HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Jan Philipp Sachse

Non-Uniform Memory Access (NUMA) Seminar
NUMA and Open-CL

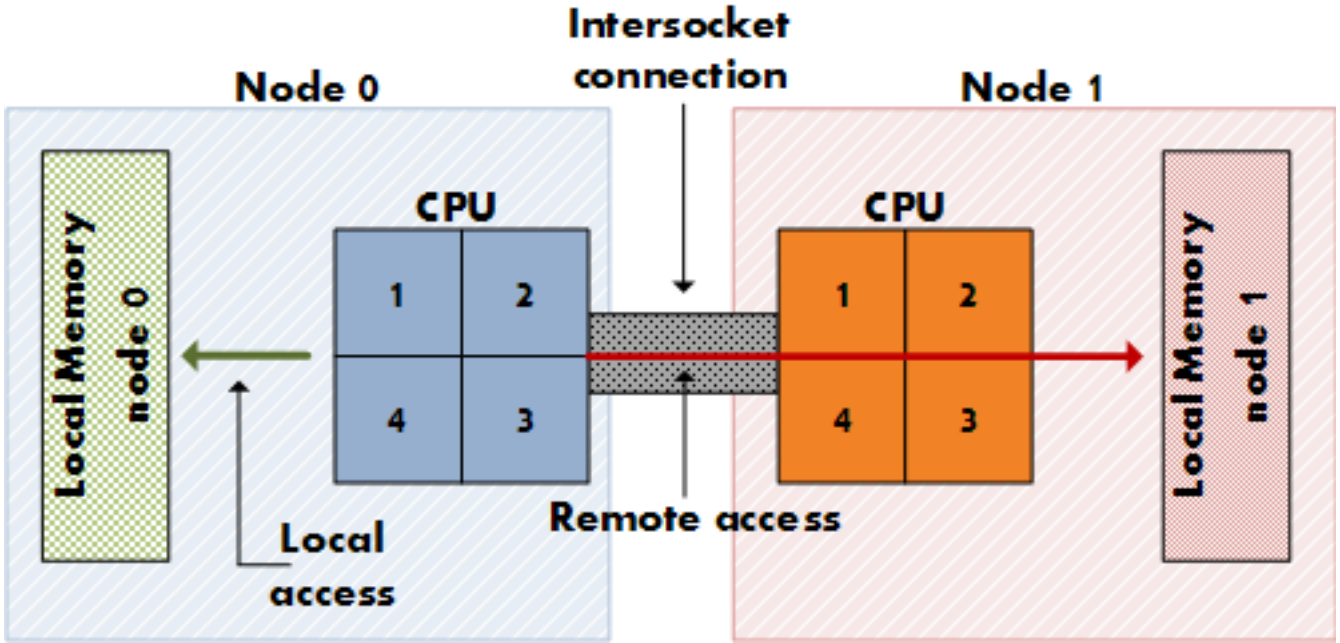
Agenda

1. Recap NUMA
2. Introduction to OpenCL
3. OpenCL 2.0
4. Next-Gen Hardware - NUMA on the Small Scale?
    1. AMD Kaveri
    2. Intel Broadwell
5. NUMA-aware Programming with OpenCL

Agenda

1. **Recap NUMA**
2. Introduction to OpenCL
3. OpenCL 2.0
4. Next-Gen Hardware - NUMA on the Small Scale?
    1. AMD Kaveri
    2. Intel Broadwell
5. NUMA-aware Programming with OpenCL

NUMA



http://i.stack.imgur.com/NB4BF.png

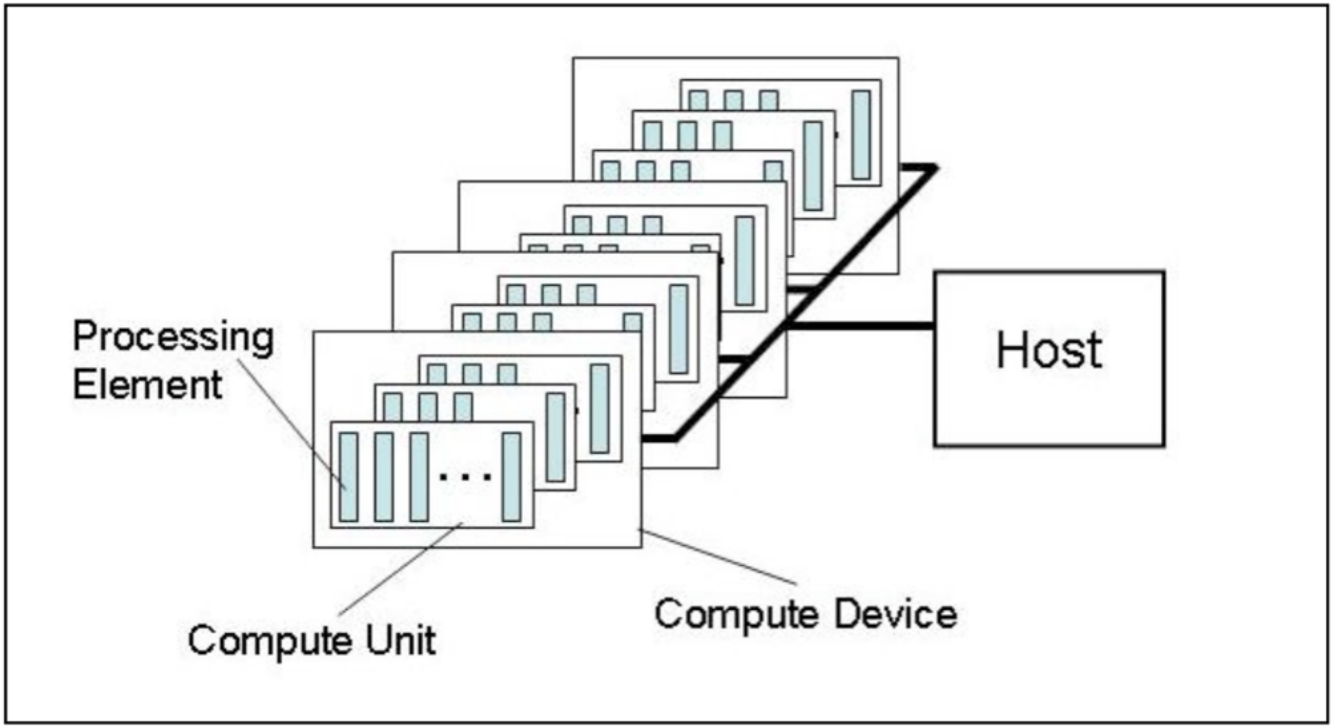
Agenda

1. Recap NUMA
2. **Introduction to OpenCL**
3. OpenCL 2.0
4. Next-Gen Hardware - NUMA on the Small Scale?
    1. AMD Kaveri
    2. Intel Broadwell
5. NUMA-aware Programming with OpenCL

OpenCL

- **Open C**ompute **L**anguage
- Unified programming interface across different computational devices (CPU, GPU, Accelerator, …)
- Initial proposal by Apple (OS X ships with integrated OpenCL support)

OpenCL Vocabulary



Platform

OpenCL Vocabulary

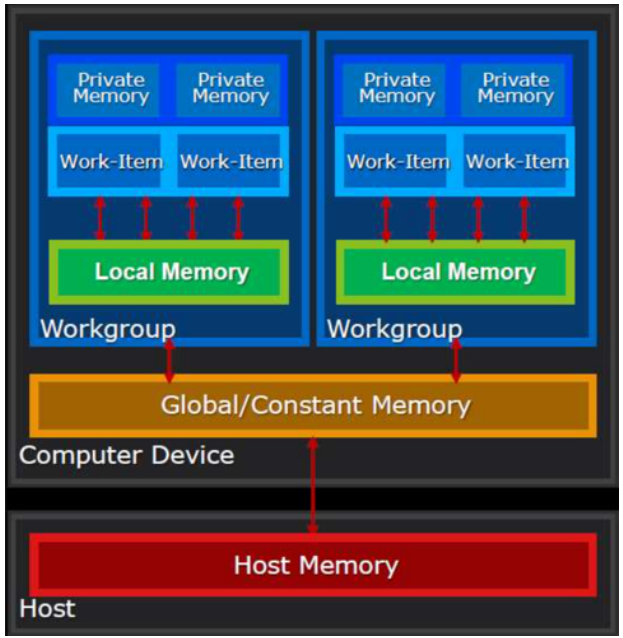
- **Kernel** - execution unit (similar to a c function)
- **Program** - collection of kernels (similar to a dynamic library)
- **Context** - environment within kernels are executed (includes devices, accessible memory, command queue(s))

- **Work Item** - execution of a kernel on one or multiple devices
- **Work Group** - group of work items that execute on a single Compute Unit, Work Items execute the same kernel instance

OpenCL - Memory Model

Memory regions:

- Host memory
- Device memory
  - Global memory
  - Constant memory
  - Local memory
  - Private memory



OpenCL - Memory Model

Memory objects:

- Buffer
  - Block of memory
  - Can contain any of the built in types
- Image
  - Holds one, two or three dimensional images
  - Read/write access from a single kernel (since OpenCL 2.0)
- Pipe (since OpenCL 2.0)
  - Ordered sequence of items
  - Exactly one kernel can connect to either side of the pipe

OpenCL - Memory Model

Memory content management:

- Read/Write/Fill commands
- Map/Unmap commands
- Copy commands
- Shared Virtual Memory (SVM) (since OpenCL 2.0)

Agenda

1. Recap NUMA
2. Introduction to OpenCL
3. **OpenCL 2.0**
4. Next-Gen Hardware - NUMA on the Small Scale?
    1. AMD Kaveri
    2. Intel Broadwell
5. NUMA-aware Programming with OpenCL

OpenCL 2.0 - Improvements

- Pipes
- Images
- Generic Address Space
- C11 Atomics
- Android Installable Client Driver Extension
- Dynamic Parallelism
- Shared Virtual Memory (SVM)

OpenCL 2.0 - Improvements

- Pipes
- Images
- Generic Address Space
- C11 Atomics
- Android Installable Client Driver Extension
- Dynamic Parallelism
- **Shared Virtual Memory (SVM)**

OpenCL 2.0 - Shared Virtual Memory

- Allows kernels to access host memory
- Three types:
  - Coarse-Grained buffer SVM (required)
  - Fine-Grained buffer SVM (optional)
  - Fine-Grained system SVM (optional)

OpenCL 2.0 - Coarse-Grained buffer SVM

- Granularity of regions of OpenCL buffer memory objects
- Consistency enforced:
  - at synchronization points between commands in a queue in a single context
  - with map/unmap commands to drive updates between the host and the device
- Similar to non-SVM, but lets kernel-instances share pointer-based data structures (such as linked-lists) with host program

OpenCL 2.0 - Fine-Grained SVM

- Granularity of individual loads/stores (may be cached)
- Consistency guaranteed at synchronization points
- Host and devices can read the same memory concurrently
- Host and devices can write the same memory concurrently if optional OpenCL SVM atomics supported (otherwise only non-overlapping regions)

Fine Grained Buffer Sharing	Fine Grained System Sharing
Fine-grained sharing within OpenCL buffers	Fine-grained sharing within the entire memory if the host
OpenCL allocation methods needed	malloc()-memory regions also supported
	OpenCL buffers unnecessary

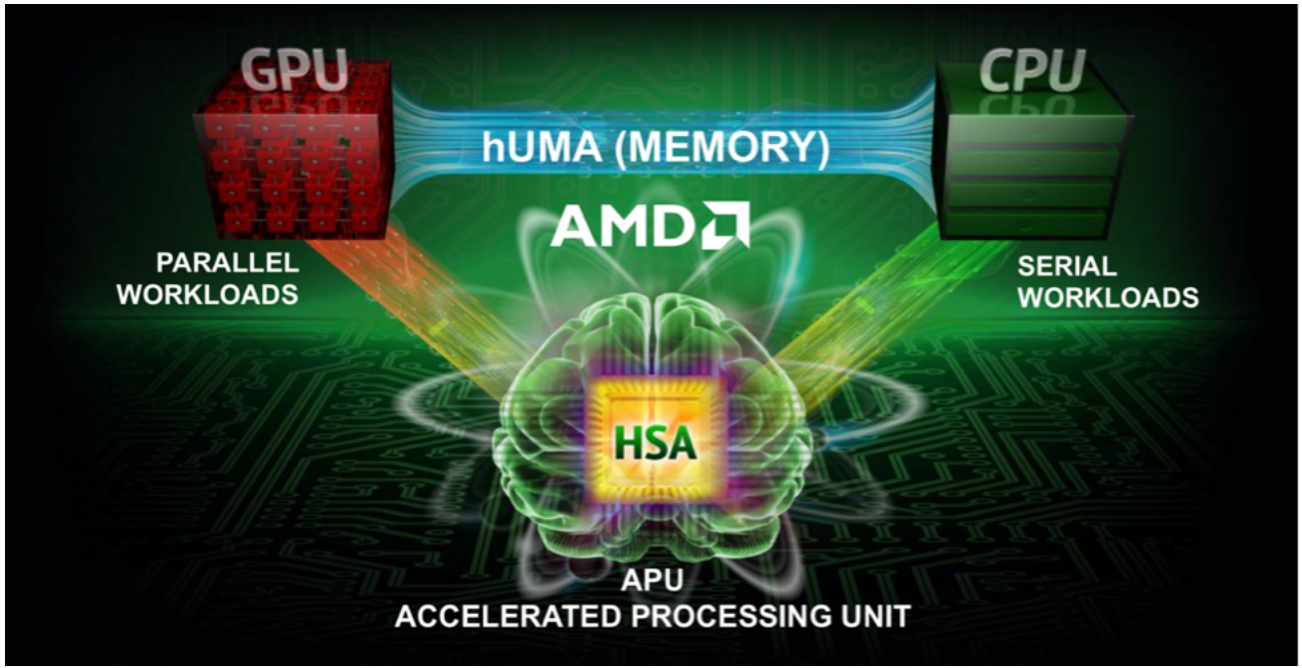
Agenda

1. Recap NUMA
2. Introduction to OpenCL
3. OpenCL 2.0
4. **Next-Gen Hardware - NUMA on the Small Scale?**
    1. AMD Kaveri
    2. Intel Broadwell
5. NUMA-aware Programming with OpenCL

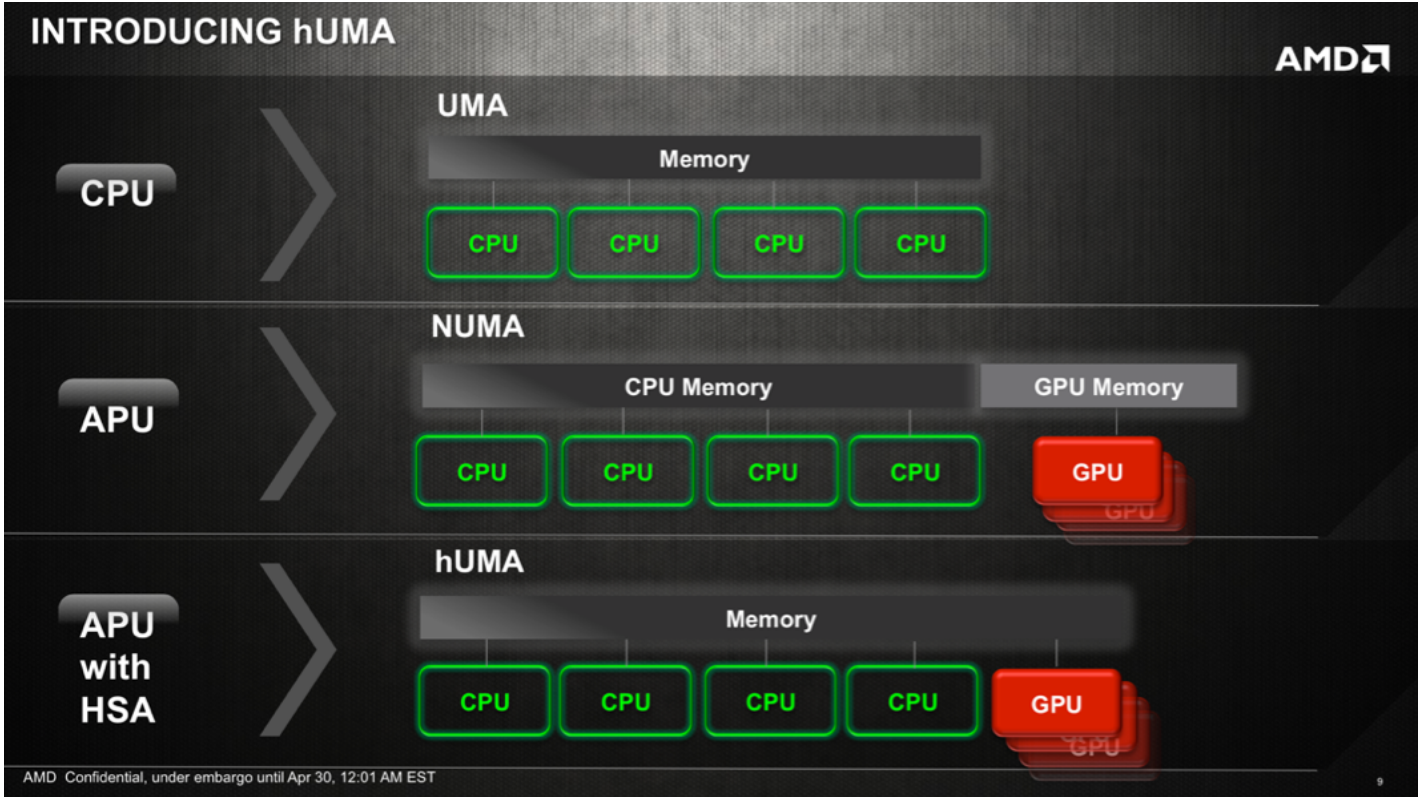
AMD Kaveri

- Accelerated Processing Unit (APU) architecture with memory accessible from CPU and CPU cores
- First architecture that implements OpenCL 2.0

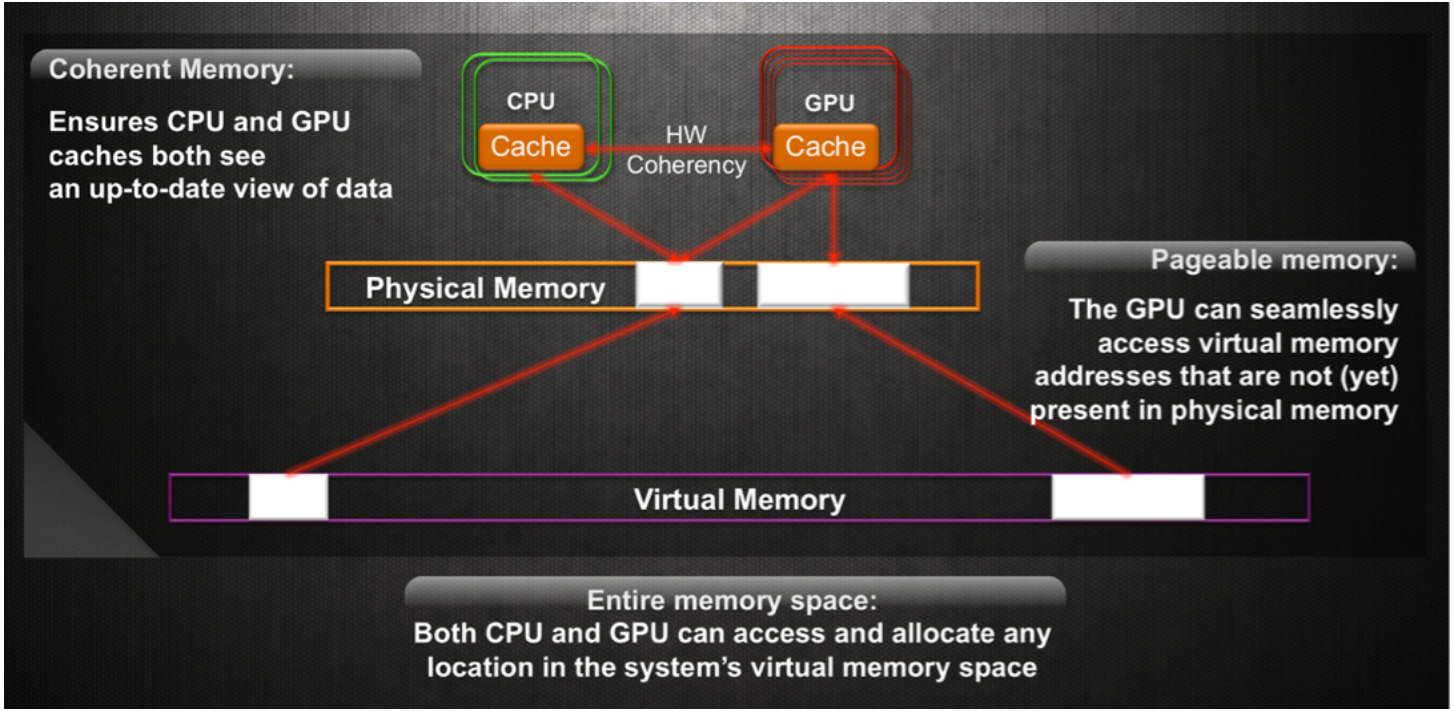
AMD Kaveri - HSA

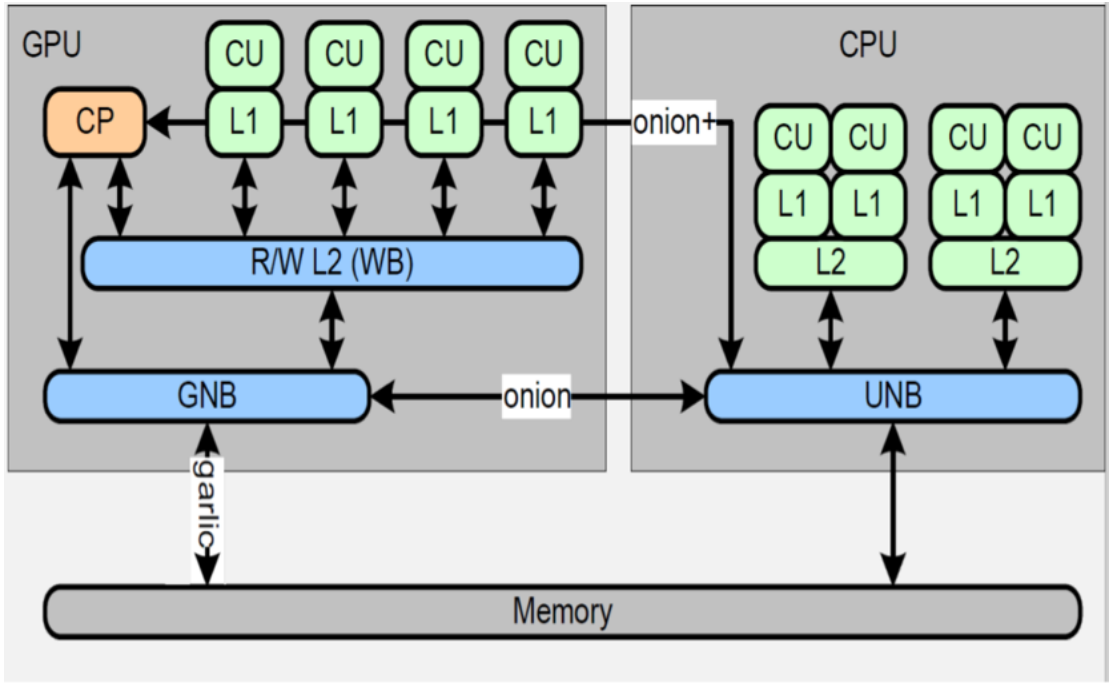


AMD Kaveri - hUMA



AMD Kaveri - hUMA




GPU
CU
CU
CU
CU
CP
L1
L1
L1
L1
onion+
CPU
CU
CU
CU
CU
L1
L1
L1
L1
L2
L2
R/W L2 (WB)
GNB
onion
UNB
garlic
Memory

Intel Broadwell

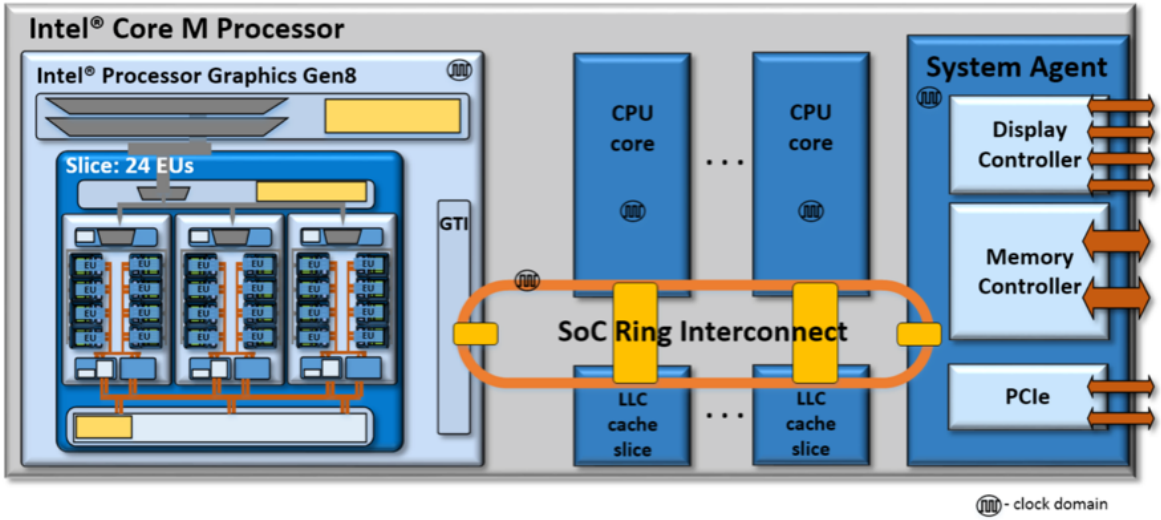
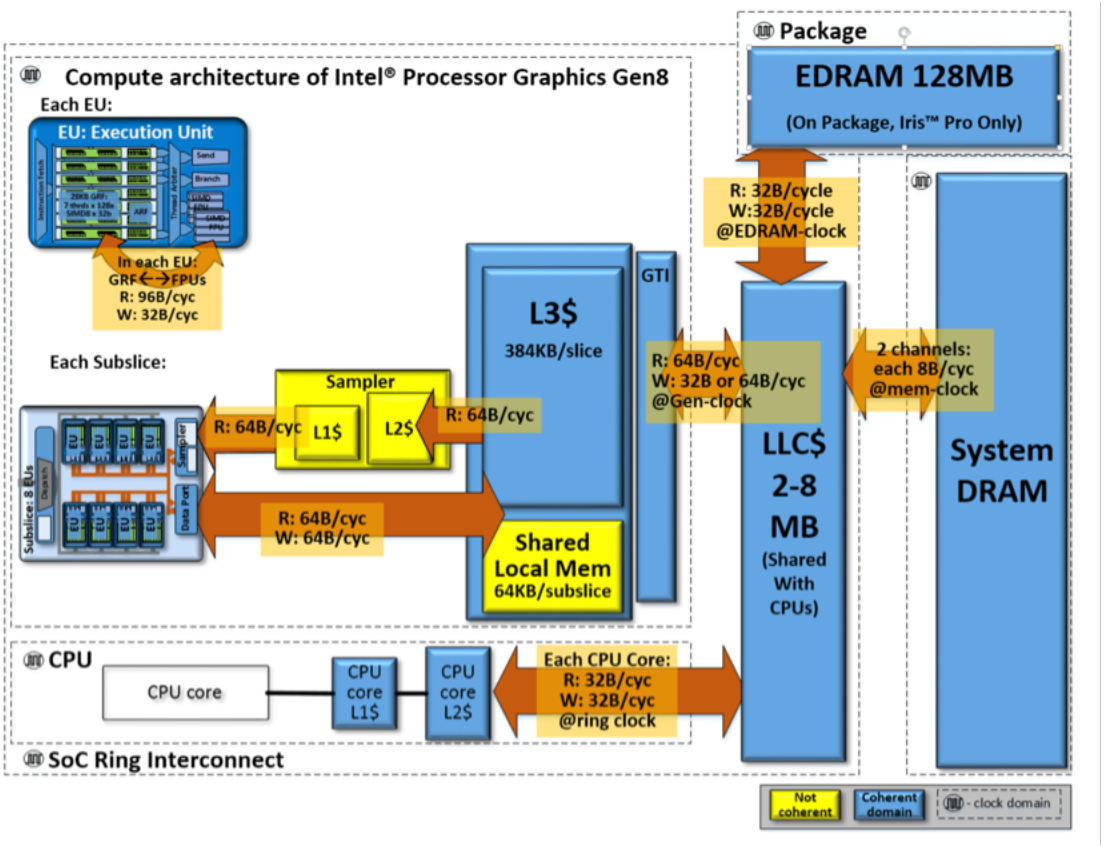


Figure 2: An Intel® Core™ M Processor SoC and its ring interconnect architecture.

Intel Broadwell



NUMA and Open-CL

Jan Philipp Sachse
07.01.2014

Agenda

1. Recap NUMA
2. Introduction to OpenCL
3. OpenCL 2.0
4. Next-Gen Hardware - NUMA on the Small Scale?
    1. AMD Kaveri
    2. Intel Broadwell
5. **NUMA-aware Programming with OpenCL**

OpenCL - Memory Model

When global memory is manipulated by multiple kernel-instances running on multiple devices, the OpenCL runtime system must manage the association of memory objects with a given device. In most cases the OpenCL runtime will implicitly associate a memory object with a device. A kernel instance is naturally associated with the command queue to which the kernel was submitted. Since a command-queue can only access a single device, the queue uniquely defines which device is involved with any given kernel-instance; hence defining a clear association between memory objects, kernel-instances and devices. Programmers may anticipate these associations in their programs and explicitly manage association of memory objects with devices in order to improve performance.

Memory Allocation

Doron Singer (Intel) Tue, 12/24/2013 - 00:28 to post comments

Hello Jianbin,

You can try the following:

1. Allocate memory yourself, using something like libnuma to ensure it's all allocated on a single socket. Make sure to align the memory to the size of the OpenCL data type you intend to use.
2. Create memory objects using CL_MEM_USE_HOST_PTR to wrap these allocations.
3. Use clCreateSubdevices to create sub-devices representing the different NUMA nodes. The current version of the SDK doesn't support partitioning by CL_DEVICE_AFFINITY_DOMAIN_NUMA, but you can use the Intel extension CL_DEVICE_PARTITION_BY_NAMES_INTEL to define which cores to assign to which sub-devices, yourself. Read more about it here:
   http://www.khronos.org/registry/cl/extensions/intel/cl_intel_device_part...
   (http://www.khronos.org/registry/cl/extensions/intel/cl_intel_device_partition_by_names.txt)

That should allow you to enqueue kernels on a single socket using the appropriate sub-device ID, and you can ensure each kernel operates on memory objects allocated on physical pages from that node.

As an aside, the reason there isn't a more straightforward way to go about things is that our testing showing a relatively low return on investment - the performance impact was negligible thanks to the Intel Quick Path Interconnect technology.

If you try this and find a case where this has a significant impact, please let us know.

Thanks,

Doron

Doron Singer (Intel) Tue, 12/24/2013 - 00:28 to post comments

Hello Jianbin,

You can try the following:

1. Allocate memory yourself, using something like libnuma to ensure it's all allocated on a single socket. Make sure to align the memory to the size of the OpenCL data type you intend to use.
2. Create memory objects using CL_MEM_USE_HOST_PTR to wrap these allocations.
3. Use clCreateSubdevices to create sub-devices representing the different NUMA nodes. The current version of the SDK doesn't support partitioning by CL_DEVICE_AFFINITY_DOMAIN_NUMA, but you can use the Intel extension CL_DEVICE_PARTITION_BY_NAMES_INTEL to define which cores to assign to which sub-devices, yourself. Read more about it here:
   http://www.khronos.org/registry/cl/extensions/intel/cl_intel_device_part...
   (http://www.khronos.org/registry/cl/extensions/intel/cl_intel_device_partition_by_names.txt)

That should allow you to enqueue kernels on a single socket using the appropriate sub-device ID, and you can ensure each kernel operates on memory objects allocated on physical pages from that node.

As an aside, the reason there isn't a more straightforward way to go about things is that our testing showing a relatively low return on investment - the performance impact was negligible thanks to the Intel Quick Path Interconnect technology.

If you try this and find a case where this has a significant impact, please let us know.

Thanks,

Doron

NUMA-aware programming using OpenCL

- **Is it possible?**
  - Kind of
  - (but it's complicated)

- It is possible to:
  - get single devices and split them into subdevices using OpenCL (>= 1.2 or 1.1 with `cl_ext_device_fission` (Intel))
  - allocate memory at a specific location using C and then wrap this memory into OpenCL-Objects so that OpenCL can use it (`CL_MEM_USE_HOST_PTR`)

- Missing link between libnuma node ids and OpenCL device ids

OpenCL Functions to get hierarchy information

- `clGetPlatformIDs()` - get all available platforms
- `clGetDeviceIDs()` - get all available devices for a platform
- `clGetDeviceInfo()` - get information about a single device
  - `CL_DEVICE_NAME`
  - `CL_DEVICE_PARTITION_PROPERTIES`
  - `CL_DEVICE_PARTITION_AFFINITY_DOMAIN`

OpenCL Functions to specify execution devices

Create a subdevice(s):

```
cl_int clCreateSubDevices (
            cl_device_id in_device,
            const cl_device_partition_property *properties,
            cl_uint num_devices,
            cl_device_id *out_devices,
            cl_uint *num_devices_ret)
```

OpenCL Functions to specify execution devices

- **OpenCL standard:**
  - `CL_DEVICE_PARTITION_EQUALLY`
  - `CL_DEVICE_PARTITION_BY_COUNTS`
  - `CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN`
    - `CL_DEVICE_AFFINITY_DOMAIN_NUMA`
    - `CL_DEVICE_AFFINITY_DOMAIN_L4_CACHE`
    - `CL_DEVICE_AFFINITY_DOMAIN_L3_CACHE`
    - `CL_DEVICE_AFFINITY_DOMAIN_L2_CACHE`
    - `CL_DEVICE_AFFINITY_DOMAIN_L1_CACHE`
    - `CL_DEVICE_AFFINITY_DOMAIN_NEXT_PARTITIONABLE`

- **Supported by Intel:**
  - `CL_DEVICE_PARTITION_EQUALLY`
  - `CL_DEVICE_PARTITION_BY_COUNTS`
  - `CL_DEVICE_PARTITION_BY_NAMES_INTEL`

Demo

http://dilbert.com/dyn/str_strip/000000000/00000000/0000000/000000/00000/6000/600/6693/6693.strip.gif

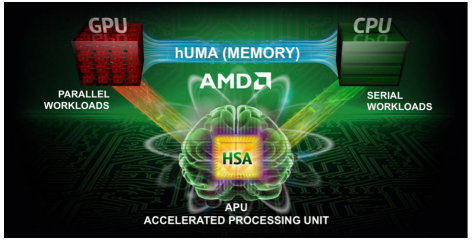
Notes

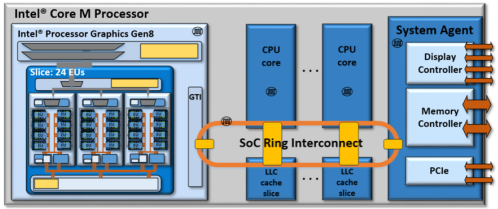
- **Problems:**
  - No working way to easily install Intel OpenCL driver on Ubuntu
  - Neither AMD nor Intel support affinity domains
  - Even though subdevices are created, threads are scheduled on multiple cores

- **Things that would be great:**
  - A simple possibility to tell OpenCL where to allocate memory
  - At least: a fully working implementation of the OpenCL standard

Overview

- Introduction to OpenCL / OpenCL 2.0
  - OpenCL Memory Model
  - Shared Virtual Memory
- Next-Gen Hardware - NUMA on the Small Scale?
  - AMD Kaveri
    - HSA
    - hUMA
  - Intel Broadwell
    - SoC Ring Interconnect
    - EDRAM
- NUMA-aware Programming with OpenCL
  - Theoretically possible
  - Affinity Domains currently not supported by AMD and Intel
  - Kernels jump from core to core (Intel)





Sources

- Khronos OpenCL Working Group. "OpenCL 2.0 Specification." Khronos Group, Nov (2013)
- Stephen Junkins. "The Compute Architecture of Intel® Processor Graphics Gen8" Intel Corporation, (2014)
- Advanced Micro Devices, Inc. "THE REVOLUTION GOES MOBILE AMD 2014 PERFORMANCE MOBILE APUS" Advanced Micro Devices, Inc., (2014)
- Phil Rogers, Joe Macri, Sasa Marinkovic. "AMD heterogeneous Uniform Memory Access" Advanced Micro Devices, Inc., (2013)
- Benedict R. Gaster. "OpenCL Device Fission" Khronos Group, Mar (2011)
- https://www.khronos.org/registry/cl/extensions/ext/cl_ext_device_fission.txt (29.12.2014)
- https://www.khronos.org/registry/cl/extensions/intel/cl_intel_device_partition_by_names.txt (29.12.2014)
- Khronos Releases OpenCL 2.0 - Khronos Group Press Release (https://www.khronos.org/news/press/khronos-releases-opencl-2.0) (29.12.2014)
- https://software.intel.com/en-us/forums/topic/497429
- OpenCL* Device Fission for CPU Performance (https://software.intel.com/en-us/articles/opencl-device-fission-for-cpu-performance) (29.12.2014)
- Neil Trevet. "OpenCL Introduction" Khronos Group, (2013) (https://www.khronos.org/assets/uploads/developers/library/overview/opencl_overview.pdf) (29.12.2014)
- "Introduction and Overview" Khronos Group, June (2010) (https://www.khronos.org/assets/uploads/developers/library/overview/OpenCL-Overview-Jun10.pdf) (29.12.2014)