CPU-GPU Heterogeneous Computing

Advanced Seminar "Computer Engineering"
Winter-Term 2015/16

Steffen Lammel
December 2, 2015

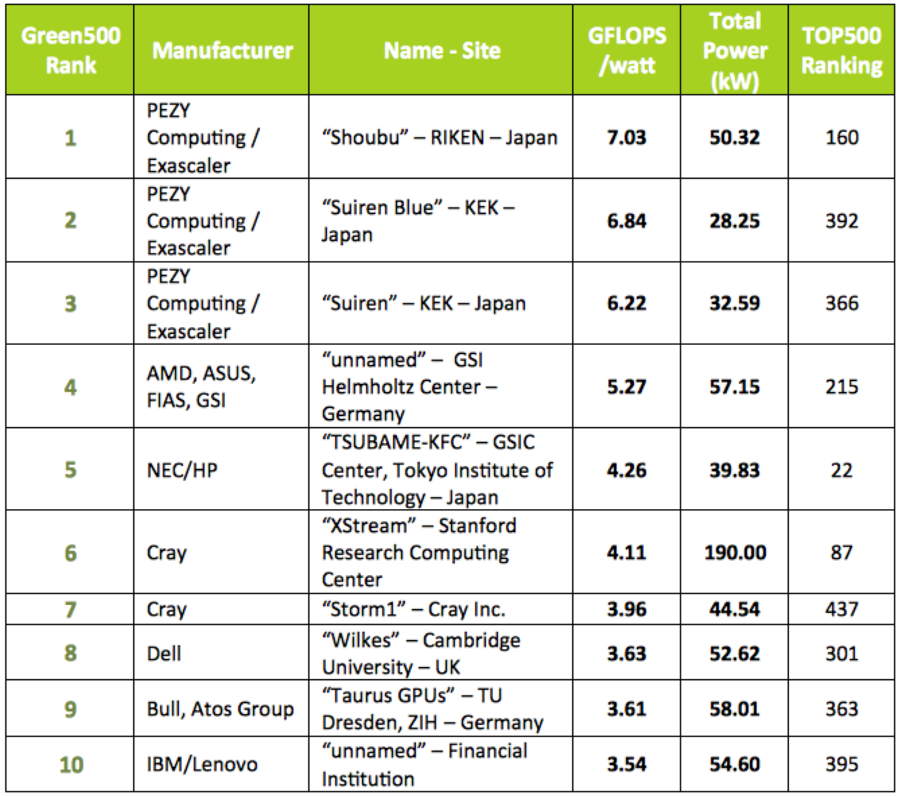
Content

Introduction

Introduction

Source: [2]

- **Grand Goal in HPC**
  - Exascale systems until the year ~2020
- **Problems**
  - Computational Power
    - Now: up to 7GF/W
    - Exascale: >=50GF/W
  - Power Budget ~20MW
  - Heat Dissipation



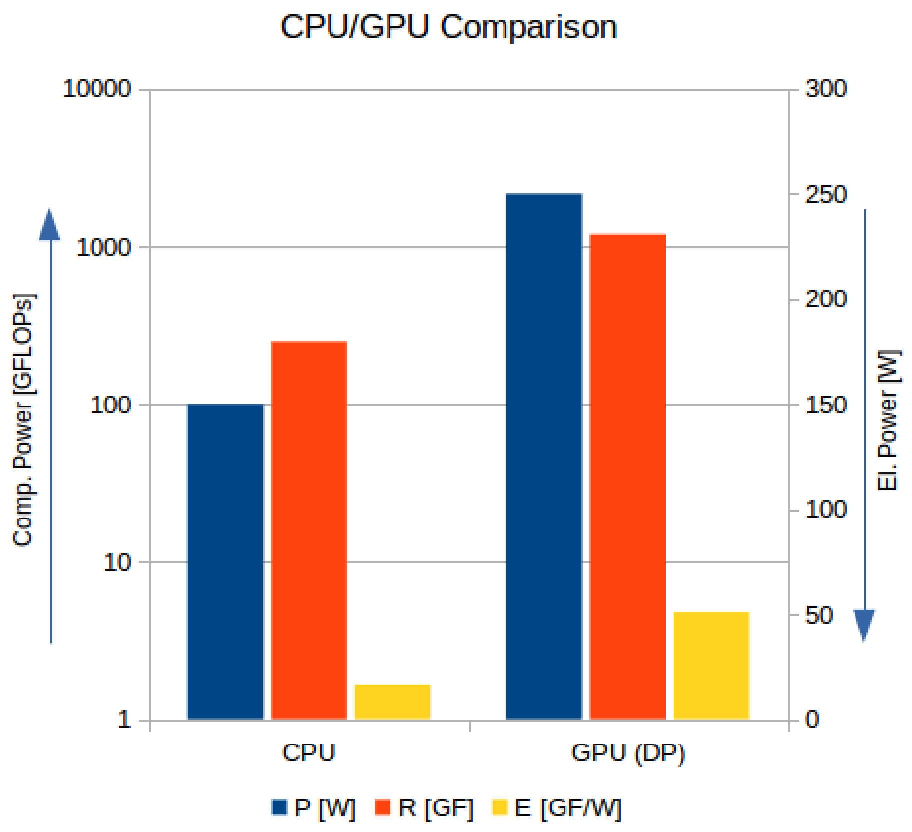
Green500 Rank	Manufacturer	Name - Site	GFLOPS /watt	Total Power (kW)	TOP500 Ranking
1	PEZY Computing / Exascaler	"Shoubu" – RIKEN – Japan	7.03	50.32	160
2	PEZY Computing / Exascaler	"Suiren Blue" – KEK – Japan	6.84	28.25	392
3	PEZY Computing / Exascaler	"Suiren" – KEK – Japan	6.22	32.59	366
4	AMD, ASUS, FIAS, GSI	"unnamed" – GSI Helmholtz Center – Germany	5.27	57.15	215
5	NEC/HP	"TSUBAME-KFC" – GSIC Center, Tokyo Institute of Technology – Japan	4.26	39.83	22
6	Cray	"XStream" – Stanford Research Computing Center	4.11	190.00	87
7	Cray	"Storm1" – Cray Inc.	3.96	44.54	437
8	Dell	"Wilkes" – Cambridge University – UK	3.63	52.62	301
9	Bull, Atos Group	"Taurus GPUs" – TU Dresden, ZIH – Germany	3.61	58.01	363
10	IBM/Lenovo	"unnamed" – Financial Institution	3.54	54.60	395

Source: [1]

Compare: #1 TOP500: ~33PF @ 1,9GF/W

Introduction

- **CPU**
  - Few cores (<= 20)
  - High frequency (~3GHz)
  - Large caches, plenty of (slow) memory (<= 1TB)
  - Latency oriented
- **GPU**
  - Many cores (> 1000)
  - Slow frequency (<=1GHz)
  - Fast memory, limited in size (<= 12GB)
  - Throughput oriented



Introduction

- **Ways increase Energy Efficiency:**
  - Get the most computational power from both domains
  - Utilize the sophisticated power-saving techniques modern CPU/GPUs offer

Terminology:

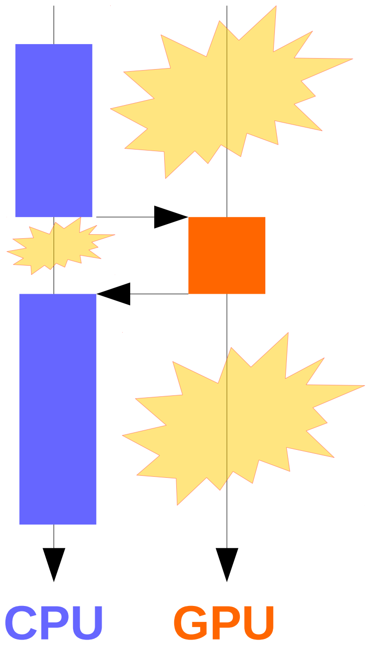
- **HCS**: Heterogeneous Computing System (hardware)
- **HCT**: Heterogeneous Computing Technique (software)
- **PU**: Processing Unit (can be both, CPU and GPU)
- **FLOPs**: Floating Point Operations per second
  - **DP**: Double Precision
  - **SP**: Single Precision
- **BLAS**: Basic Linear Algebra Subprograms
- **SIMD**: Single Instruction Multiple Data

Heterogeneous Computing Techniques (HCT)

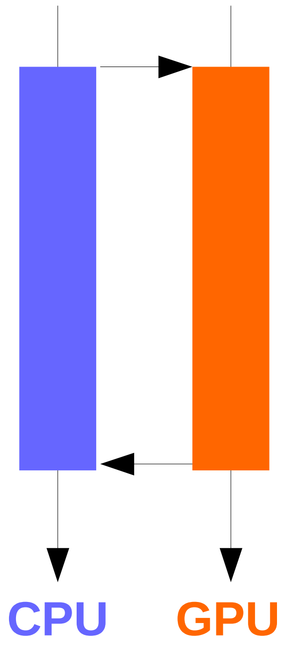
Runtime Level

HCT - Basics

- **Worst case:**
  - Only one PU is active at a time



- **Ideal case:**
  - All PUs do (useful) work simultaneously

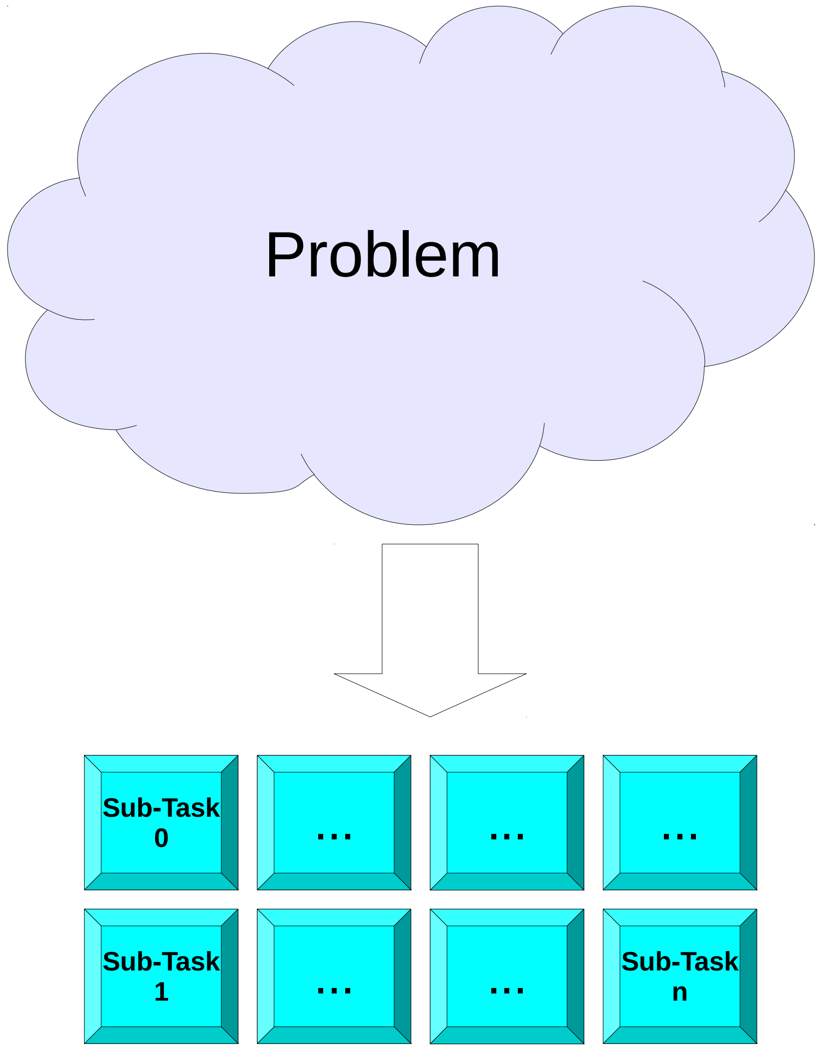


HCT - Basics

- **Examples are idealized**
  - Real world applications consist of several different patterns
- **Typical Processing Units (PU) in HCS**
  - Tens of CPU cores/threads
  - Several 1000 GPU cores/kernels
- **Goals of HCT**
  - All PUs have to be utilized (in a useful way)

HCT – Workload Division

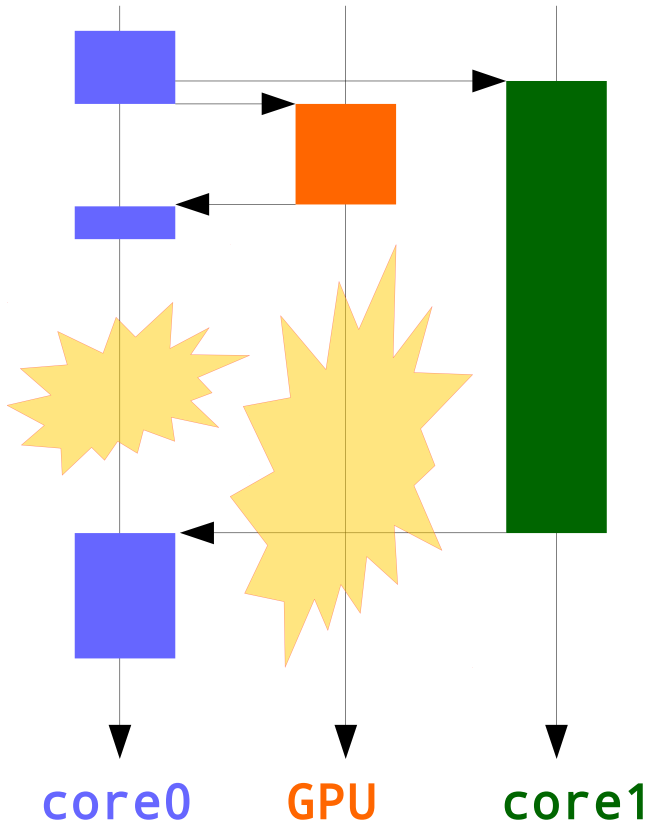
- **Basic Idea:**
  - Divide the whole problem into smaller chunks
  - Assign each sub-task to a PU
  - compare: “PCAM”, [5]
    - Partition
    - Communicate
    - Agglomerate
    - Map



HCT – Workload Division
(naive)

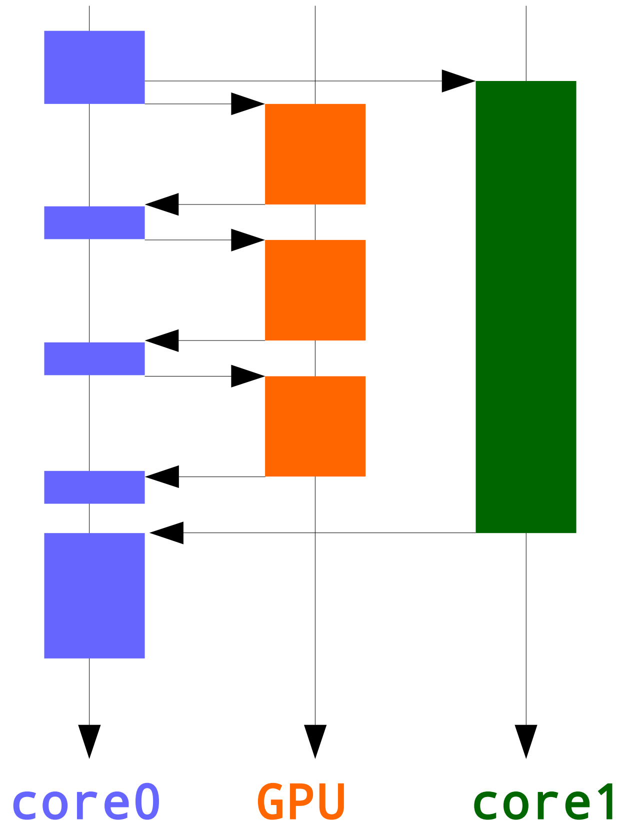
Example:

- **Dual-Core System**
  - CPU + GPU
  - Naive data distribution
- **CPU core 0**
  - Master/Arbiter
- **CPU core 1**
  - Worker
- **GPU**
  - Worker
- **Huge idle periods for GPU and CPU core 0**



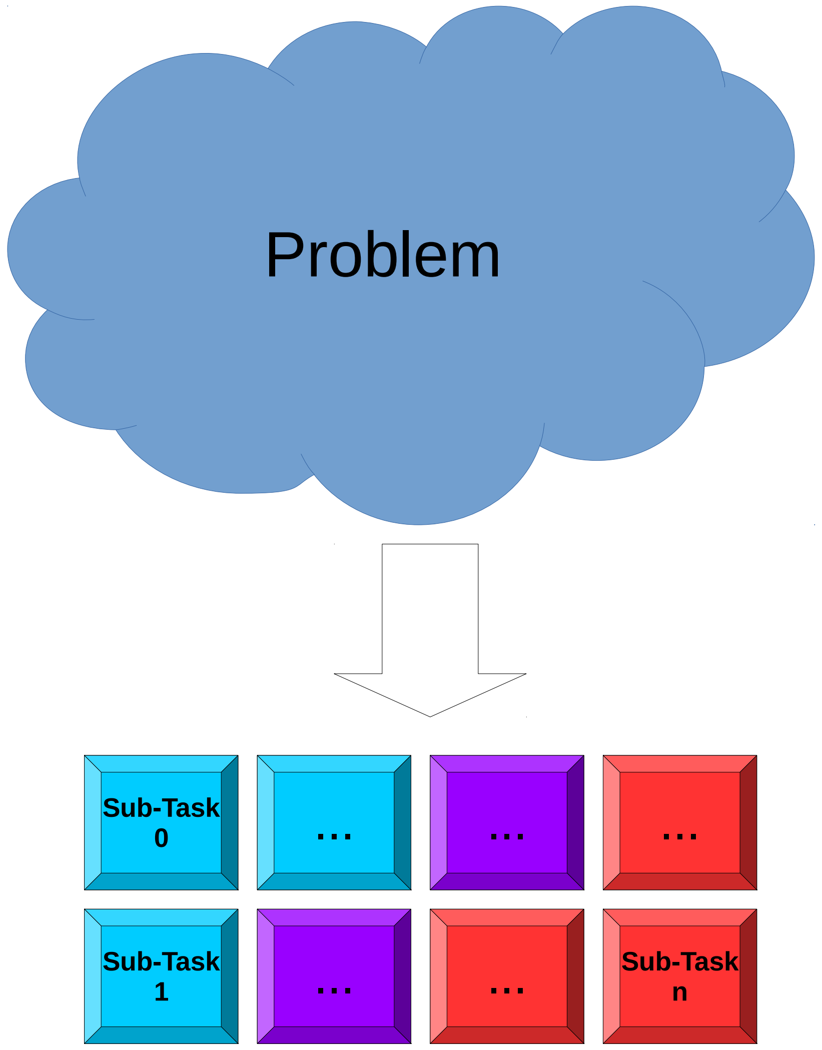
HCT – Workload Division
(relative PU performance)

- **Approach: use relative performance of each PU as metric**
  - A microbenchmark or performance model deemed the GPU 3x faster than than the CPU
  - Partition the work in a 3:1 ratio to the PUs
  - Task granularity and the quality/nature of the microbenchmark are the key factors here



HCT – Workload Division
(characteristics of sub-tasks)

- **Idea:**
  - Use the nature of the sub-tasks to leverage performance
  - **CPU affine** tasks
  - **GPU affine** tasks
  - tasks which run roughly **equally well on all PUs**

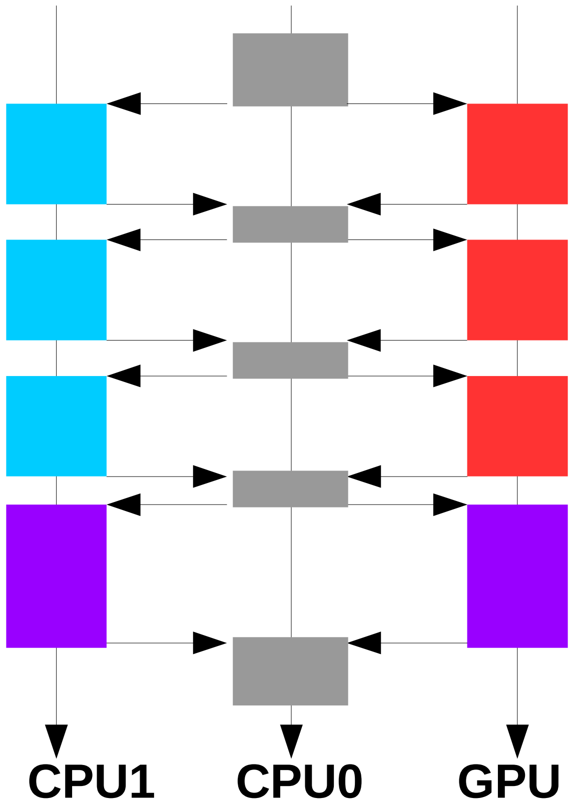


HCT – Workload Division

(nature of sub-tasks)

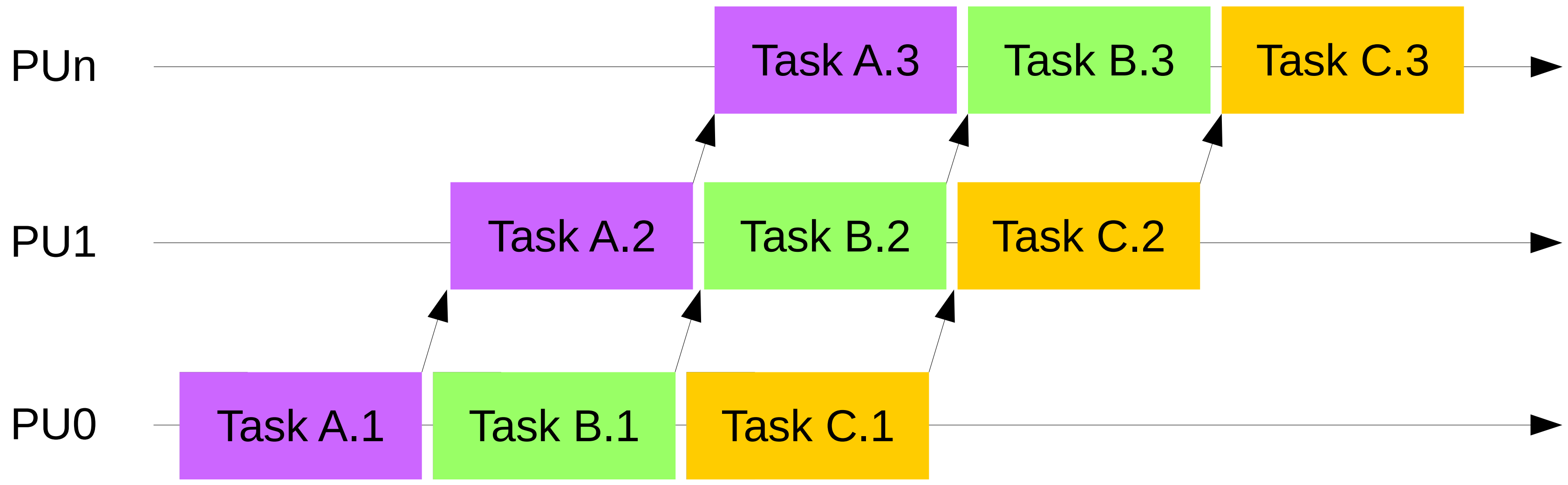
- **Map the tasks to the PU it performs best on**
  - Latency: CPU
  - Throughput: GPU

- **Further scheduling metrics:**
  - Capability (of the PU)
  - Locality (of the data)
  - Criticality (of the task)
  - Availability (of the PU)



HCT – Workload Division
(pipeline)

- **If overlap is possible: Pipeline**
  - Call kernels asynchronously to hide latency
  - Small penalty to fill and drain the pipeline
  - Good utilization of all PUs if the pipeline is full



HCT – Workload Division
(relative PU performance)

Summary: Metrics for workload division

- **Performance of PUs**
- **Nature of sub-tasks**
  - Order
    - Regular Patterns --> GPU (little communication)
    - Irregular Patterns --> CPU (lots of communication)
  - Memory Footprint
    - Fits into VRAM? --> GPU
    - Too Big? --> CPU
  - BLAS-Level
    - BLAS-1/2 --> CPU (Vector-Vector. Vector-Matrix operations)
    - BLAS-3 --> GPU (Matrix-Matrix operations

- **Historical Data**
  - How well did each PU perform in the previous step?
- **Availability of PU**
  - Is there a function/kernel for the desired PU?
  - Is the PU able to take a task (scheduling-wise)?

Heterogeneous Computing Techniques (HCT)

Frameworks and tools

HCT – Framework Support

- **Implementing these techniques is tedious and error-prone**
  - Better: Let a framework do this job!
- **Framework for load-balancing**
  - Compile-Time Level (static scheduling)
  - Runtime Level (dynamic scheduling)
- **Framework for parallel-abstraction**
  - Write the algorithm as a sequential program and let the tools figure out how to utilize the PUs optimally
  - Sourcecode annotations to give the run-time/compiler hints what approach is the best (comp.: OpenMP #pragma_omp_xxx)
  - Scheduling: dynamic or static
- **Partitioning and work-division principles shown before apply here as well!**

HCT – Framework Support

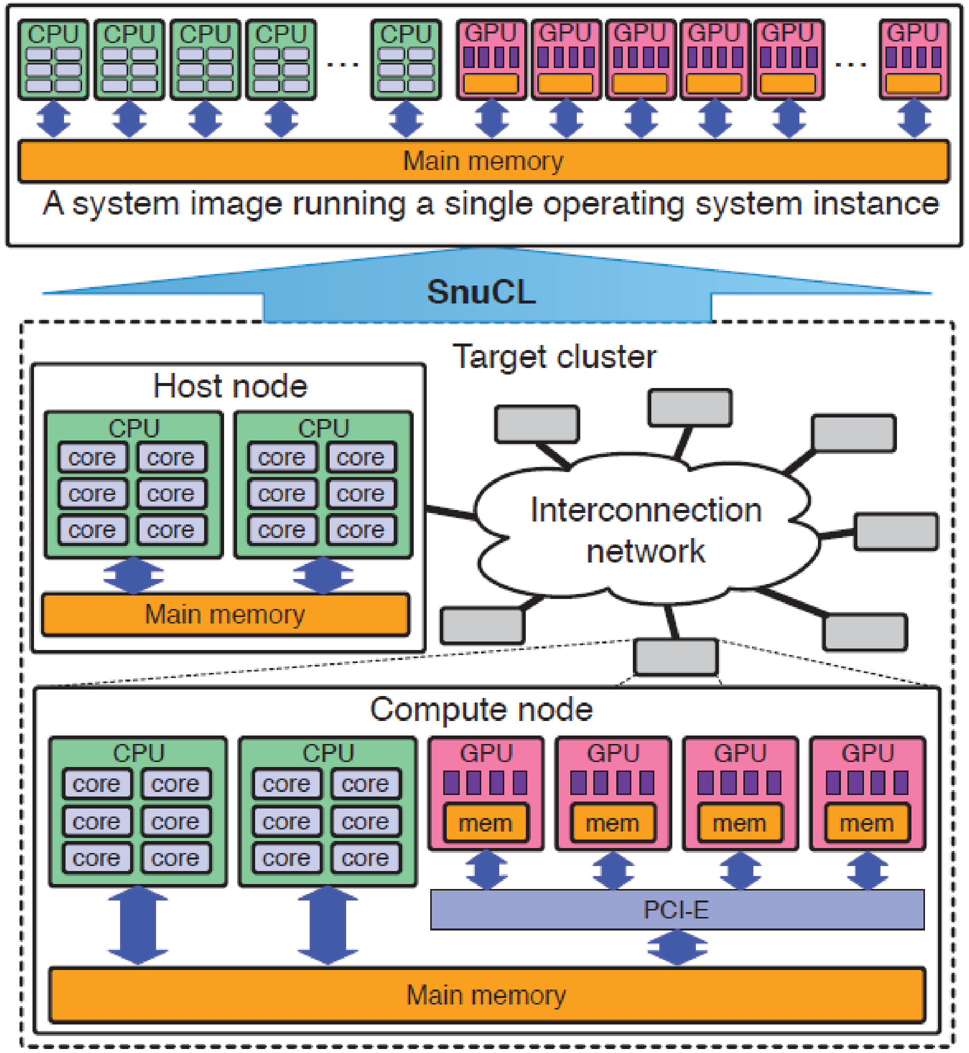
- **Generic PU specific tools and frameworks**
  - CUDA+Libraries (Nvidia GPU)
  - OpenMP, Pthreads (CPU)

- **Generic heterogenous-aware frameworks**
  - OpenCL, OpenACC
  - OpenMP (“offloading”, since v4.0)
  - CUDA (CPU-callback)

- **Custom Frameworks (interesting Examples)**
  - Compile-Time Level (static scheduling approach)
    - SnuCL
  - Run-Time Level (dynamic scheduling approach)
    - PLASMA

HCT – Framework Support
(Example: SnuCL)

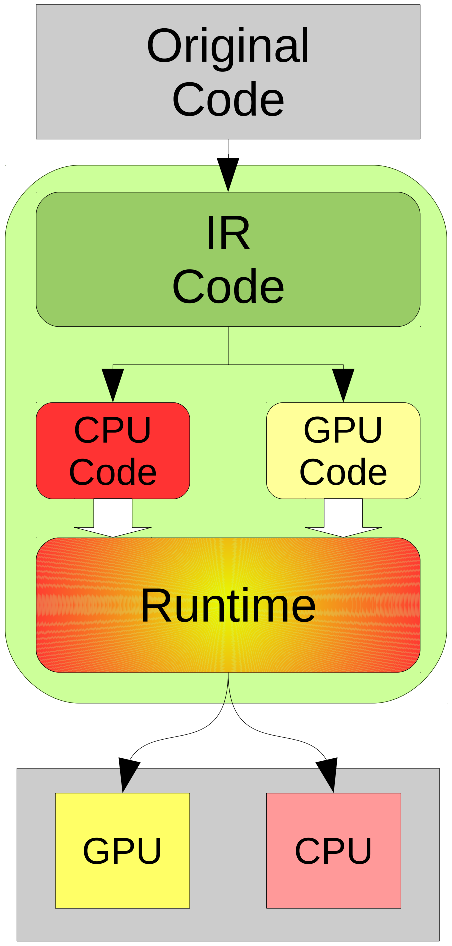
- **Creates a "virtual node" with all the PUs of a Cluster**
- **Use a message passing interface (MPI) to distribute the workloads to the distant PUs**
- **Inter-Node communication is implicit**



Source: [4]

HCT – Framework Support
(Example: PLASMA)

- **Intermediate representation**
  - Independent of PU
- **PU-specific implementation based on IR**
  - Utilizes the PU's specific SIMD capabilities
- **A Runtime decides assignment to PU dynamically**
  - Speed-Up depends on workload



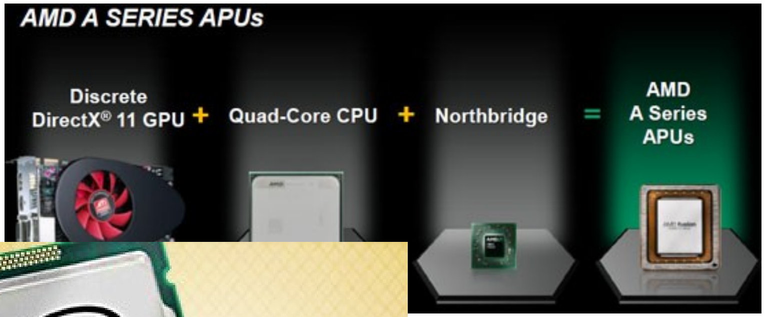
Heterogeneous Computing Techniques (HCT)

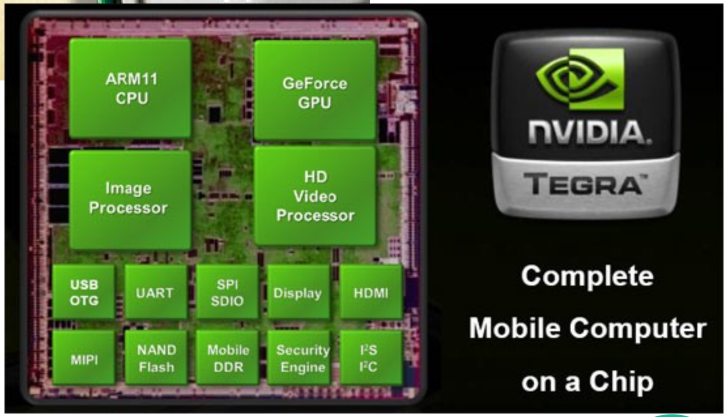
Fused HCS

HTC – Fused HCS

- **CPU and GPU share the same die**
  - … and the same address space!
- **Communication paths are significantly shorter**

- **AMD “Fusion” APU**
  - x86 + OpenCL
- **Intel “Sandy Bridge” and successors**
  - x86 + OpenCL
- **Nvidia “Tegra”**
  - ARM + CUDA





Energy saving with HCS

Energy saving with HCS

- **Trade-off**
  - Performance vs. energy consumption
- **Modern PUs are delivered with extensive power-saving features**
  - e.g.: Power Regions, Clock Gating
- **Less aggressive energy saving in HPC**
  - Reason: get rid of state transition penalties
- **Aggressive ES in mobile/embedded**
  - Battery-life is everything in this domain

Energy saving with HCS
(hardware: DVFS)

- **Dynamic Voltage/Frequency Scaling (DVFS)**
  - $P = C*V^2*f$
    - with $f \sim V$; C=const.
  - Reduce f by 20%
    - → P: -50%!
  - How far can we lower f/V to meet our timing constraints?

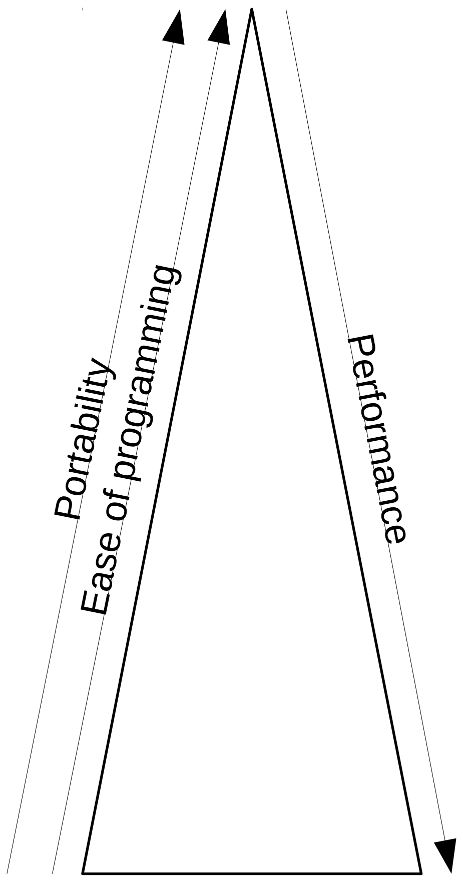
Energy saving with HCS
(software: intelligent work distribution)

- **Intelligent Workload Partitioning**
  - Power model of tasks and PU
  - Assign tasks with respect to power model
  - Take communication overhead into account

Conclusion

Conclusion
(programming aspects)

- **Trade off**
  - Usability vs. performance
  - Portability vs. performance
- **High Performance requires high programming effort**
  - “Raw” CUDA/OpenMP
- **Code-abstracting frameworks can support developers to a certain degree**
  - OpenCL, OpenACC, custom solutions
- **Dynamic scheduling frameworks can accelerate an application**
  - Complicated cases, many PUs



Conclusion
(energy aspects)

- **HCS do contribute to a better performance/watt ratio**
- **Intelligent workload partitioning**
  - equally important for performance and energy saving
- **DVFS is a key technique for energy-saving**
- **Fused HCS can fill nichès where communication is a key factor**

Thank you for your attention!

Questions?

References

Papers:
A Survey of CPU-GPU Heterogeneous Computing Techniques (Link)
SnuCL: an OpenCL Framework for Heterogeneous CPU/GPU Clusters (Link)
PLASMA: Portable Programming for SIMD Heterogeneous Accelerators (Link)

Images:
[1] http://www.hpcwire.com/2015/08/04/japan-takes-top-three-spots-on-green500-list/
[2] http://www.top500.org/lists/2015/11/
[3] https://en.wikipedia.org/wiki/Performance_per_watt#FLOPS_per_watt
[4] http://snucl.snu.ac.kr/features.html
[5] http://www.mcs.anl.gov/~itf/dbpp/text/node15.html