

The background features a large red triangle on the left and a black triangle on the right. A white square frame is positioned in the center, containing a perspective view of a digital corridor. The corridor's walls and floor are composed of a grid of small, glowing red and yellow cubes, creating a sense of depth and digital architecture.

AMD

Fusion<sup>12</sup>  
DEVELOPER SUMMIT



**Fusion**<sup>12</sup>  
DEVELOPER SUMMIT

# ***ASSESSING THE RELEVANCE OF APU FOR HIGH PERFORMANCE SCIENTIFIC COMPUTING***

Issam SAID  
Total / UPMC-LIP6  
PhD candidate

*issam.said@lip6.fr*

Joint work with:  
Henri CALANDRA , *Total*  
Romain DOLBEAU, *CAPS Entreprise*  
Pierre FORTIN, *UPMC-LIP6*  
Jean-Luc LAMOTTE, *UPMC-LIP6*

*CONTEXT*



# INTRODUCTION

- Study of depth imaging applications on AMD Fusion APUs.
- Closely follow the road map of the Fusion products.
- Try to determine how far does the APU qualify for seismic applications.



## INTRODUCTION / PCI Express bottleneck

- Graphic Processing Units (GPUs) have developed very rapidly in recent years.
- They become valuable choice for a wide range of scientific applications.
- Despite the impressive computation power and fast internal memory of GPUs, applications with high CPU-GPU communication requirements can be bottlenecked by the low transfer rate of the PCI Express bus. For example: depth imaging applications on GPU.
- APUs may address this problem by removing the PCI interconnection and combines both CPU and GPU in a low power consuming chip.
- In the scope of this work, we only consider using the integrated GPU of an APU as it represents the major computation power (**Trinity**: 77% ).
- **But:**
  - Integrated GPUs are one order of magnitude less compute powerful than discrete GPUs
  - Integrated GPUs have lower memory bandwidth than discrete GPUs
- **Can we expect the integrated GPUs to be more suitable for a certain range of applications (with an appropriate problem size) than discrete GPUs?**

## ***INTRODUCTION / Work plan***

- In this talk we investigate the relevance of APUs for High Performance scientific Computing.
- We survey the different data placement strategies and show their impact on applications performances.
- Then we use a 3D stencil OpenCL kernel (in single precision) to compare the APU performance with CPU and discrete GPU.
- We also use a more realistic application based on stencil computations : a wave propagation modeling kernel, to the same comparative study.

# HARDWARE SPECIFICATION

	CPU	Discrete GPUs		APU Integrated GPUs	
Micro-architecture	Thuban	Cayman	Tahiti	Llano Beaverceek	Trinity Devastator
Model	Phenom	HD6970	HD7970	A8-3850	A10-5700
Clock rate (GHz)	2.8	0.88	0.925	0.6	0.711
Compute units	6	24	32	5	6
Memory size (GB)	8	2	2	0.5	0.5
Peak bandwidth	50	176	256	25.6	25.6
Peak flops (Gflop/s)	134 <sup>1</sup>	2700	3700	480	546

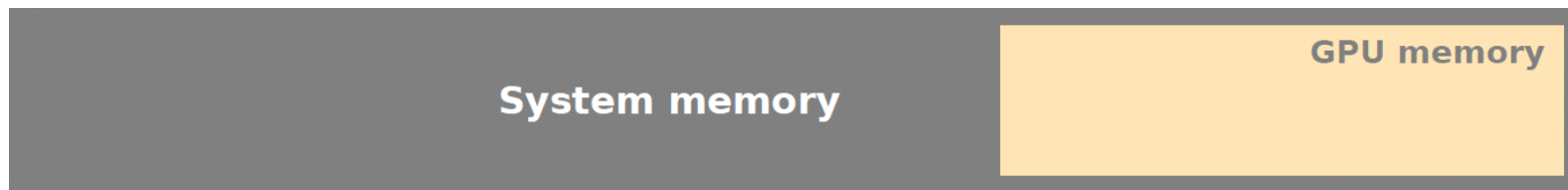
OpenCL 1.1, Windows Catalyst 12.1 driver, AMD APP SDK 2.6

<sup>1</sup> considering one add operation concurrent to one multiply operation on each cpu clock

# ***APU DATA PLACEMENT STRATEGIES***

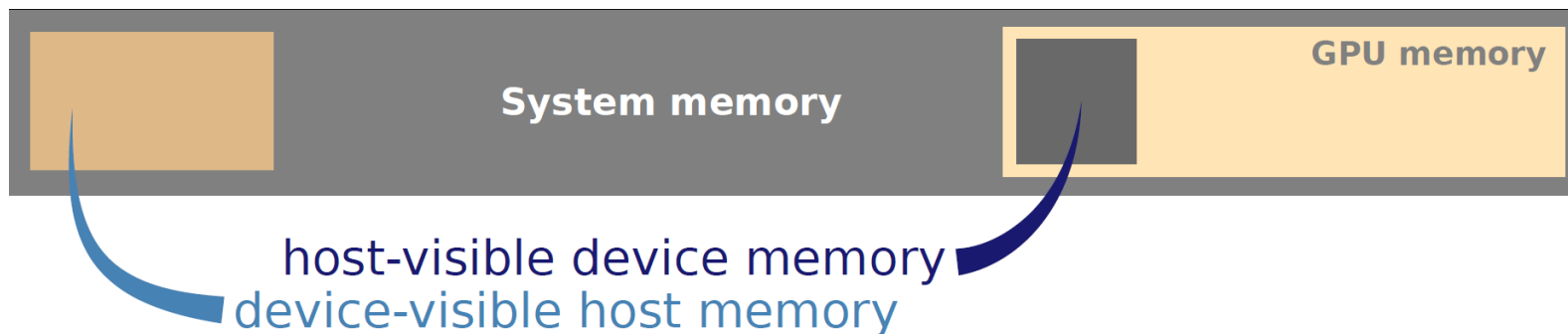


**Fusion**<sup>12</sup>  
DEVELOPER SUMMIT



- The integrated GPU memory is a sub-partition of the system memory.
- Compute units can access memory using 2 buses:
  - **GARLIC (fast bus)**: maximum theoretical transfer rate is about 25.6 GB/s.
  - **ONION (slow bus)**: maximum theoretical transfer rate is about 8 GB/s.
- Memory objects can be shared between CPU (**host**) and the integrated GPU (**device**): **zero-copy** buffers (available only with Windows drivers).

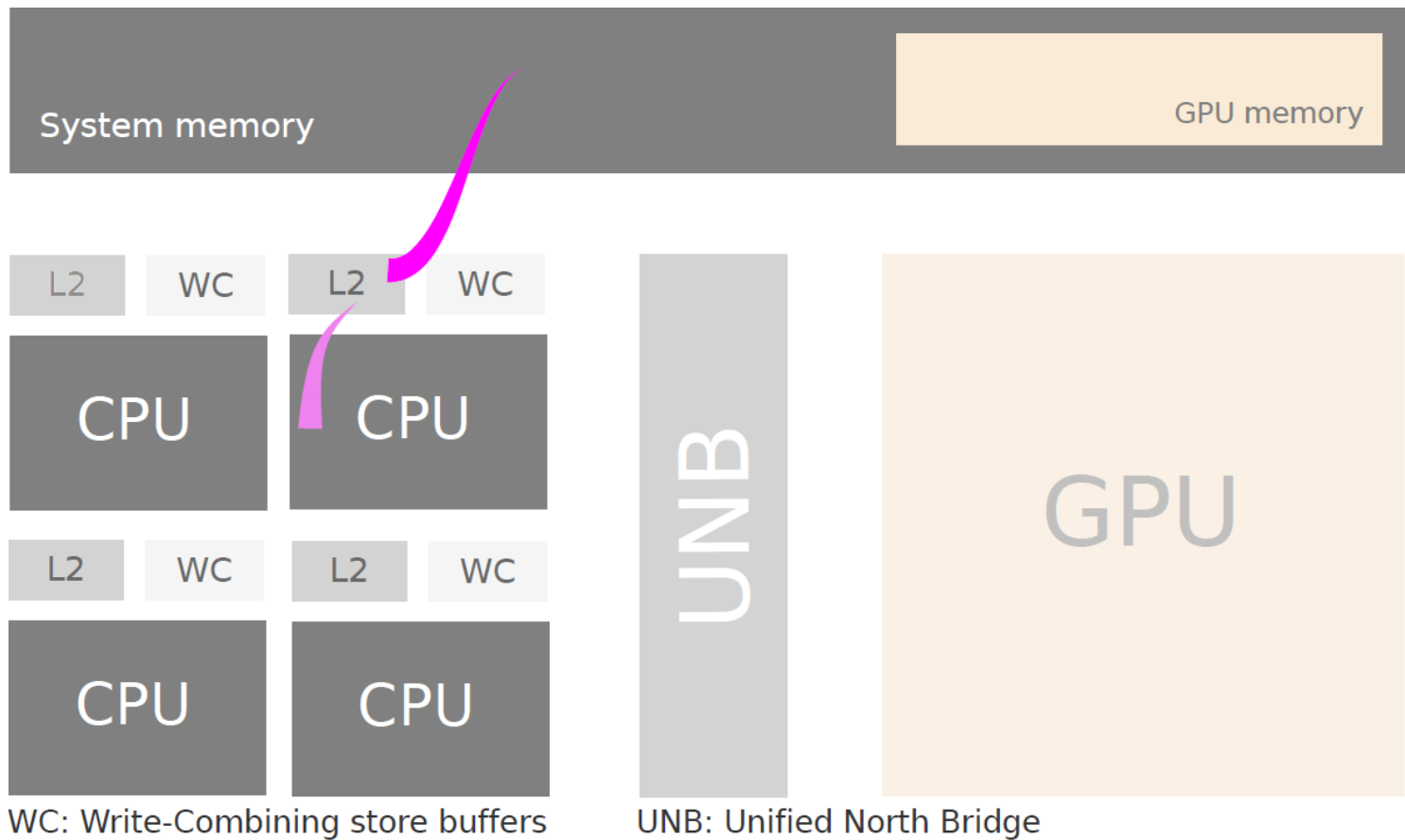
## APU MEMORY SYSTEM / Memory locations



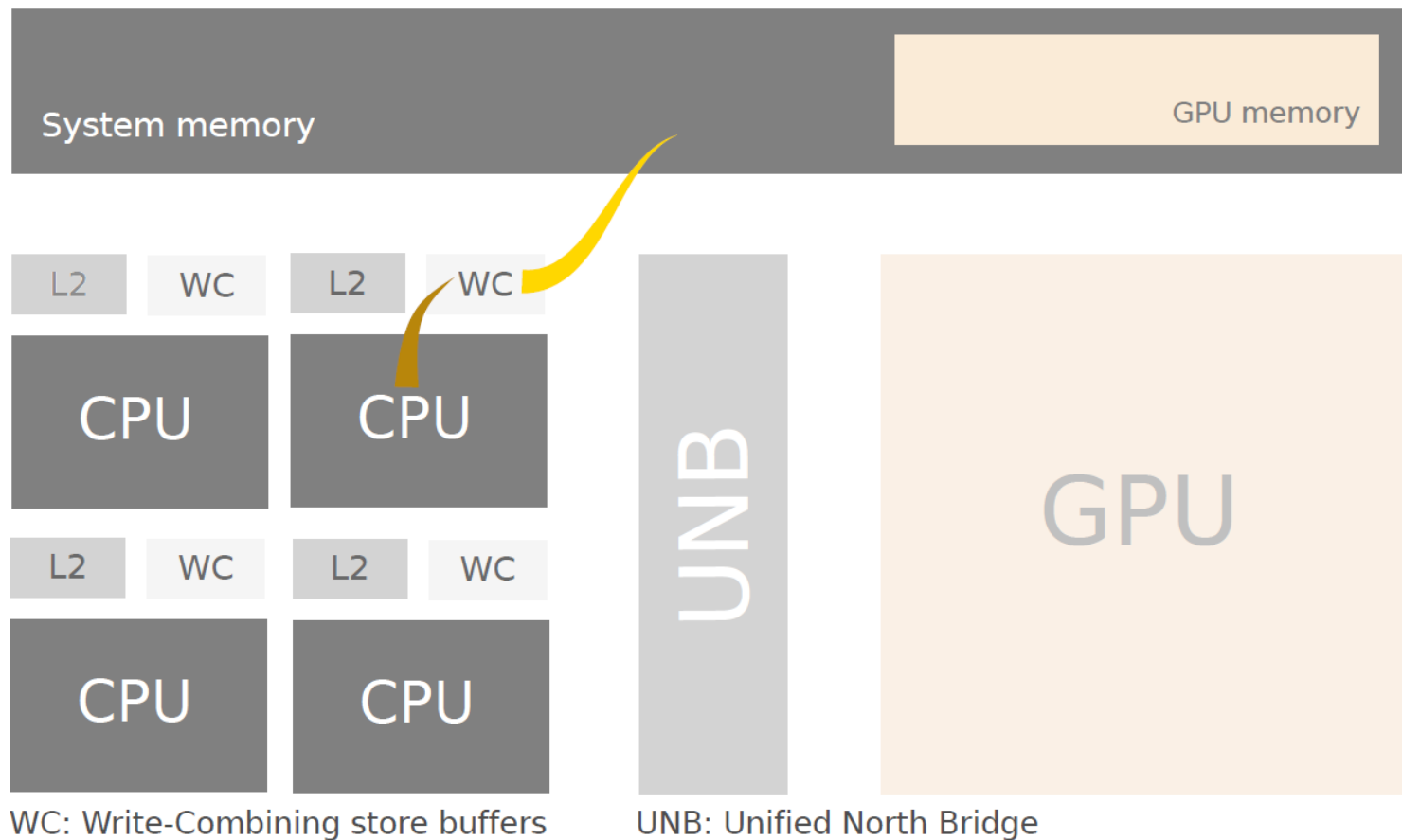
- The device can **access** a limited memory space of the host and vice versa.
- Within an APU, the possible memory locations are:
  - **cacheable** memory: **"c"** (pinned for efficient data transfer between CPU and GPU)
  - GPU memory: **"g"**
  - Zero copy buffers: **"z"** in device-visible host memory
  - **USWC**: **"u"** (Windows only), zero copy buffers with efficient contiguous CPU writes and efficient GPU reads
  - **GPU persistent** or host-visible device memory **"p"**

USWC: Uncacheable Speculative Write Combining

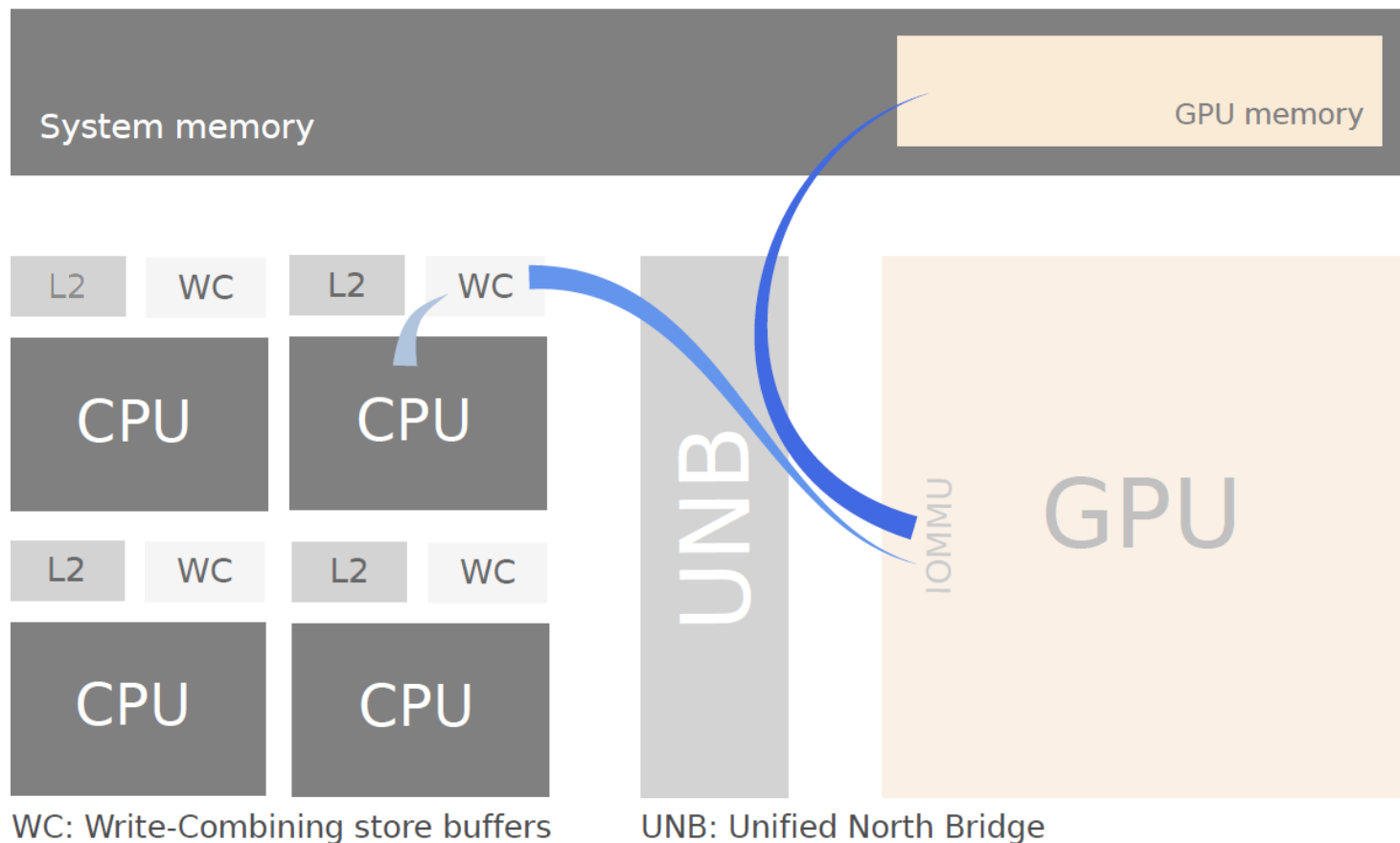
## CPU TO CACHEABLE MEMORY «C»



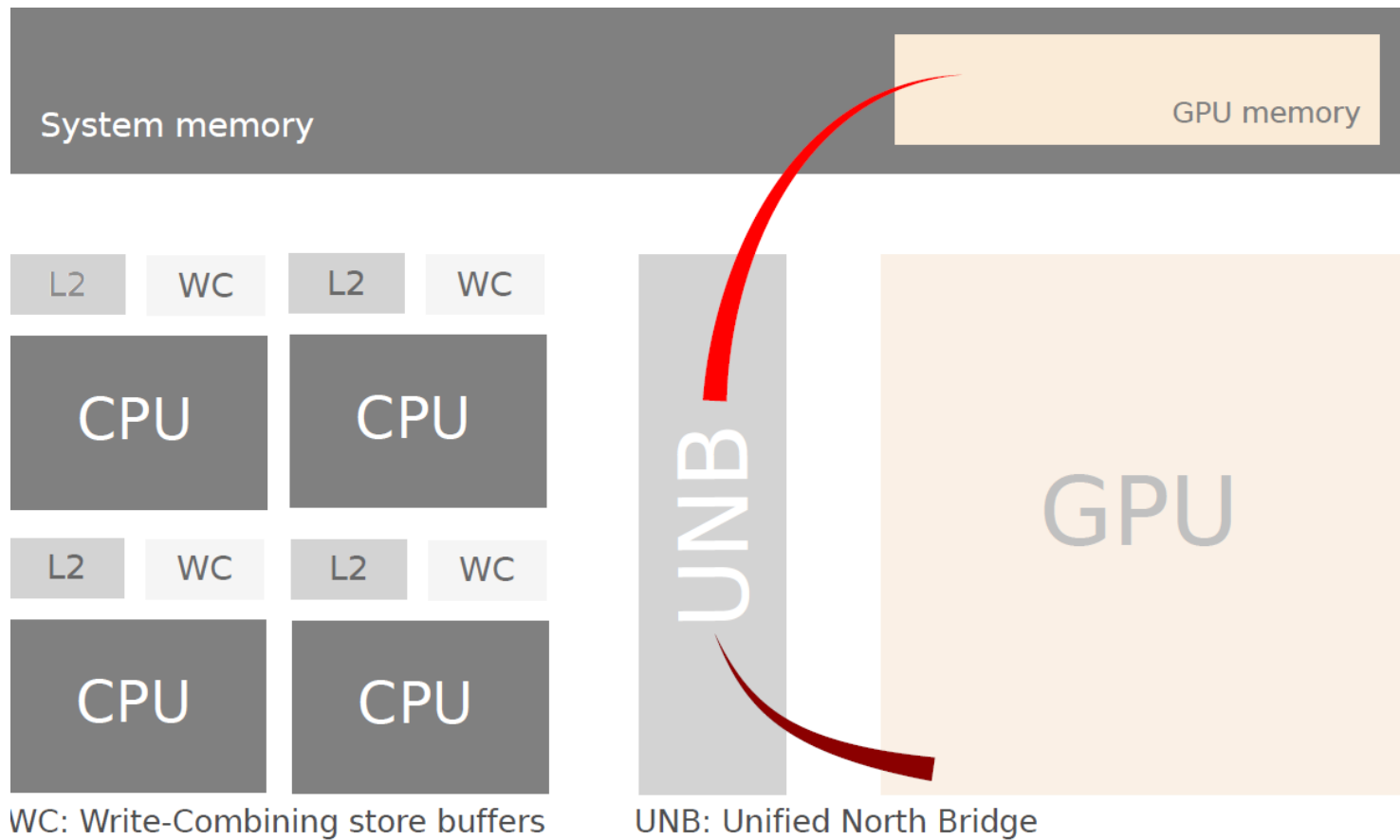
## CPU TO USWC «U»



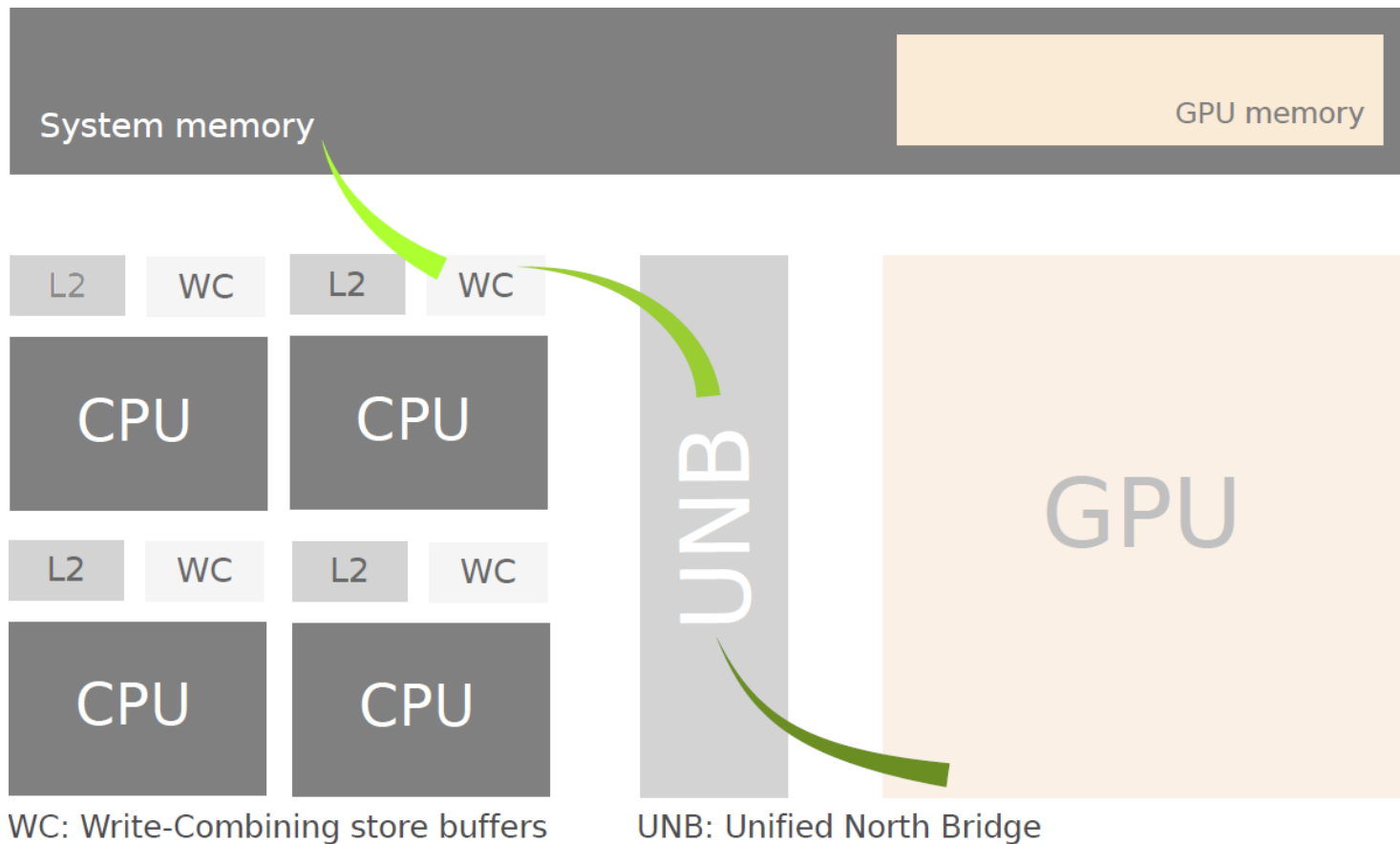
## CPU TO GPU PERSISTENT MEMORY «p»



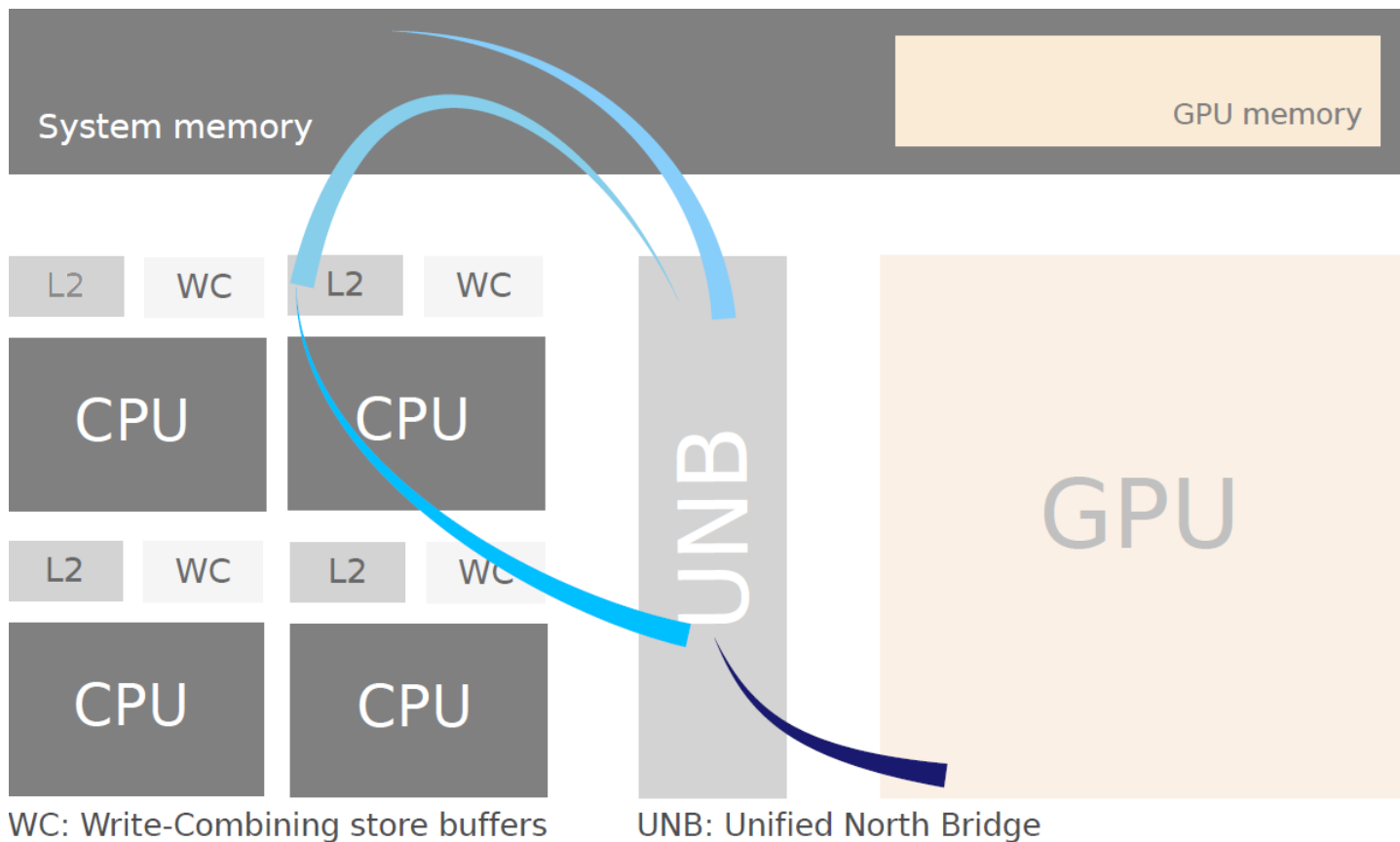
## GPU TO GPU MEMORY «g»



## GPU TO USWC «u»



## GPU TO PINNED HOST MEMORY «Z»



## DATA PLACEMENT ON APU

- There are multiple choices for an application to transfer data between CPU and GPU within an APU :
  - “**cg**”: explicit data copy from the CPU partition “**c**” to the GPU partition “**g**”
  - “**gc**”: explicit data copy from the GPU partition “**g**” to the CPU partition “**c**”
  - “**z**”: no data copy but slower GPU access
  - “**u**”: no data copy but GPU read-only (via **GARLIC**)
  - “**p**”: no data copy but slower CPU access
- Data placement is a performance key factor on APUs.
- The use of the **GARLIC** bus is strongly recommended.
- In order to leverage good performances, users need to find the most efficient data placement strategy for input and output buffers of a GPU kernel.

# DATA PLACEMENT BENCHMARK

We develop a benchmark that moves data back and forth the different locations of the APU memory. We try different combinations , and use the following dataflow:

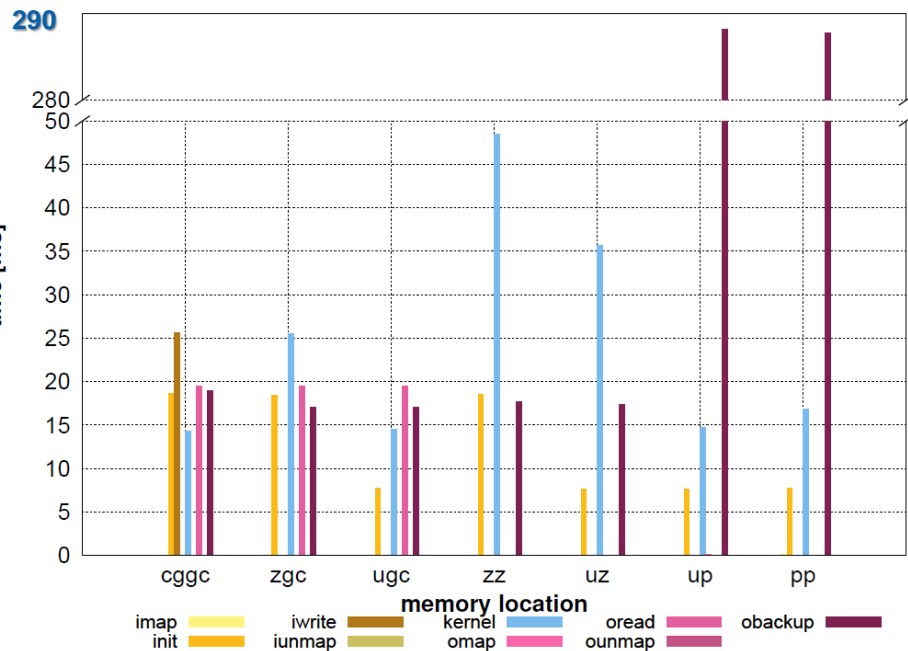
- Map input buffer when needed “**imap**”
- Initialize input buffer “**init**”
- Copy input to the GPU memory space when needed “**iwrite**”
- Unmap input if already mapped “**iunmap**”
- Run OpenCL kernel “**ktime**” (memory copy kernel)
- Map output buffer if necessary “**omap**”
- Copy output buffer from the GPU memory space when needed “**oread**”
- Unmap output buffer if already mapped “**ounmap**”
- Copy output buffer to a temporary host buffer to make sure that the data resides on the CPU memory space “**obackup**”

## ***BENCHMARKING RULES***

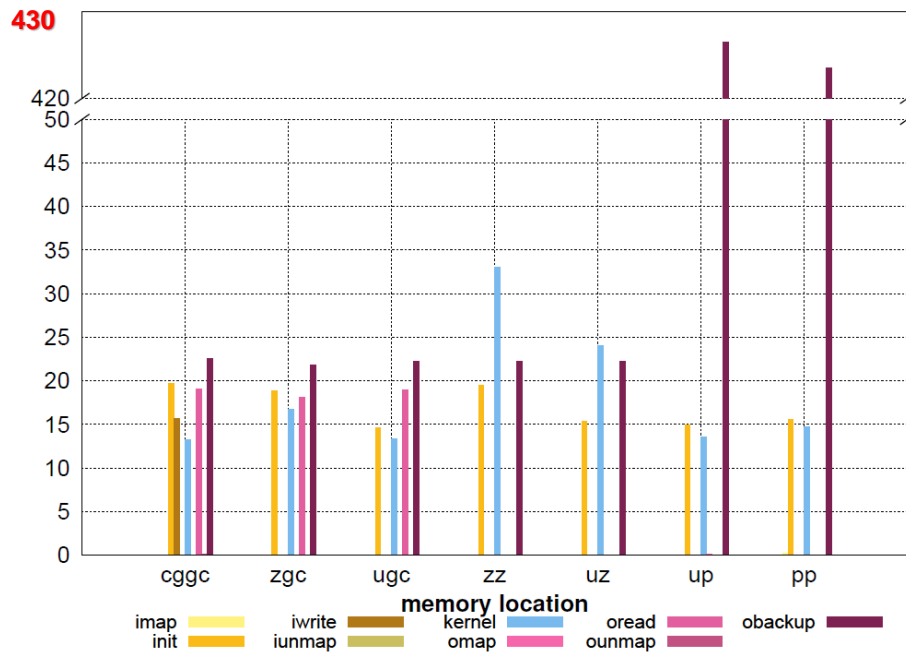
- We use system wall-clock for timing.
- We run each OpenCL kernel multiple times (up to 40) after a device “**warm up**”.
- Numerical results of parallel computations are validated against those of serial computations.

# EXPERIMENTAL RESULTS

## Llano – buffer size 128 MB



## Trinity – buffer size 128 MB



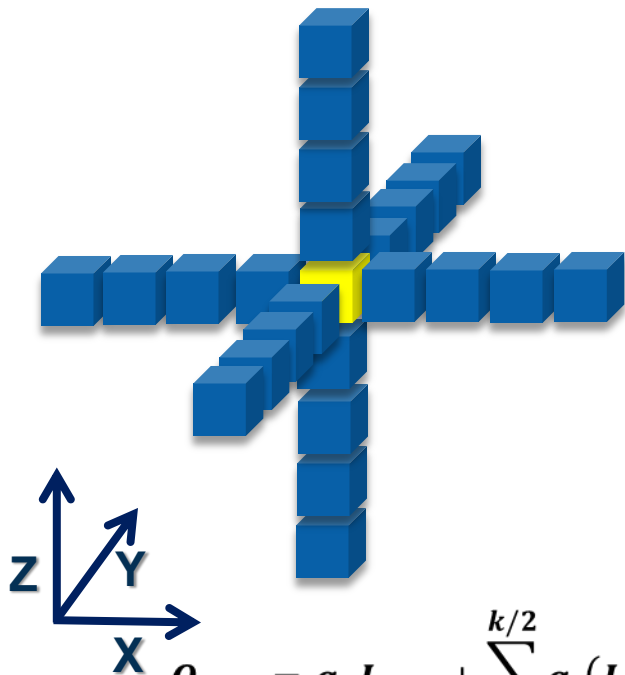
# ANALYSIS

- Explicit data copy rate between CPU and integrated GPU:
  - Is measured at **4** to **5.5 GB/s** when using **ONION**
  - Is measured at **12** to **13 GB/s** when using **GARLIC**
- The GPU reads from USWC are as fast as GPU reads from GPU memory.
- CPU writes to GPU persistent memory are fast but reads are very slow (“**obackup**”).
- CPU contiguous writes to USWC (“**u**”) offer the highest bandwidth.
- Zero copy buffers can be useful as they save memory space on the GPU memory.
- GPU memory accesses to “**z**” (**ONION**) are slower than accesses to “**u**” (**GARLIC**) and “**g**”.
- We select the following strategies:
  - **cggc**
  - **uz**
  - **ugc**
  - **up**

# *STENCIL COMPUTATIONS*



## DEFINITION



- Stencil computations are a class of algorithms that constitute the core of many scientific simulation codes.
- Widely used in direct solution methods for **PDE** (Partial Differential Equation) such as Finite Difference methods.
- A linear summation of an input element and its neighboring values weighed by specific coefficients (**stencil coefficients**).
- A  $k^{th}$  order in space stencil requires  $k$  input elements (neighbors) on each dimension.
- $3 * k + 1$  input elements are required in order to compute one output.
- We use an **8<sup>th</sup>** order 3D space stencil in this work to compare CPU/APU/GPU performance.
- We apply the selected data placement strategies on APUs.

$$O_{x,y,z} = a_0 I_{x,y,z} + \sum_{i=1}^{k/2} a_i (I_{x-i,y,z} + I_{x+i,y,z} + I_{x,y-i,z} + I_{x,y+i,z} + I_{x,y,z-i} + I_{x,y,z+i})$$

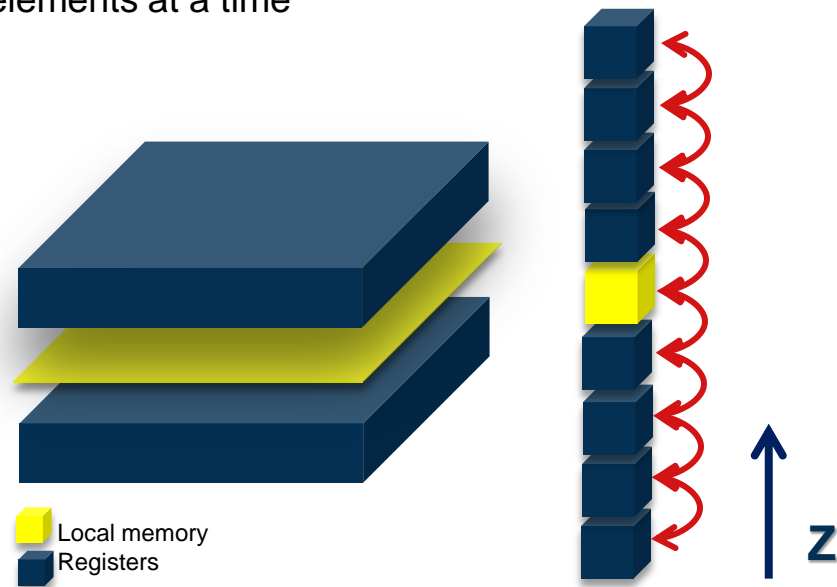
# OPENCL IMPLEMENTATION OF 3D STENCIL COMPUTATIONS

## Kernel description

- We apply a 2D work-item grid on the 3D domain
- We first implement a **scalar** version:
  - each work item computes **X** columns along the Z dimension of the domain ( $X \rightarrow ILP^1$ )
  - **X** is determined via auto-tuning (in most cases  $X=2$  or  $X=4$ )
  - all memory accesses are performed on global memory
- In the second version (**vectorized**) we vectorize the code and use OpenCL **float4** data type:
  - depending on the device register file size, each work-item computes  $4X$  ( $X=2$  or  $4$ ) columns along the Z dimension
- Finally, we use domain tiling in local memory in order to benefit from data reuse (**local vectorized**)

## Blocking in local memory

- Input data is fetched, slice by slice, from global memory to local memory and is efficiently reused within a workgroup to compute multiple output elements at a time

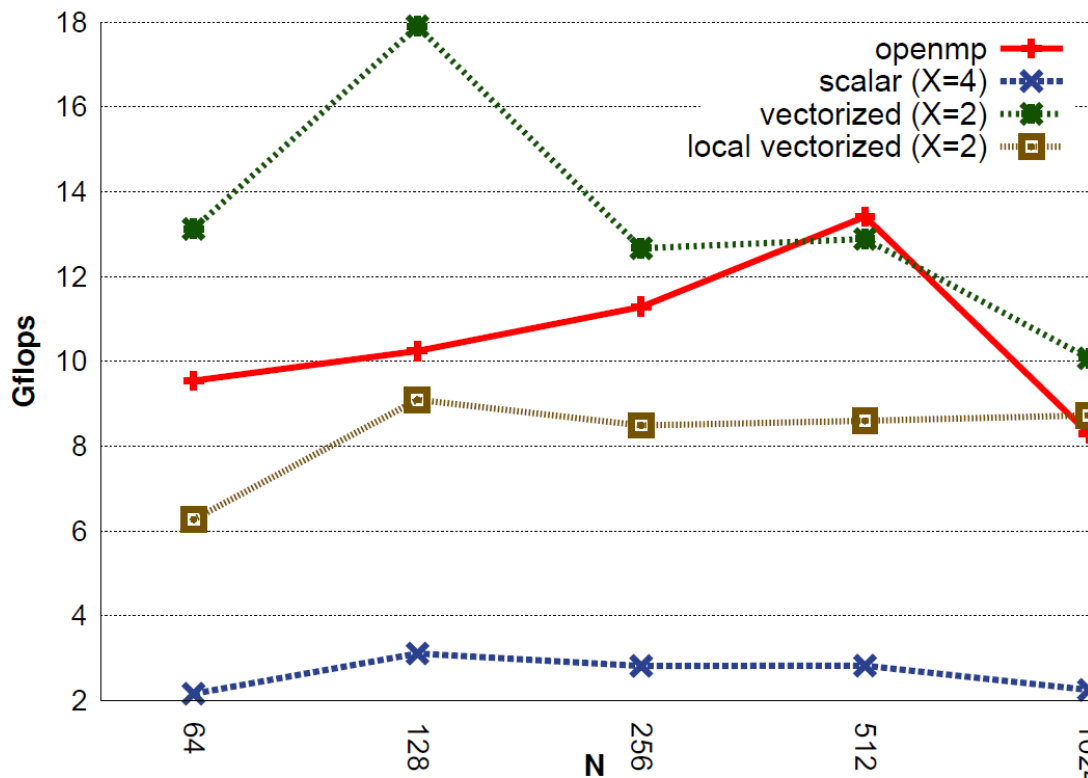


<sup>1</sup> Instruction Level Parallelism

# EXPERIMENTAL RESULTS / CPU

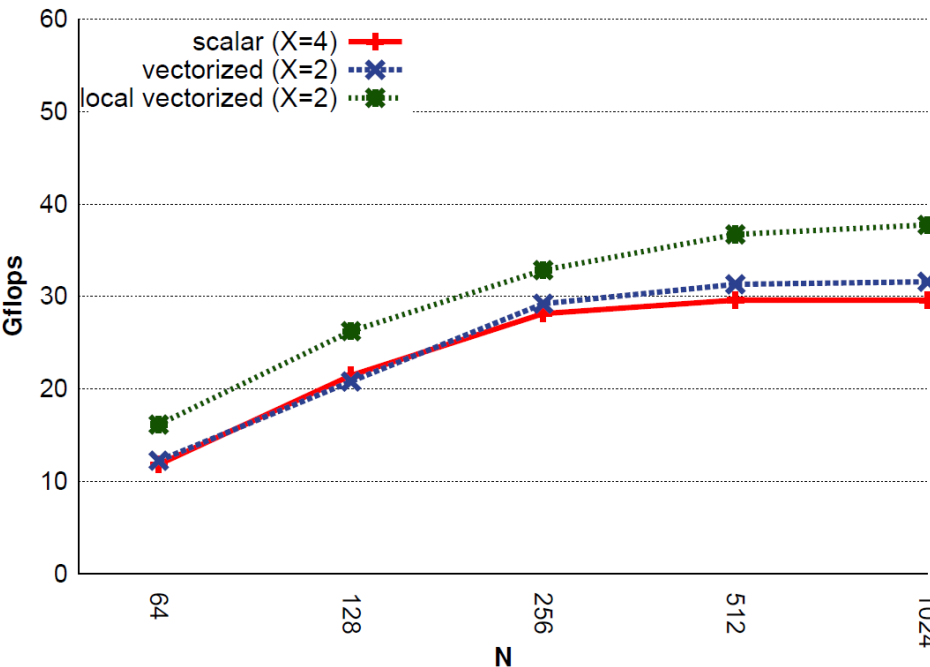
## AMD Phenom TM II x6 1055T

- The domain size varies as  **$N \times N \times 32$** .
- OpenMP F90 code (without domain tiling) compiled with Intel Fortran Compiler.
- OpenCL is faster or as fast as OpenMP .
- Thanks to CPU caches, **vectorized** version is faster than the **local vectorized** implementation.

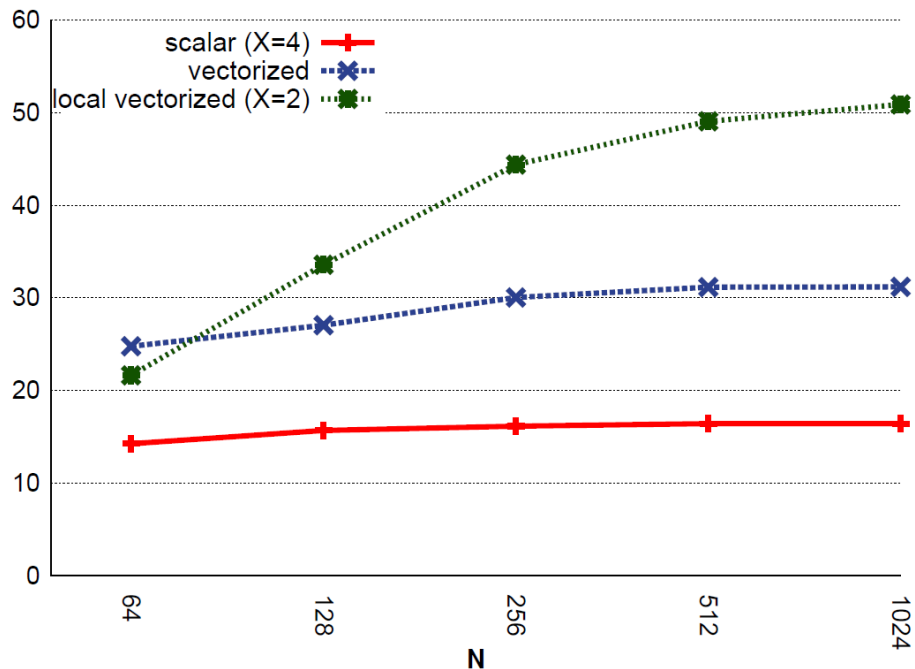


# EXPERIMENTAL RESULTS | Integrated GPUs

## Llano – domain size = NxNx32



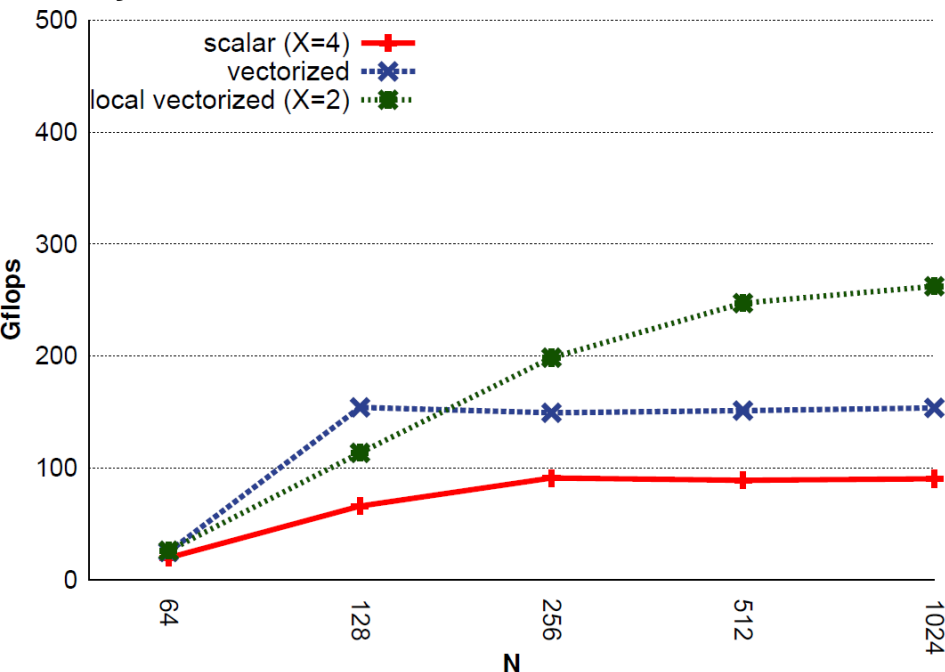
## Trinity – domain size = NxNx32



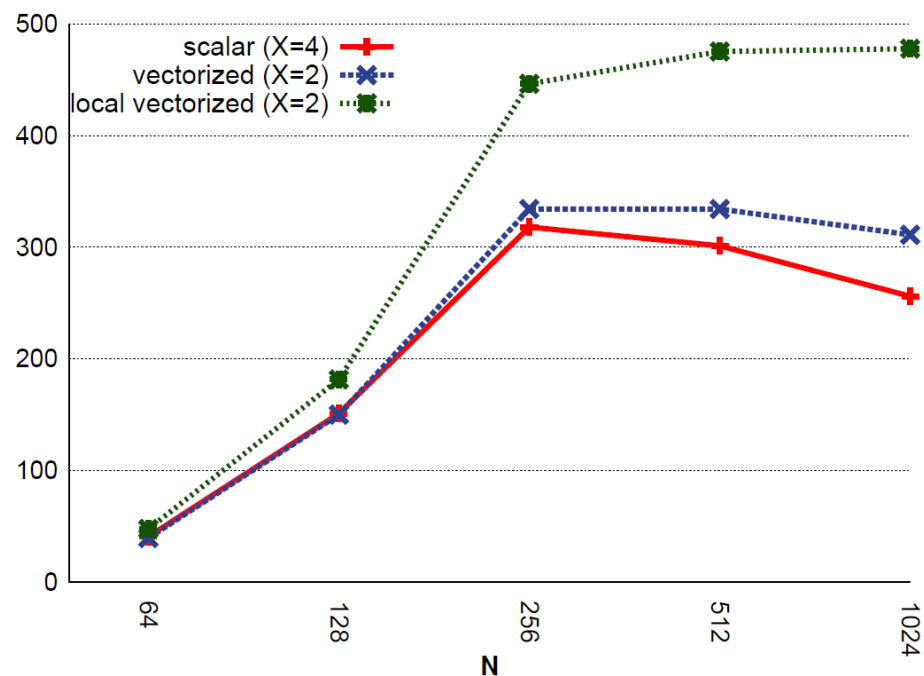
■ The **local vectorized** version outperforms the other implementations.

# EXPERIMENTAL RESULTS / Discrete GPUs

## Cayman – domain size = NxNx32



## Tahiti – domain size = NxNx32



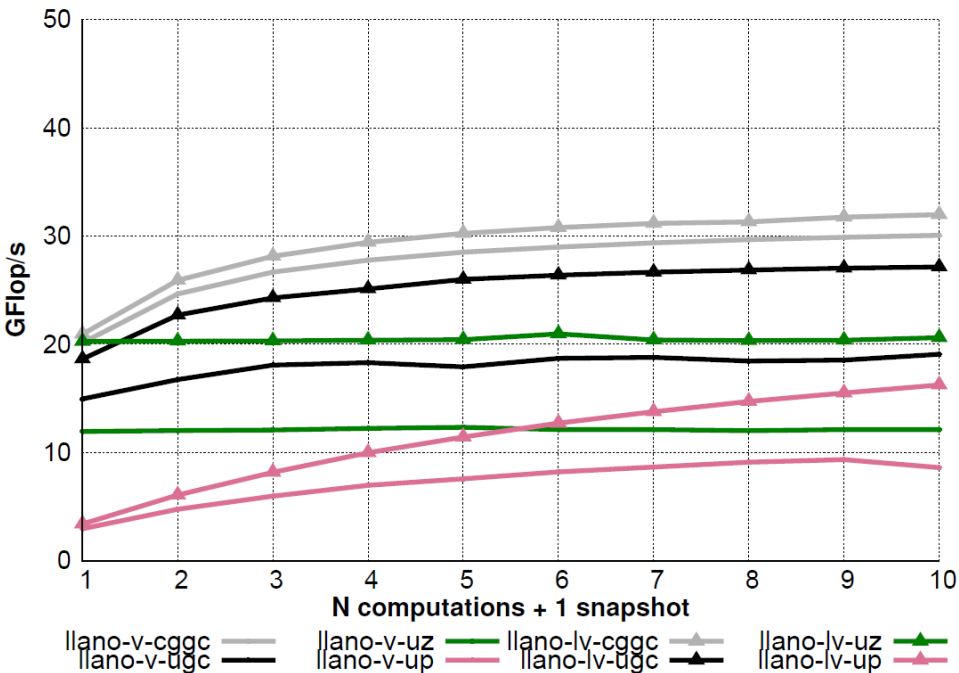
- The **local vectorized** version is the most efficient implementation for all architectures.
- For **Tahiti** the **scalar** version with (X=4) is almost as good as the **vectorized** version. This is due to the new scalar design (Graphic Core Next). A **local scalar** version for **Tahiti** is a work in progress.

## ***SNAPSHOTTING AND DATA PLACEMENT IMPACT***

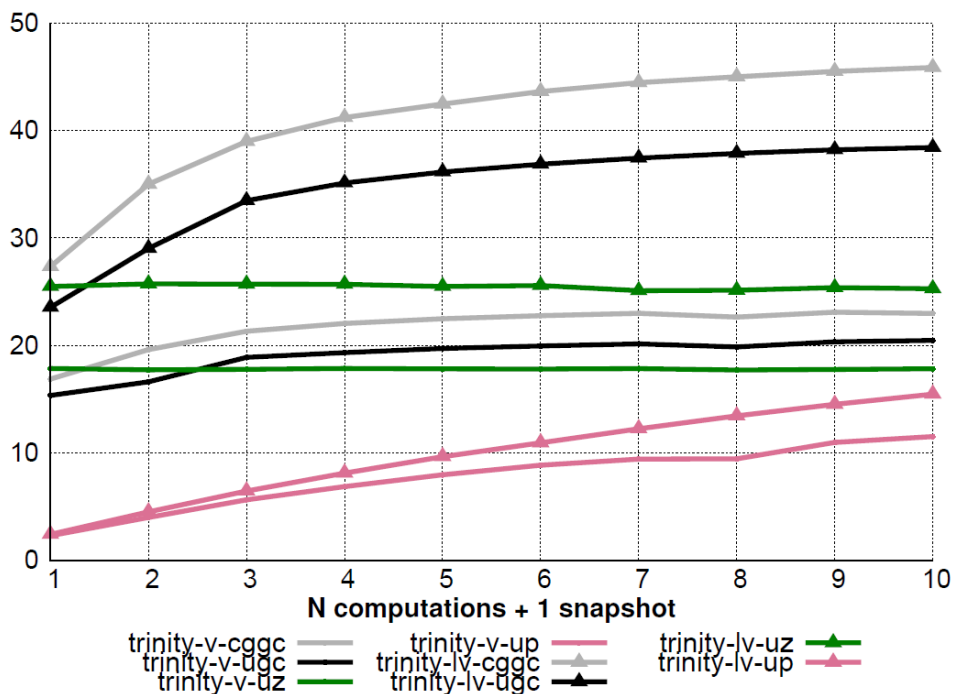
- Stencil computations on GPU requires sending the computed data back to the CPU in order to perform further tasks such as I/O. We denote this process “data snapshotting”.
- We believe that the frequency of data snapshotting can also be a performance key factor and also an additional parameter of our comparative study.
- We run the 3D stencil kernel multiple times and measure its performance as a function of the snapshotting frequency.
- Also we run the 3D stencil kernel while taking into consideration multiple data placements.

# EXPERIMENTAL RESULTS | Impact of data placement on APU performance

## Llano, domain size: 1024x1024x32



## Trinity, domain size: 1024x1024x32

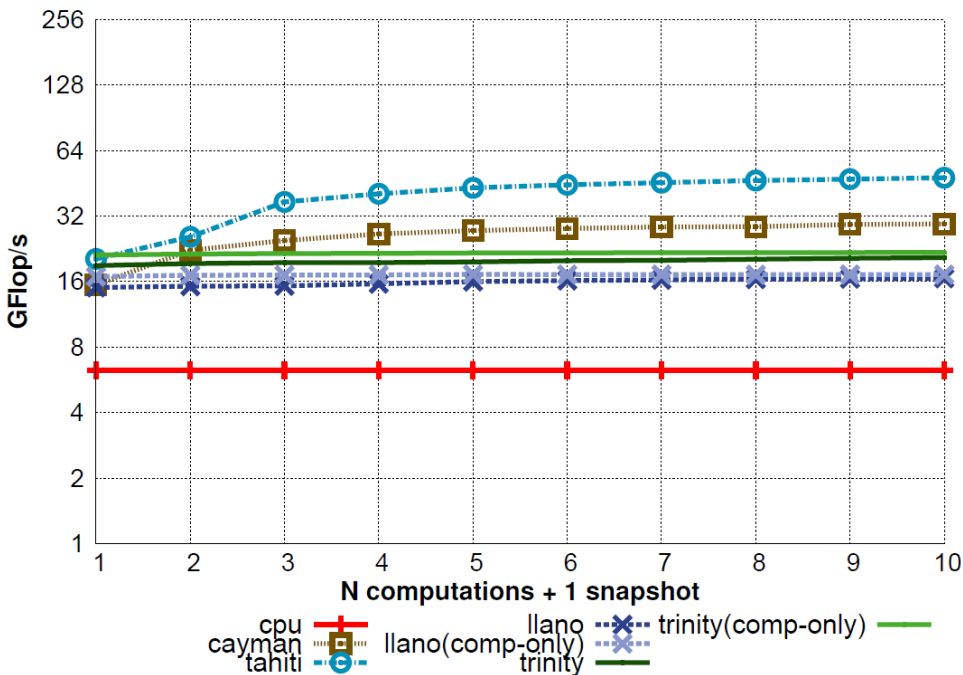


▪ “v” denotes the **vectorized** version and “lv” denotes the **local vectorized** version.

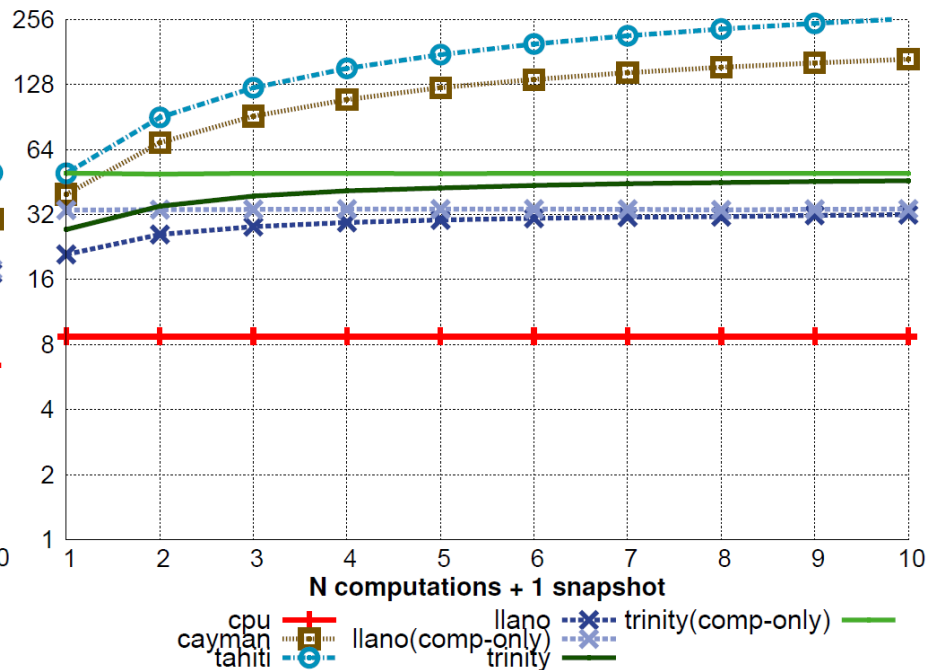
▪ “cggc” with “lv” appears to be the most efficient data placement strategy for the stencil kernel.

# EXPERIMENTAL RESULTS | CPU/APU/GPU comparison

## Domain size: 64x64x32



## Domain size: 1024x1024x32

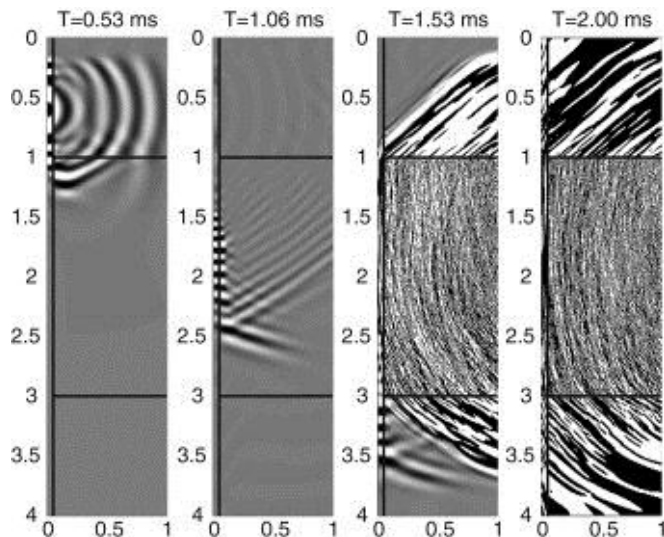


- “**comp-only**” denotes performance measurements without taking into consideration the cost of data copies between the CPU and the integrated GPU.

# *SEISMIC WAVE PROPAGATION SIMULATION*



## WAVE PROPAGATION / Definition



(Lisitsa & Lys, J. Comput. Appl. Math., 234(6), 1803-1809, 2010)

- We extend the stencil computations to a real world application: an **acoustic** wave propagation modeling kernel.
- We use **FDTD (Finite Difference Time Domain)** method to find a time-domain solution of the **3D** wave equation:

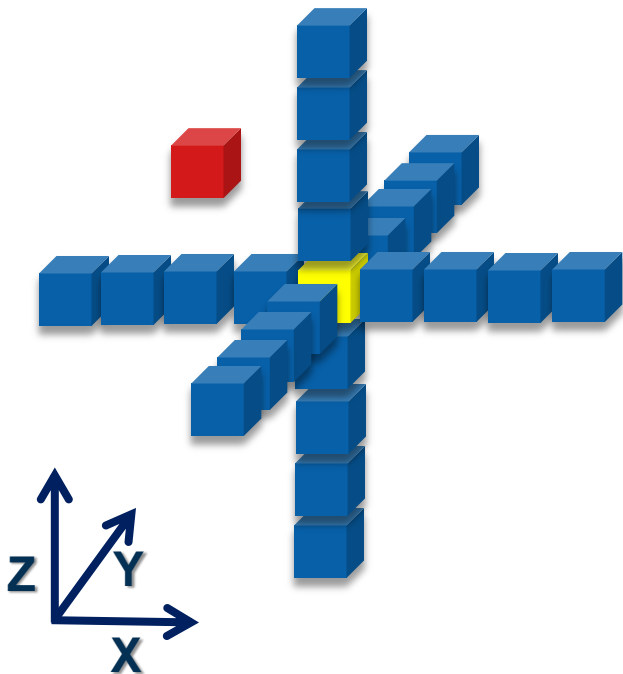
$$\frac{1}{c^2 \rho} \frac{\partial^2 u}{\partial t^2} - \nabla \left( \nabla \frac{1}{\rho} u \right) = f(t) \equiv \frac{1}{c^2 \rho} \frac{\partial^2 u}{\partial t^2} - \frac{1}{\rho} \Delta u = f(t)$$

- Where ***u*** is the particle velocity, ***c*** is the wave velocity, ***ρ*** is the acoustic density of the used medium, and ***f*** is the source terms (second derivative of the Gaussian function).
- For the sake of simplicity we consider ***ρ*** constant, the equation is simplified to:

$$\frac{1}{c^2} \frac{\partial^2 u}{\partial t^2} - \sum_{r \in x,y,z} \frac{\partial^2 u}{\partial r^2} = f(t) = cste * e^{-\alpha(t-t_0)^2}$$

- We need to find ***u(r(x,y,z), t)*** for all ***r*** vectors of the domain at each time step ***t***.

## WAVE PROPAGATION / Numerical scheme

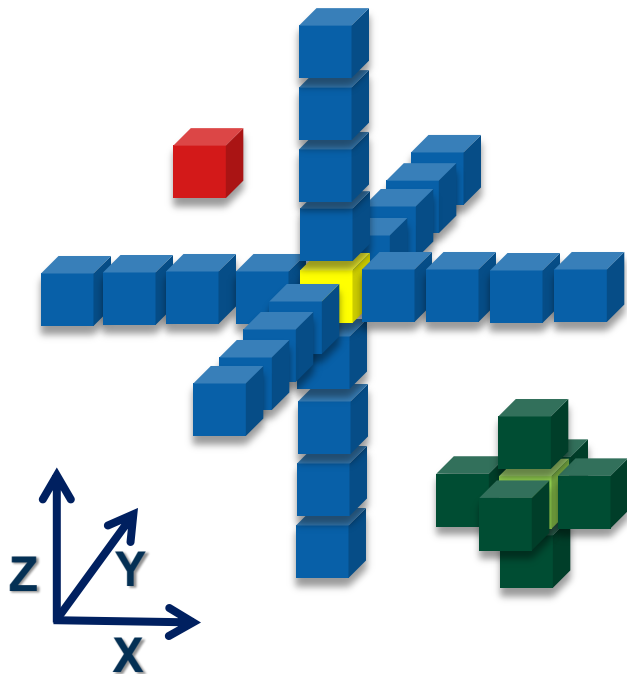


- The computational domain is represented by a regular mesh.
- We use an **8<sup>th</sup>** order 3D stencil for space discretization.
- We use the **leapfrog** time integration method (**2<sup>nd</sup>** order stencil) for time discretization.
- The numerical solution is expressed as follows:

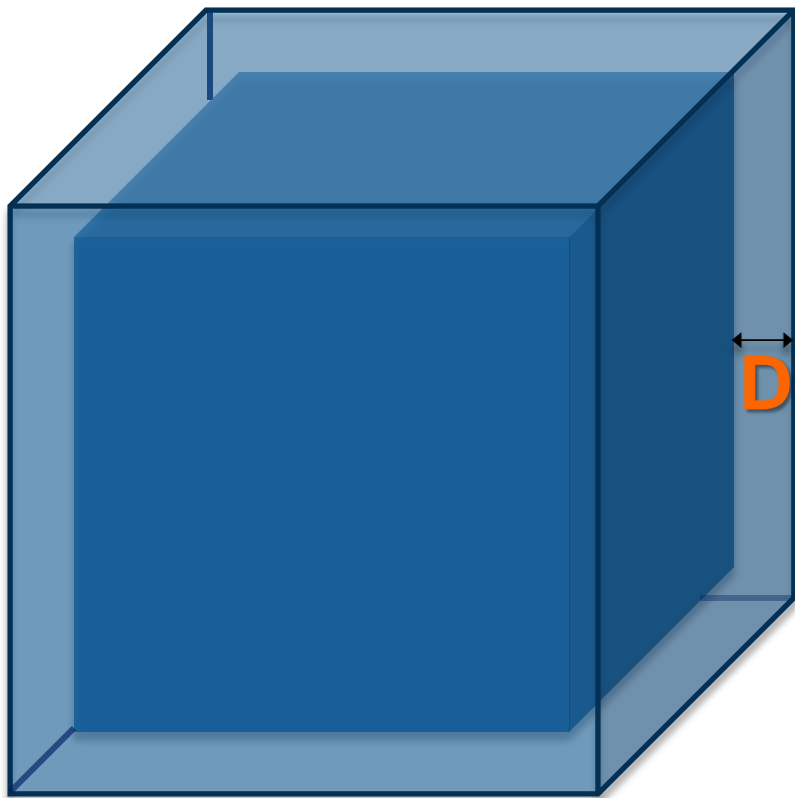
$$u_{x,y,z}^{n+1} = 2u_{x,y,z}^n - u_{x,y,z}^{n-1} + \frac{c^2}{\Delta h^2} \sum_{i=-k/2}^{k/2} a_i (u_{x+i,y,z}^n + u_{x,y+i,z}^n + u_{x,y,z+i}^n)$$

- Where  $u_{x,y,z}^n$  is the **wave field** at time step  $n$  of the particle  $(x, y, z)$ .
- $(3 * k + 1) + 1$  input elements are required for one output.

## WAVE PROPAGATION / Boundary conditions



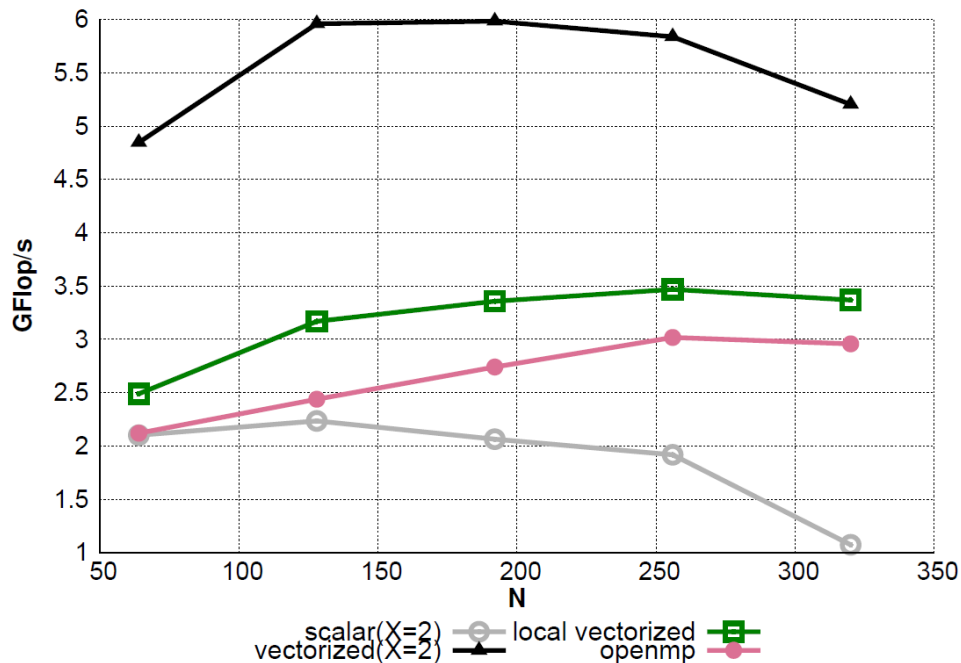
- We also consider **boundary conditions**.
- The velocity is **nil** on the domain boundaries, which generates spurious wave reflexions that spoil the solution everywhere in the grid.
- We use **PML** (**P**erfectly **M**atched **L**ayer) method to **absorb** the wave fields energy on the boundaries.
- Fictitious absorbing layers on each grid dimension.
- Inside each absorbing layer a damping term is added to the wave equation.
- $(3 * k + 1) + 1 + 7$  input elements are required for one output (the damping terms are computed using a 2<sup>nd</sup> order stencil).



- Similar to the previous stencil computations.
- The domain is divided in 2 subdomains:
  - Inner domain: without PML damping
  - Outer domain: with PML damping
- 2 different numerical computations.
- We apply the same optimizations and data placement strategies as discussed previously.
- APU/GPU computations can be subject of **branch divergence** when **work-items** of the same **wave-front** are assigned to both inner domain and outer domain which impacts the kernel performance (10% of performance enhancement on **Tahiti** when switching **D** from 18 to 16) .
- The OpenCL kernel is tuned enough for a comparative study between the described architectures (further optimizations are in progress).

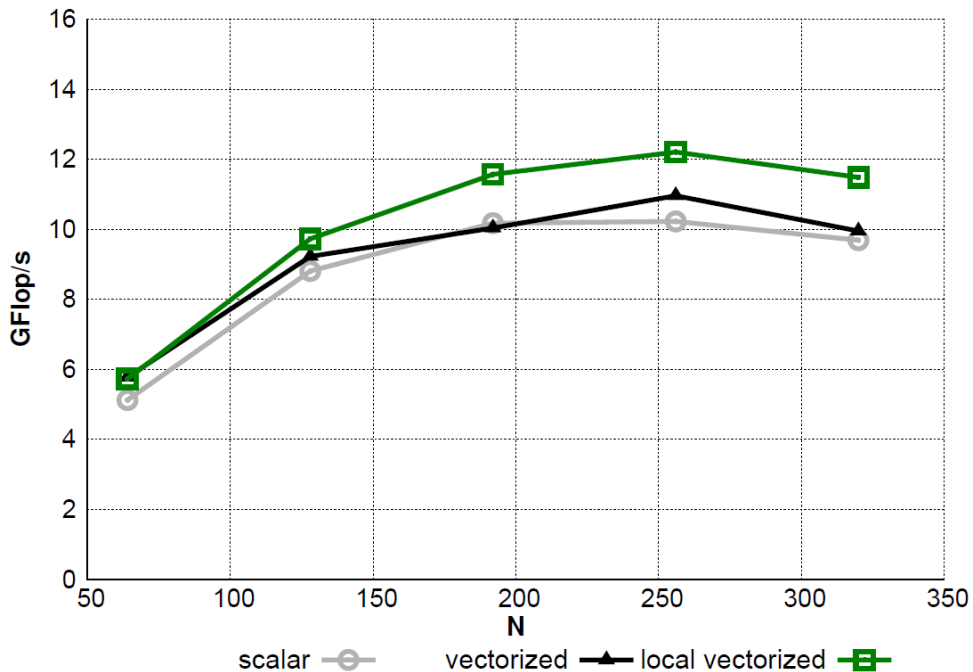
## EXPERIMENTAL RESULTS / CPU

- **AMD Phenom TM II x6 1055T**
- The domain size varies as  **$N \times N \times N$** .
- OpenMP (F90 code compiled with Intel Fortran Compiler).
- On the CPU the **vectorized** version is the most efficient version.
- OpenCL is faster than OpenMP (without domain tiling) on the CPU.

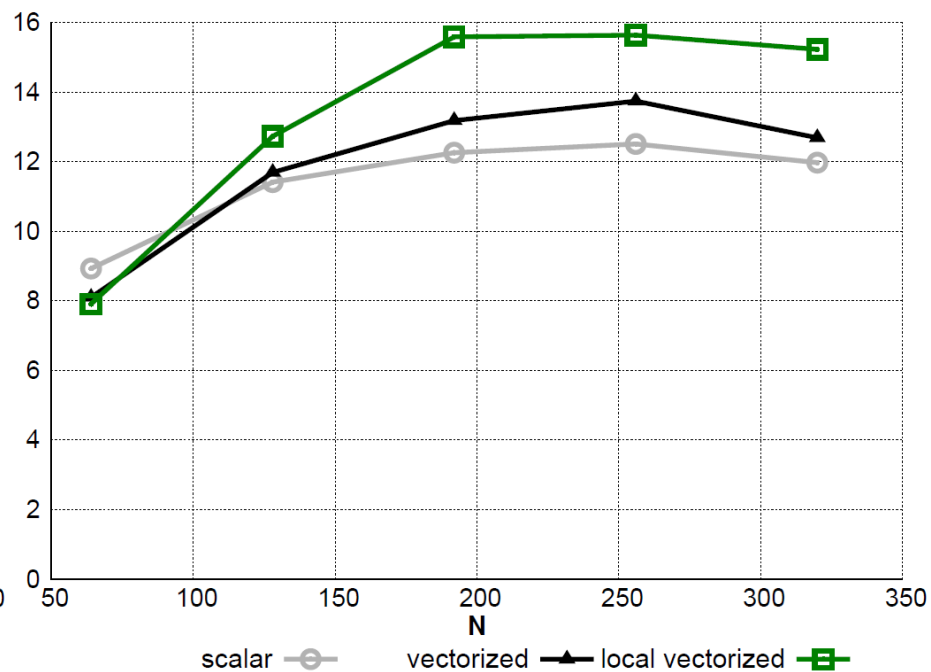


# EXPERIMENTAL RESULTS / Integrated GPUs

## Llano – domain size = NxNxN



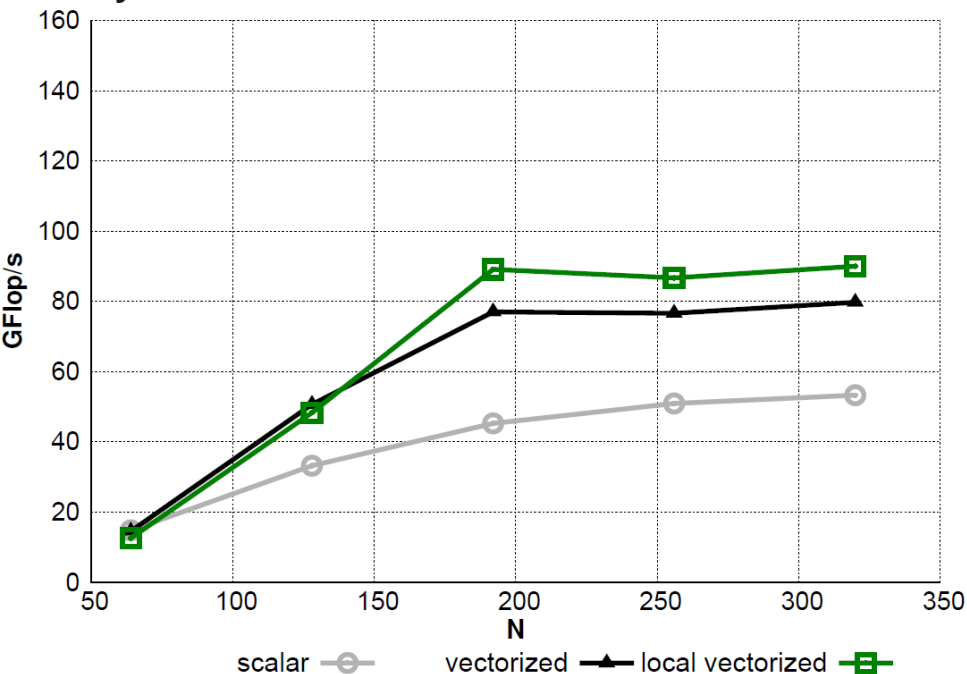
## Trinity – domain size = NxNxN



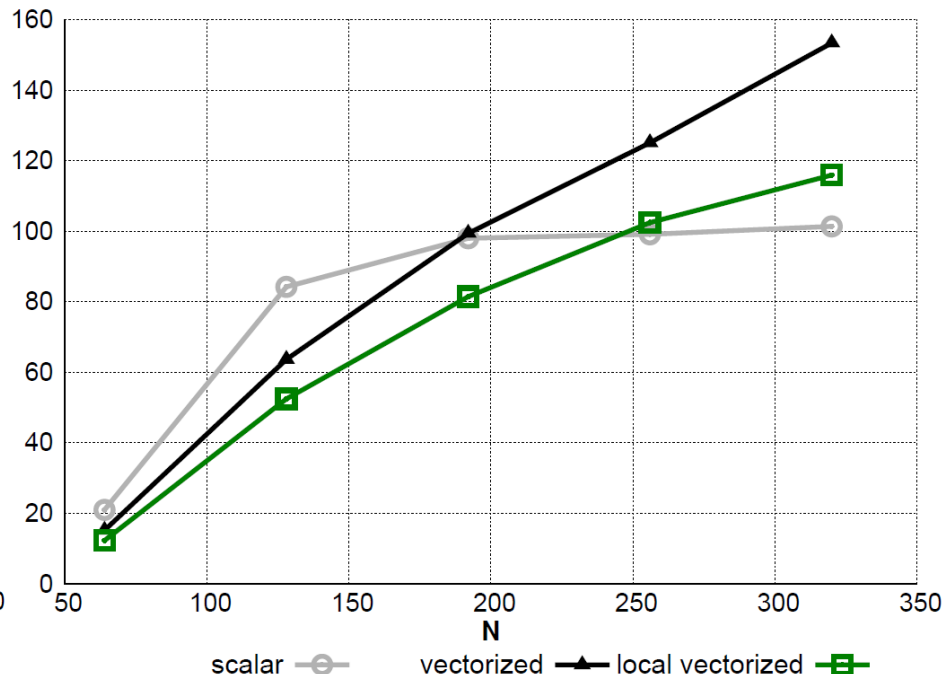
- The **local vectorized** version is the most efficient implementation for APUs.

## EXPERIMENTAL RESULTS / Discrete GPUs

Cayman – domain size =  $N \times N \times N$



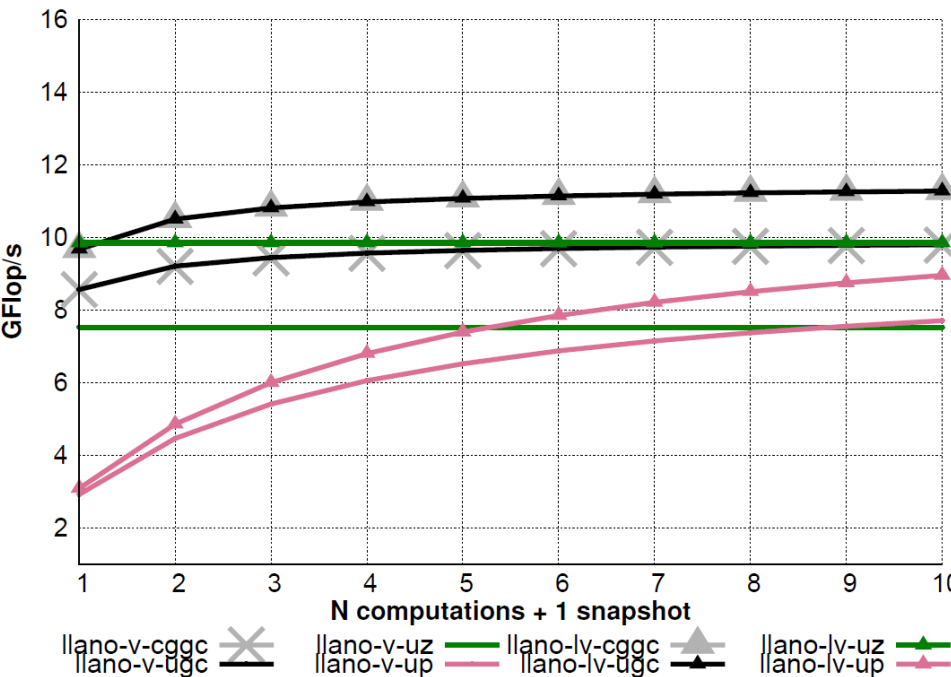
Tahiti – domain size =  $N \times N \times N$



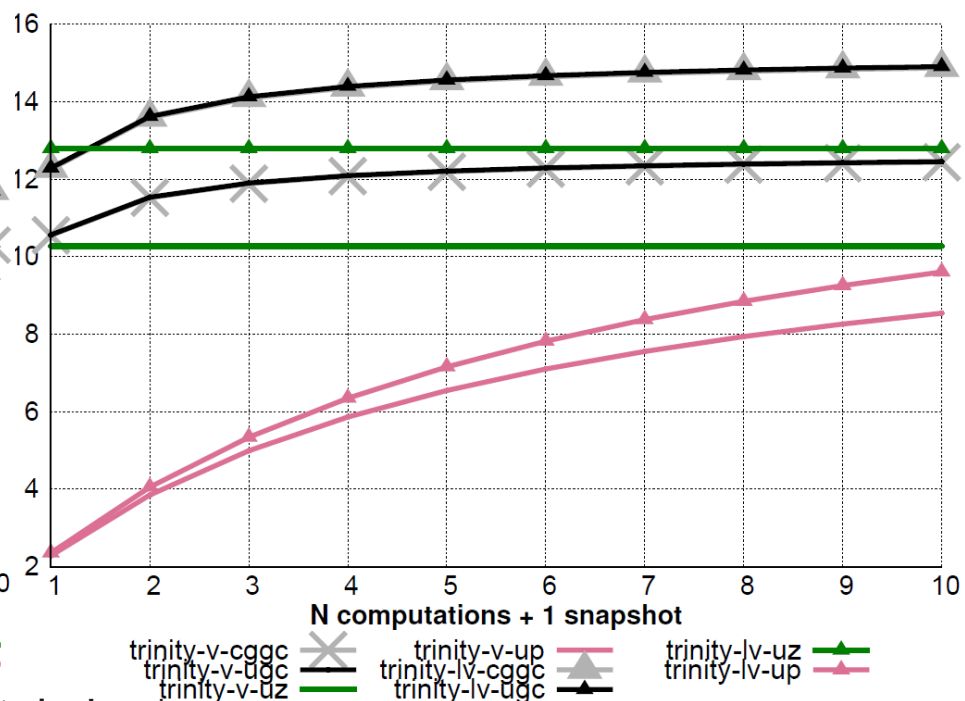
- The **local vectorized** version is the most efficient implementation for **Cayman** but not for **Tahiti** which is unexpected.

# EXPERIMENTAL RESULTS | Impact of data placement on APU performance

## Llano, domain size = 320x320x320



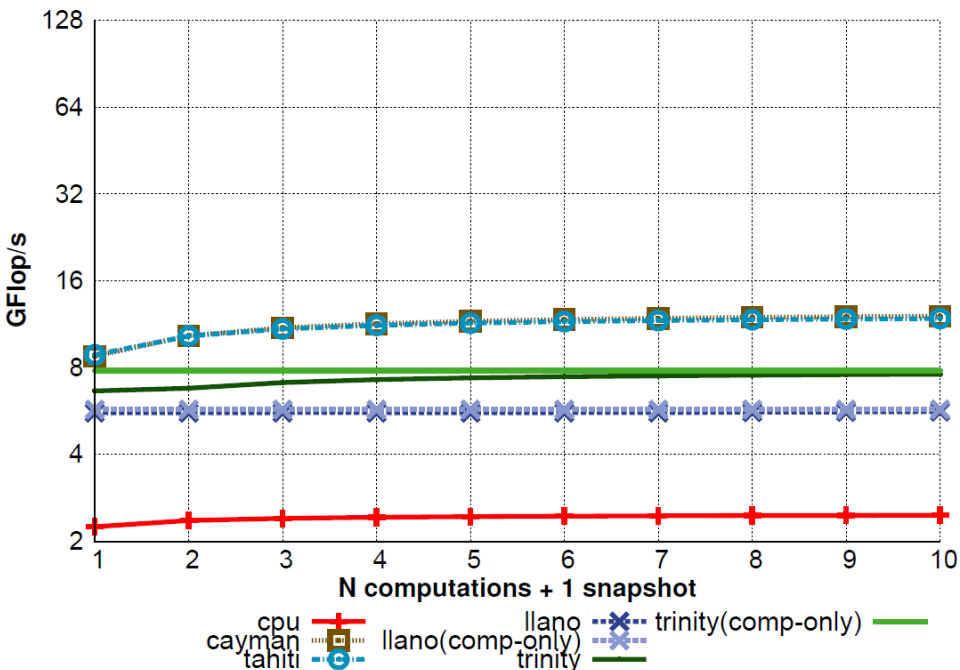
## Trinity, domain size = 320x320x320



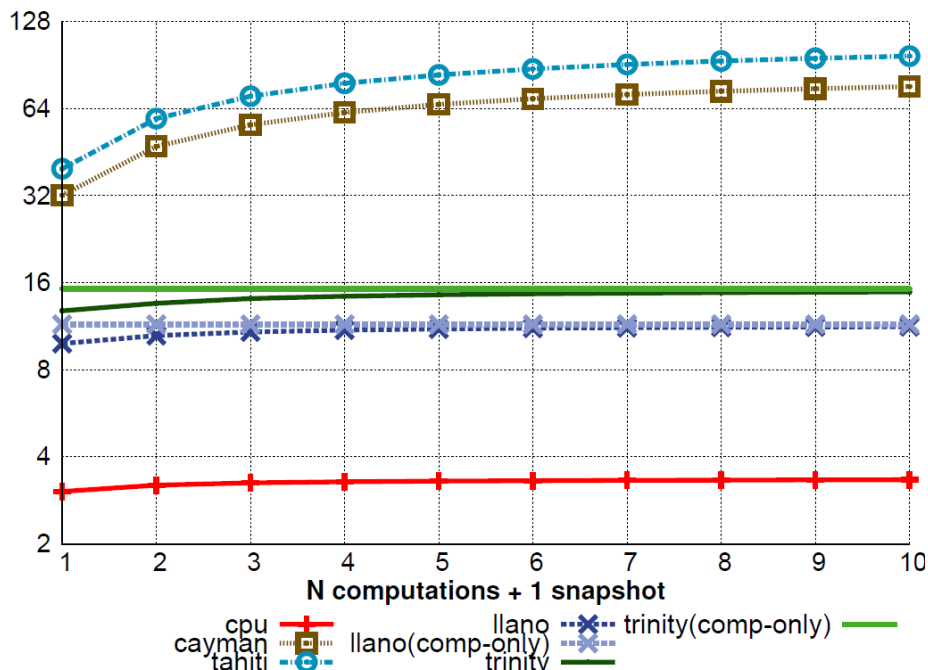
- “v” denotes the **vectorized** version and “lv” denotes the **local vectorized** version.
- The max between “uz” and “cgdc” with “lv” is the most efficient data placement strategy for the wave propagation kernel.

# EXPERIMENTAL RESULTS | CPU/APU/GPU comparison

Domain size: 64x64x64



Domain size: 320x320x320



- “**comp-only**” denotes performance measurements without taking into consideration the cost of data copies between the CPU and the integrated GPU.

# *CONCLUSION*



## CONCLUSION

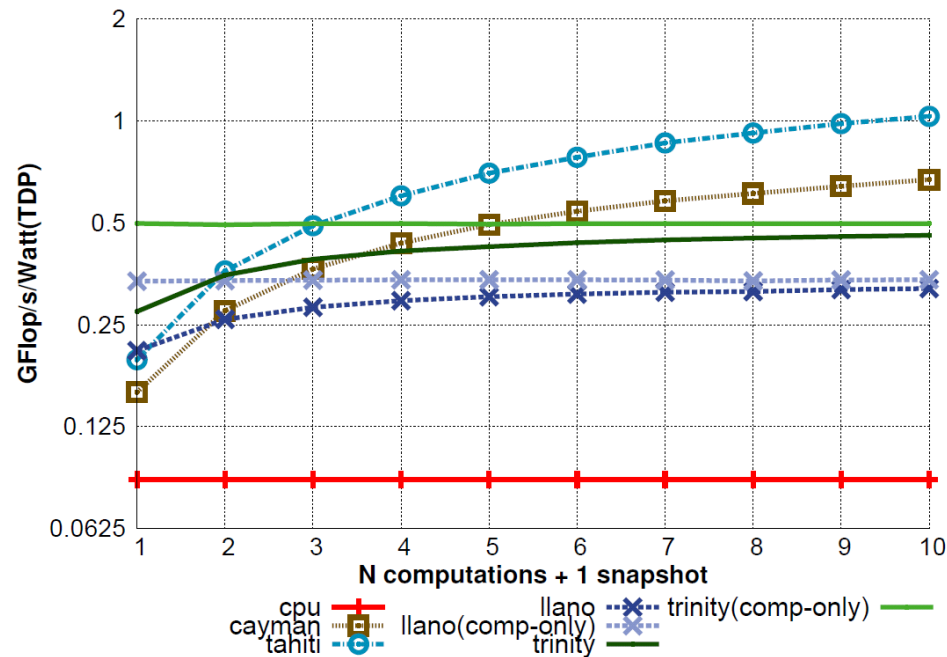
- Considering the computation times only, we obtain good OpenCL performances on CPU, integrated GPU, and discrete GPU.
- Current APUs stencil computations performance cannot outperform discrete GPUs.
- The stencil optimization techniques are not sufficient for the wave modeling kernel (PML damping computations are costly).
- Data placement strategies are a key performance factor for APUs. We will apply the different strategies in the upcoming APUs in order to track their impact on application performance.
- Power consumption should be taken into consideration when comparing CPU/APU/GPU OpenCL performances: see next slide.

### Future work:

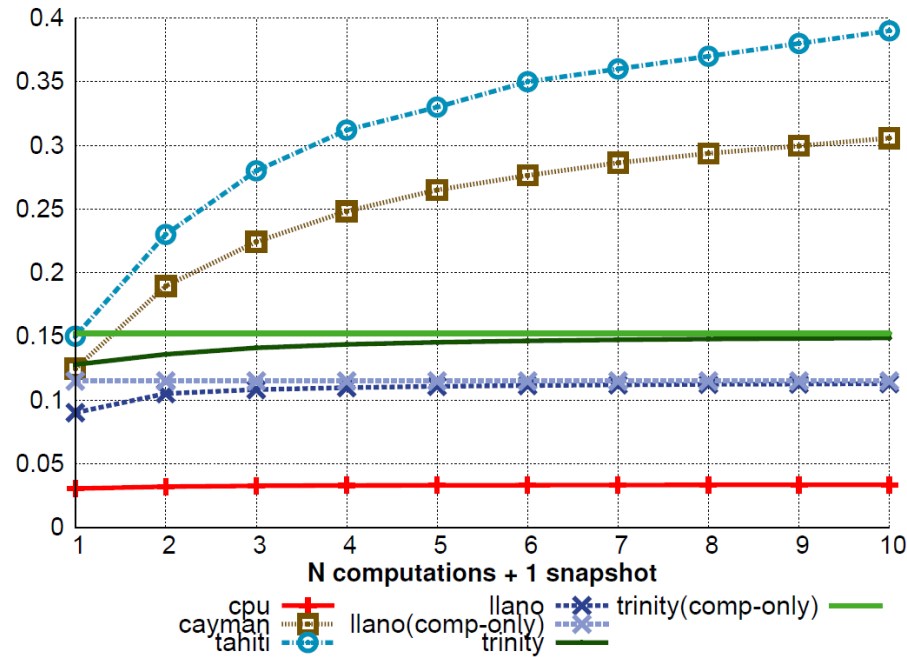
- We will consider APU hybrid implementations.
- We will consider more accurate techniques for measuring power consumption.
- HSA.

# EXPERIMENTAL RESULTS | Performance and power consumption (TDP)

## Stencil kernel, domain size: 320x320x320



## Seismic wave kernel, domain size: 320x320x320



■ We consider the following TDP values: 100W for the CPU and APUs, and 250W for discrete GPUs.

■ APUs outperform discrete GPUs for high frequencies of snapshot retrieval.

## Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

[For AMD-speakers only] © 2012 Advanced Micro Devices, Inc.

[For non-AMD speakers only] The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD's positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.