So if it is not unified memory, what is it?

streamcomputing.eu /blog/2013-11-14/cuda-6-unified-memory-explained/

Blog

14 November 2013

0

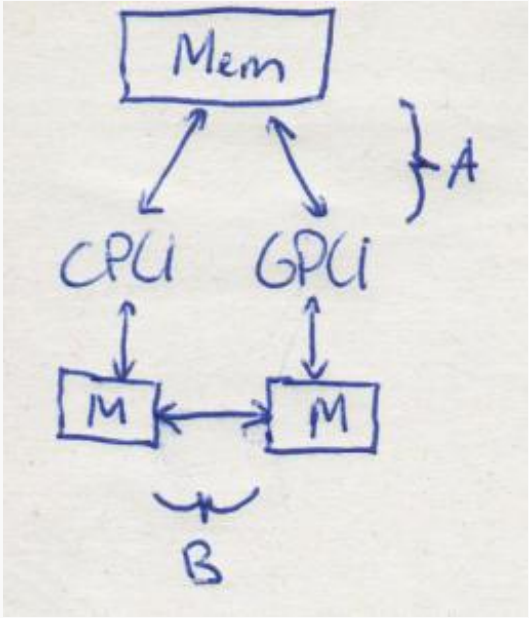
CUDA 6 Unified Memory explained

Vincent Hindriksen | , |

AMD, ARM-vendors and Intel have been busy unifying CPU and GPU memories for years. It is not easy to design a model where 2 (or more) processors can access memory without dead-locking each other.

NVIDIA just announced CUDA 6 and to my surprise includes “Unified Memory”. Am missing something completely, or did they just pass their competitors as it implies one memory? The answer is in their definition:

> ***Unified Memory*** *— Simplifies programming by enabling applications to access CPU and GPU memory without the need to manually copy data from one to the other, and makes it easier to add support for GPU acceleration in a wide range of programming languages.*



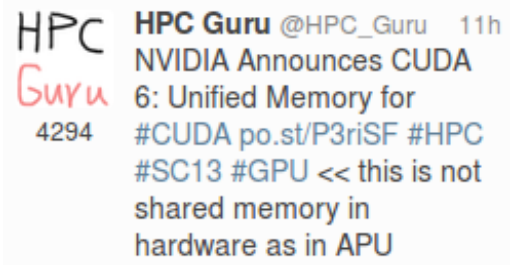
A) Unified Memory Access (UMA). B) NVIDIA’s Unified Virtual Addressing (UVA), now rebranded as “Unified Memory”.

The official definition is:

> ***Unified Memory Access*** *(UMA) is a shared memory architecture used in parallel computers. All the processors in the UMA model share the physical memory uniformly. In a UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data.*

See the difference?

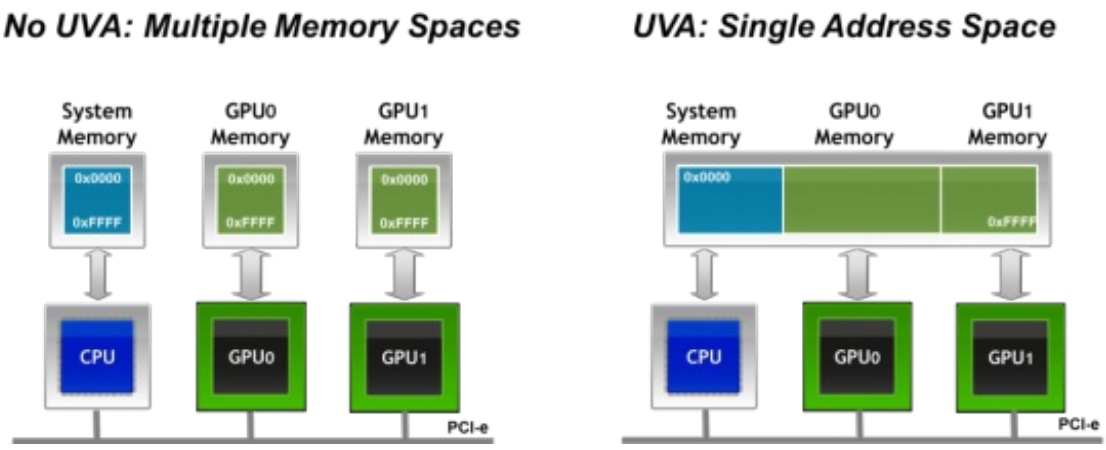
The image at the right explains it differently. A) is how UMA is officially defined, and B is how NVIDIA has redefined it.



So NVIDIA’s Unified Memory solution is engineered by marketeers, not by hardware engineers. On Twitter, I seem not to be the only one who had the need to explain that it is different from the terminology the other hardware-designers have been using.

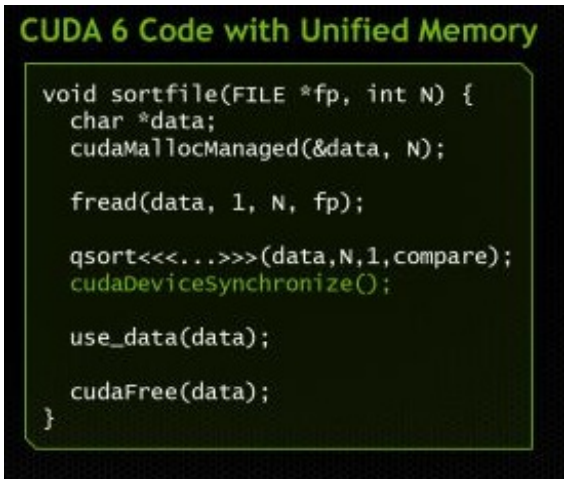
It is intelligent synchronisation between CPU and GPU-memory. The real question is what the

difference is between **Unified Virtual Addressing** (UVA, introduced in CUDA 4) and this new thing.



UVA defines a single Address Space, where CUDA takes care of the synchronisation when the addresses are physically not on the same memory space. The developer has to give ownership to or the CPU or the GPU, so CUDA knows when to sync memories. It does need CudeDeviceSynchronize() to trigger synchronisation (see image).

From **AnandTech**, which wrote about Unified (virtual) Memory:



> *This in turn is intended to make CUDA programming more accessible to wider audiences that may not have been interested in doing their own memory management, or even just freeing up existing CUDA developers from having to do it in the future, speeding up code development.*

So its to attract new developers, and then later taking care of them being bad programmers? I cannot agree, even if it makes GPU-programming popular – I don't bike on highways.

From **Phoronix**, which discussed the changes of NVIDIA Linux driver 331.17 :

> *The new NVIDIA Unified Kernel Memory module is a new kernel module for a Unified Memory feature to be exposed by an upcoming release of NVIDIA's CUDA. The new module is nvidia-uvm.ko and will allow for a unified memory space between the GPU and system RAM.*

So it is **UVM 2.0, but without any API-changes**. That's clear then. It simply matters a lot if it's true or virtual, and I really don't understand why NVIDIA chose to obfuscate these matters.

In OpenCL this has to be done explicitly with mapping and unmapping pinned memory, but is very comparable to what UVM does. I do think UVM is a cleaner API.

Let me know what you think. If you have additional information, I'm happy to add this.