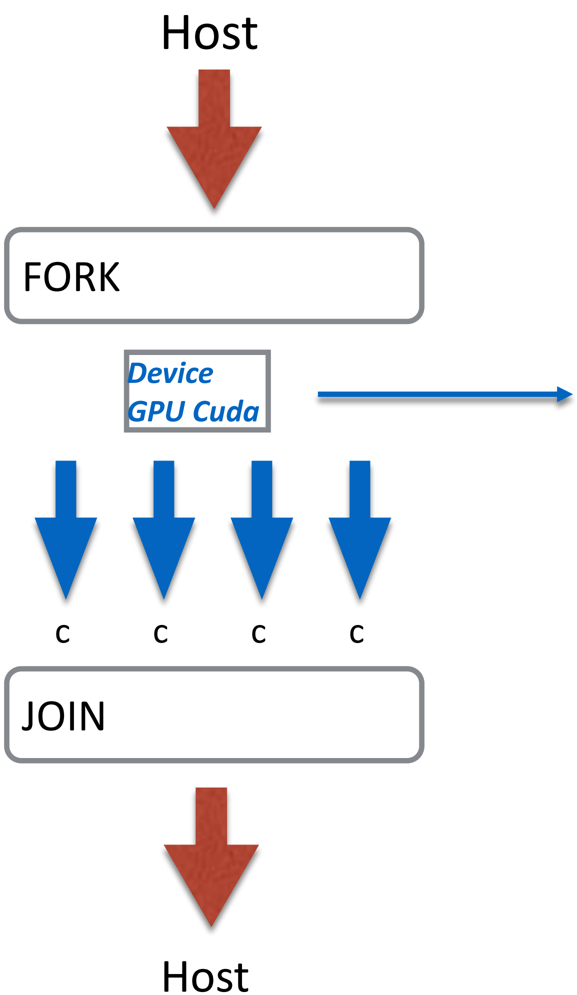
Parallel WaComM

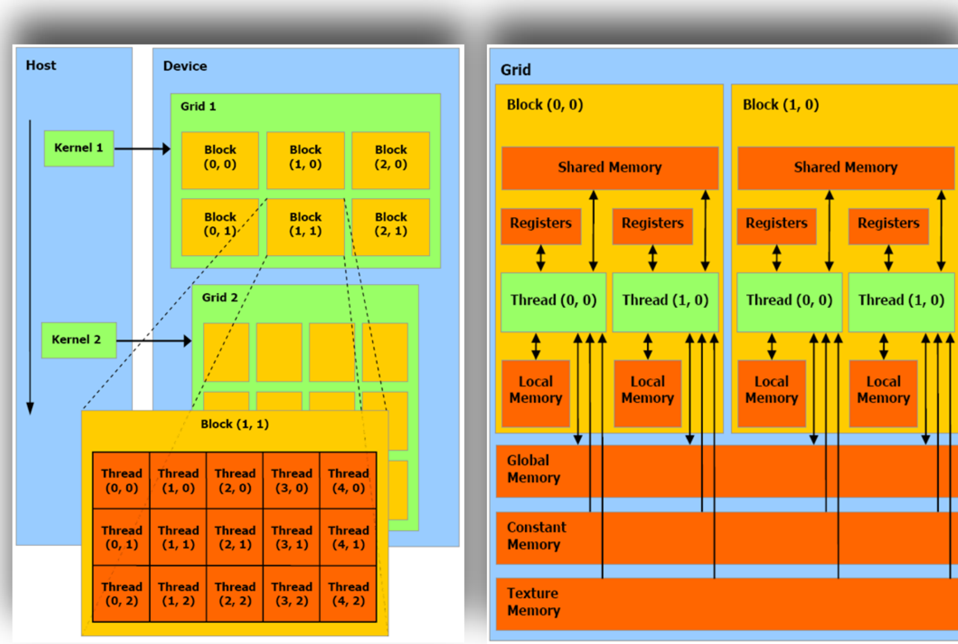


Parallel computations on data structures ***u v w*** and ***akt***. The results obtained will be managed for the calculation of the variable ***conc (c)***.

Parallel Wacomm + GPU Cuda

Interval Thread [2,8]





Funded by

Modeling, forecasting and mapping of pollution by escherichia coli and salmonella in coastal areas dedicated to mussel production.

project.

IZS ME 04/12 RC/C78C120017001

http://meteo.uniparthenope.it

Current GVirtuS limitations

- The CUDA plugin is **not** thread-safe at the consumer/front-end side

- The management of the **pinned memory** and **unified memory** will be a serious challenge

Conclusions

- GVirtuS works with CUDA 6.x

- Clusters of (*inexpensive*) ARM computing nodes can share one or more high-end GPUs hosted on x86_64 machines.

- Clusters of **x86_64** or **ARM** machines can consume GPUs available on the cloud
- Virtual clusters of x86_64 machines could share virtual GPUs

Future directions

Get, test and extend

- High Performance Scientific Computing Smart-Lab: http://hpsc.uniparthenope.it

- The source code public repository: https://bitbucket.org/montella/gvirtus-dist