

# Lecture 17: Heterogeneous Parallel Programming

Vivek Kumar

Computer Science and Engineering

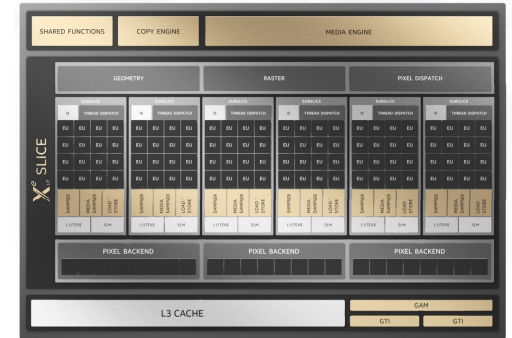
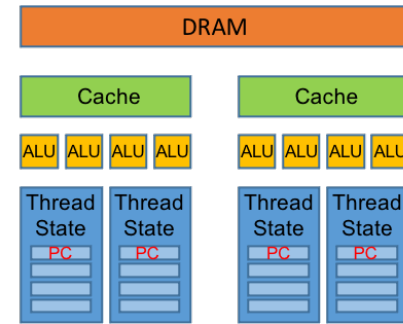
IIT Delhi

[vivekk@iiitd.ac.in](mailto:vivekk@iiitd.ac.in)

# Last Lecture (Recap)

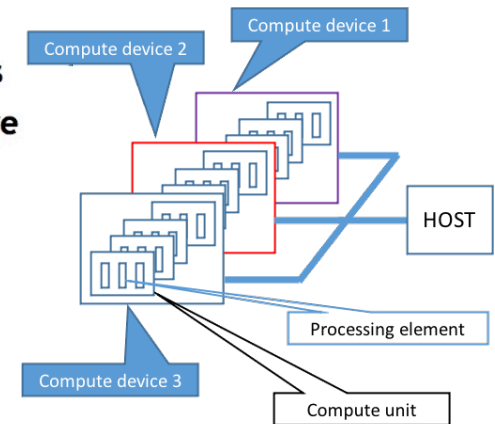
- Multicore processors are latency oriented, whereas GPUs are throughput oriented

```
int* a = new int[size];
int* b = new int[size];
// create 'a' vector on the GPU
compute::vector<int> vector_a(size, context);
// create 'b' vector on the GPU
// copy data from the host to the device
compute::future<void> future_a = compute::copy_async(
    a, a+size, vector_a.begin(), queue
);
// wait for copy to finish
future_a.wait();
// Create function defining the body of kernel
BOOST_COMPUTE_FUNCTION(int, vector_sum, (int x), {
    return x * 1.9 ;
});
// Launch the computation on the GPU using
// the command queue created above
compute::transform(
    vector_a.begin(),
    vector_a.end(),
    vector_b.begin(),
    vector_sum
);
// transfer results back to the host array 'c'
compute::copy(vector_b.begin(), vector_b.end(), b);
```



## Executing OpenCL Programs

1. Query host for OpenCL devices
2. Create a context to associate OpenCL devices
3. Create programs for execution on one or more associated devices
4. Select kernels to execute from the programs
5. Create memory objects accessible from the host and/or the device
6. Copy memory data to the device as needed
7. Provide kernels to command queue for execution
8. Copy results from the device to the host



# Today's Class

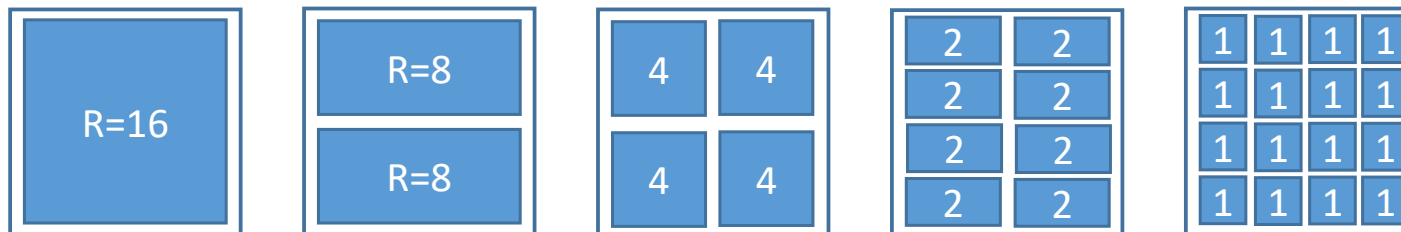
- ➡ ● Amdahl's law revisited
- Integrated CPU-GPU architectures
- Runtime solution for hybrid CPU-GPU parallelism

# Amdahl's Law Revisited

- Recall, maximum possible speedup for a program having fraction “f” of its total execution “W” parallelizable on N processing resources
  - $\text{Speedup}(f, N) = T_{\text{Sequential}} / (T_{\text{Sequential}} + T_{\text{Parallel}}) = W / \{(1-f)W + fW/N\}$
  - $\text{Speedup}(f, N) = 1 / \{ (1-f) + f/N \}$
- The assumption is that there is uniform scalability of processing resources when using more processors

# Amdahl's Law Revisited (Symmetric Cores)

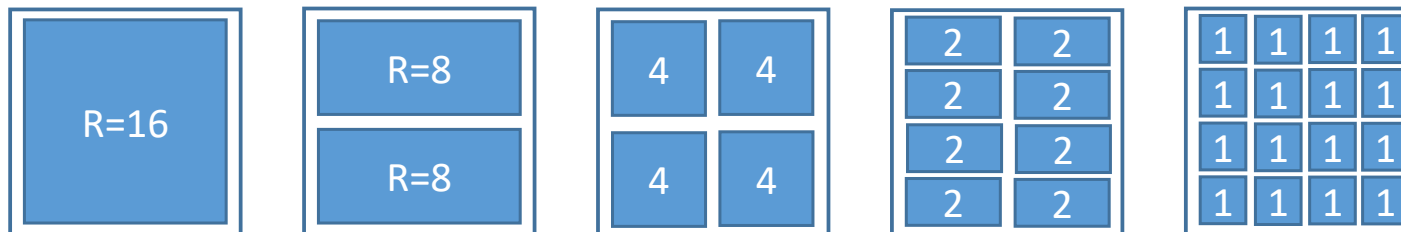
- Examples of different possible multicore processors that are using the same number of total processing resources
  - Each processor has identical cores (**symmetric processor**)
  - Each processor is using the same number of resources ( **$N=16$** )
    - E.g.,  **$N$**  could be total number of transistors on a processor
  - Each core is consuming  **$R$**  number of these resources
    - Hence, cores per processor is  **$N/R$**



Given a choice, which of these processors you would buy for your system?

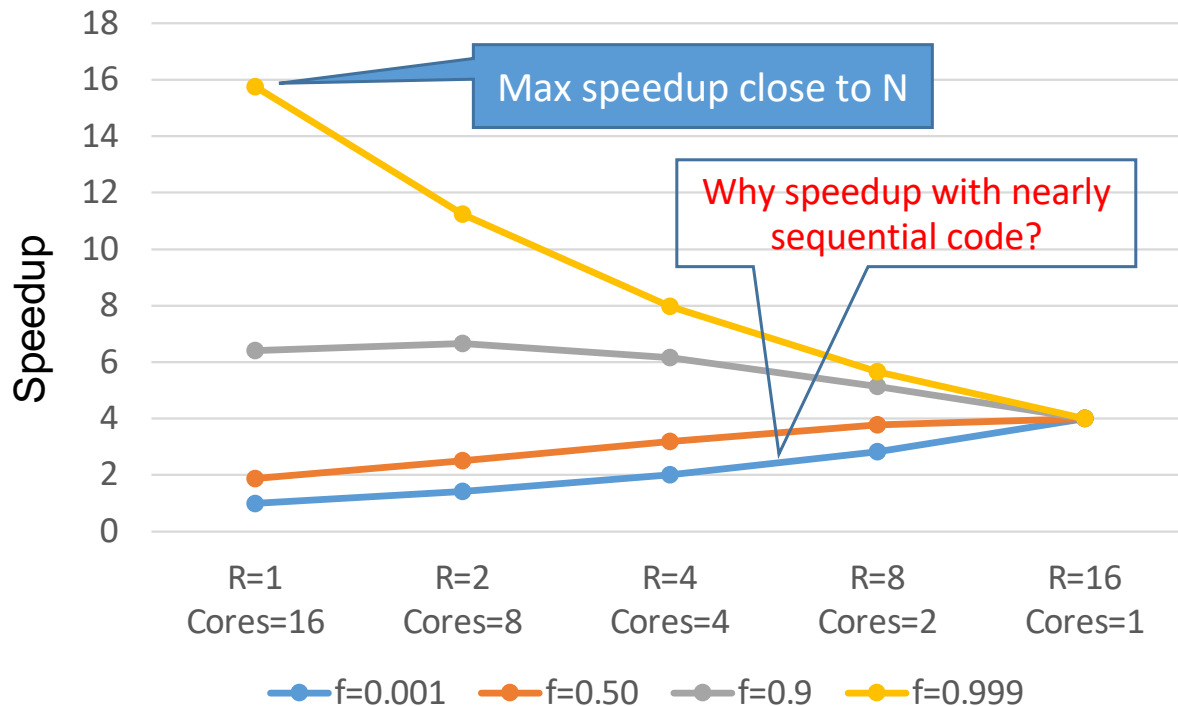
# Amdahl's Law Revisited (Symmetric Cores)

- Serial fraction of the program would now have a single core (sequential) performance on these combinations as:
  - $(1-f) / \text{Perf}_R$ 
    - $\text{Perf}_R$  is the performance of each of the single core of the processor type R shown below
- Parallel fraction use  $N/R$  cores at the rate  $\text{Perf}(R)$  each
  - $f / (\text{Perf}(R) * (N/R)) = f * R / \text{Perf}(R) * N$
- $\text{Speedup}(f, R, N) = 1 / ( \{(1-f)/\text{Perf}(R)\} + \{f * R / (\text{Perf}(R) * N)\} )$



# Amdahl's Law Revisited (Symmetric Cores)

- Symmetric multicore processor with total resources **N=16**. **Perf(R) approximated as square root of R** (adding resources won't give linear performance – **Why?**)
  - $\text{Speedup}(f, R, 16) = 1 / ( \{ (1-f)/\text{Perf}(R) \} + \{ f \cdot R / (\text{Perf}(R) \cdot 16) \} )$

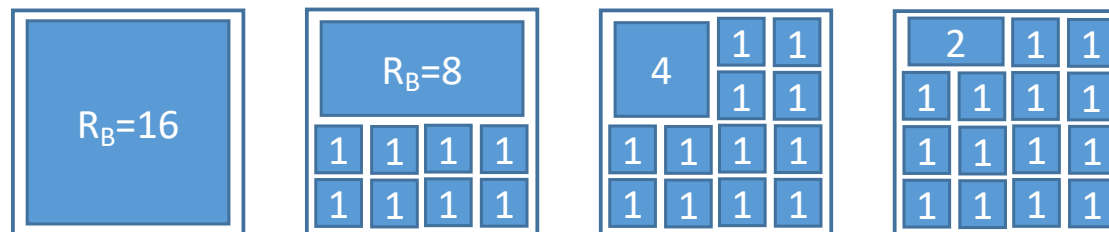


- At  $f=0.5$ , single core speedup better than 16 cores
  - Code should have parallelism for taking the benefit of multicores
- $f=0.999$  achieves far better speedup than  $f=0.9$  using same 16 cores
  - “f” matters – Need to have as much parallelism as possible
- **Any value of “f” achieves same speedup using R-16, C-1**

Hill and Marty 2008

# Amdahl's Law Revisited (**A**symmetric Cores)

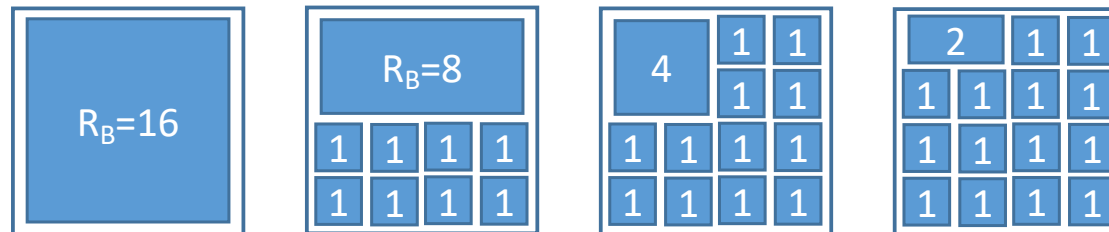
- Examples of different possible asymmetric multicore processors
  - **Symmetric** processor requires all cores to be equal, but **asymmetric** processor could have a mix of big and little cores
  - Each processor is using the same number of total resources (**N=16**)
    - One **B**ig core with  $R_B$  resources would leave  $N - R_B$  resources for little cores
    - Assuming each little core has  $R=1$ , total number of little cores are  $N - R_B$





# Amdahl's Law Revisited (**A**symmetric Cores)

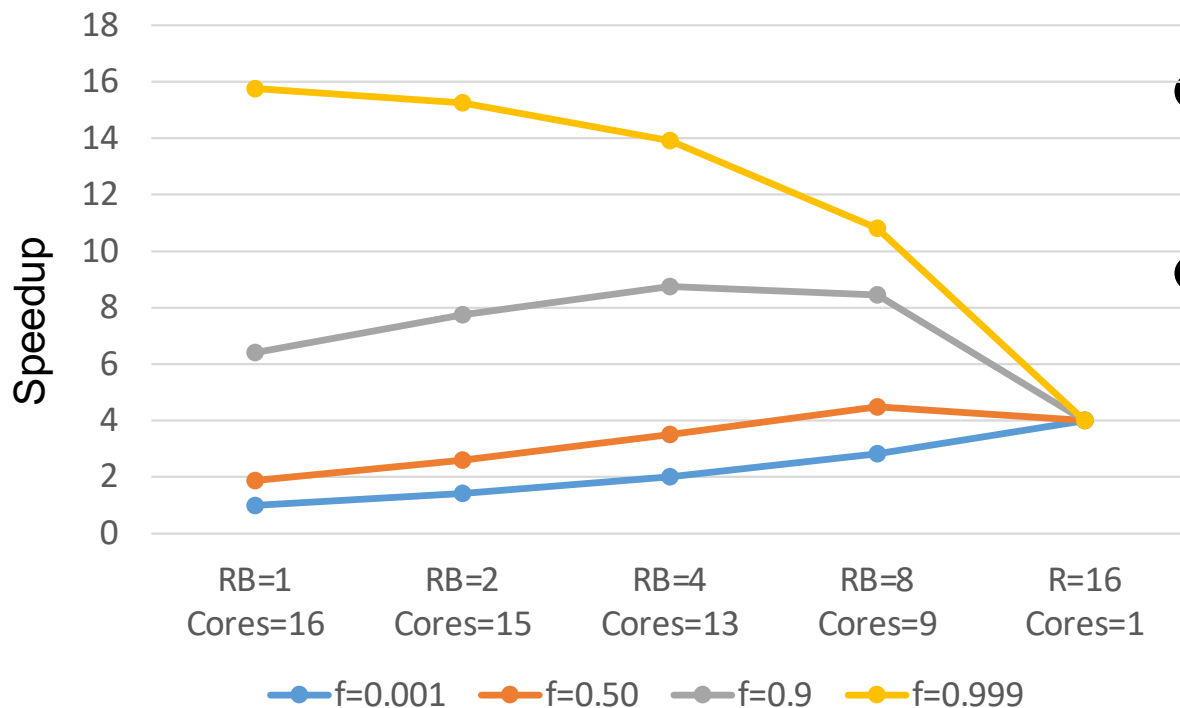
- Serial fraction would still be represented as in symmetric
  - $(1-f) / \text{Perf}(R_B)$
- Parallel fraction using one big core with  $\text{Perf}(R_B)$  performance and  $N - R_B$  little cores with  $\text{Perf}(1)=1$  performance would now be
  - $f / (\text{Perf}(R_B) + N - R_B)$
- $\text{Speedup}(f, R_B, N) = 1 / ( \{ (1-f) / \text{Perf}(R_B) \} + \{ f / (\text{Perf}(R_B) + N - R_B) \} )$



# Amdahl's Law Revisited (**A**symmetric Cores)

- Asymmetric multicore processor with total resources  $N=8$ .  $\text{Perf}(R_B)$  modeled as square root of  $R_B$

- $\text{Speedup}(f, R_B, 16) = 1 / ( \{ (1-f) / \text{Perf}(R_B) \} + \{ f / (\text{Perf}(R_B) + 16 - R_B) \} )$



- Most real world applications are a mix of sequential and parallel code
- Providing a heterogeneous processor with different types of cores can help in improving the performance of both the sequential and parallel portion
  - Sequential part runs on the big core
  - Parallel part runs on a bunch of little cores

Hill and Marty 2008

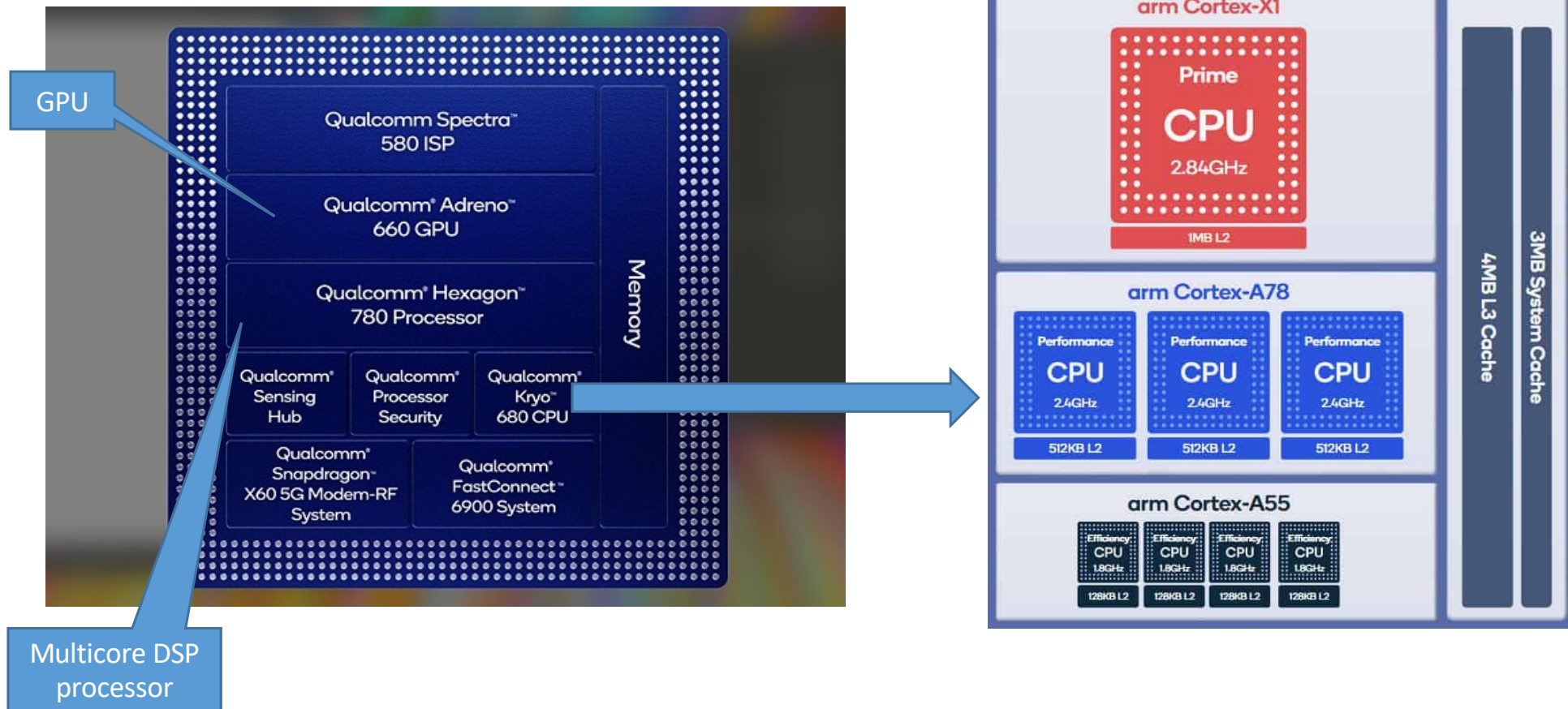
# Software Issues with Asymmetric Multicore

- When to use the big core v/s little cores?
  - Or, how to use them simultaneously if such application arrives
- Managing the locality
- Achieving energy efficiency execution

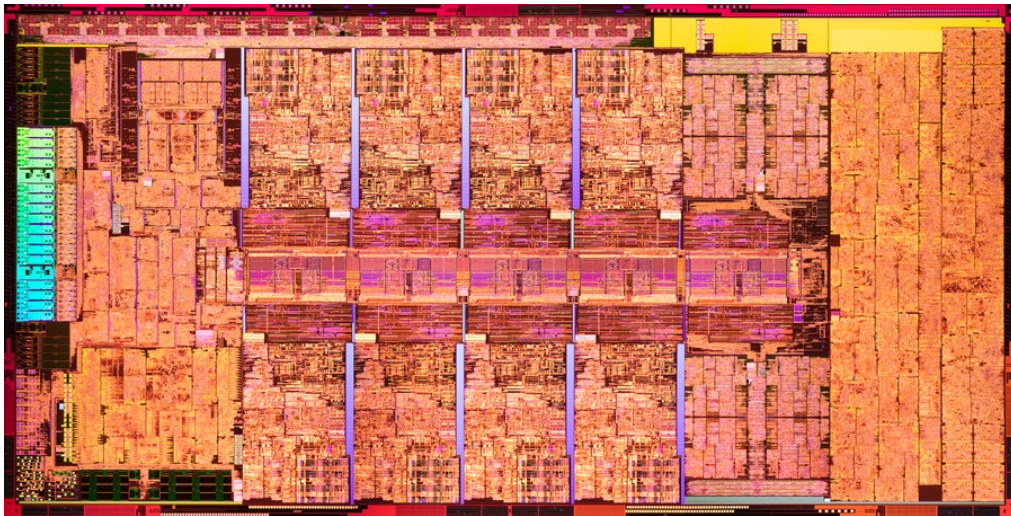
# Today's Class

- Amdahl's law revisited
- ➔ ● Integrated CPU-GPU architectures
- Runtime solution for hybrid CPU-GPU parallelism

# Qualcomm Snapdragon Mobile SoC



# Intel Heterogeneous SoC Architecture



- Alder Lake S (2021)

- 8 **P**erformance and 8 **E**fficient cores
  - P cores max frequency 5Gz
  - E cores max frequency 3.8GHz
  - Up to 24 threads supported
- Integrated GPU with 32 Execution Units
- Shared memory across CPU and GPU

Picture source: [https://en.wikichip.org/wiki/intel/microarchitectures/alder\\_lake](https://en.wikichip.org/wiki/intel/microarchitectures/alder_lake)

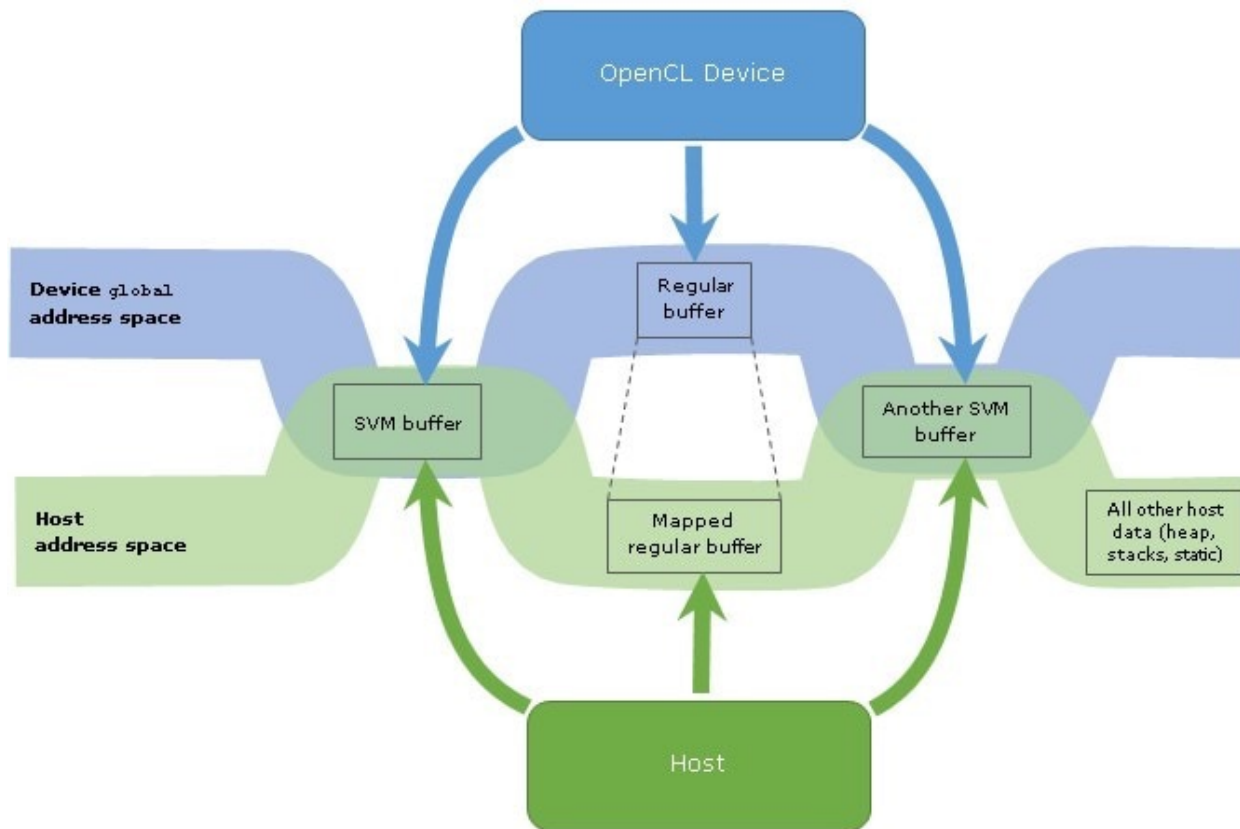
# Programming Integrated CPU-GPU SoC

1. **Setup inputs on the host CPU**
2. **Allocate memory on the host CPU**
- ~~3. **Allocate memory on the GPU**~~
- ~~4. **Copy inputs from the host to GPU**~~
5. **Start GPU kernel**
- ~~6. **Copy output from the GPU to host**~~

- Intel integrated CPU-GPU architecture
  - Host and the device share the same physical DRAM unlike the discrete GPUs
    - Enables using the same copy of memory between the CPU and GPU
      - Avoids explicit memory transfers for GPU kernel execution
- Supported by OpenCL 2.0 Shared Virtual Memory (SVM) feature



# OpenCL 2.0 Shared Virtual Address Space



- Allows sharing pointers across host and device
  - Coherency in the shared data modified across host and device on Intel integrated GPUs
- Supported by Boost.compute using simple APIs

Source: <https://www.intel.com/content/www/us/en/developer/articles/technical/opencl-20-shared-virtual-memory-overview.html>



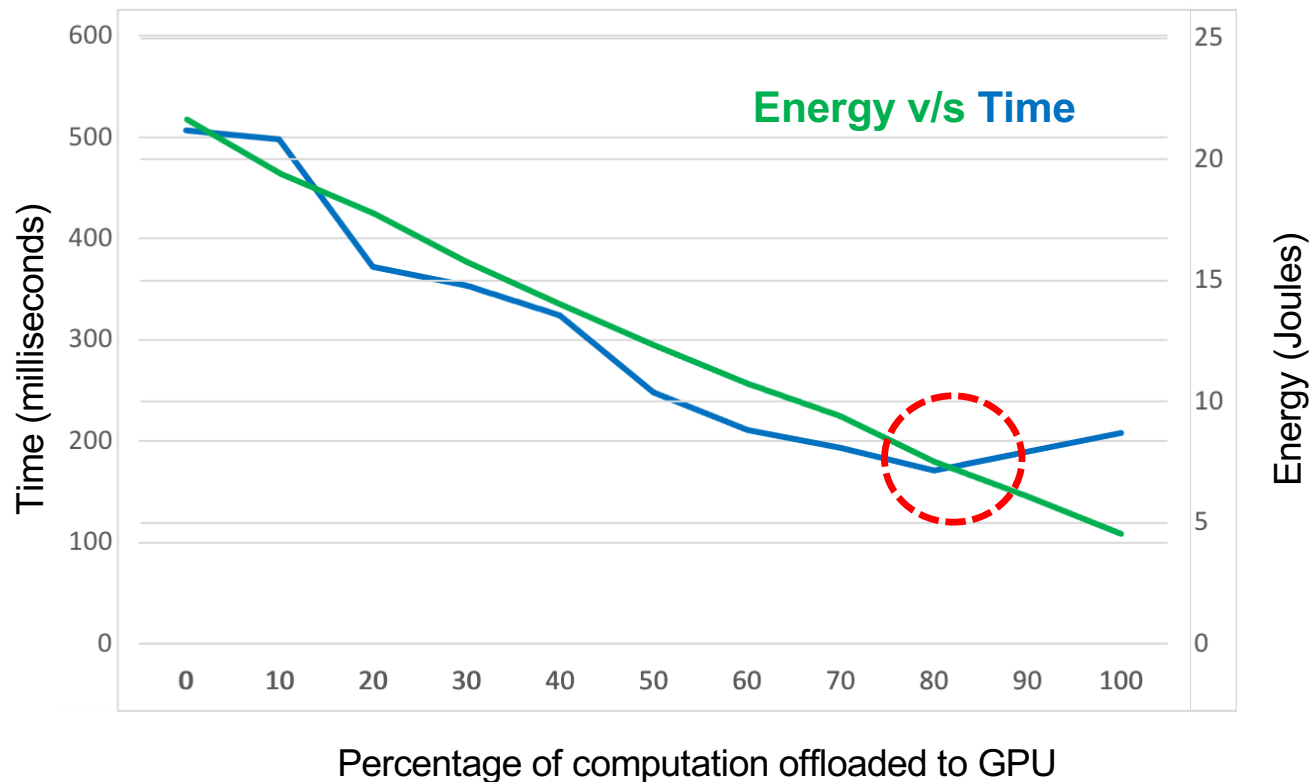
# Using Boost.Compute SVM on Intel SoC

- Demo of programs available in course GitHub repository
  - GPU-only vector addition
    - <https://github.com/hipec/cse513/blob/main/lec17/vecadd.cpp>
  - Hybrid CPU-GPU matrix multiplication
    - <https://github.com/hipec/cse513/blob/main/lec17/matmul.cpp>

# Today's Class

- Amdahl's law revisited
- Integrated CPU-GPU architectures
- ➔ ● Runtime solution for hybrid CPU-GPU parallelism

# Hybrid CPU-GPU Matrix Multiplication



- Experiment using the hybrid CPU-GPU matrix multiplication inside the course GitHub repo (size=1024x1024)
- Intel i7-6700 processor
  - 4-core CPU @ 3.4GHz
  - 24-EU GPU @ 350MHz
- Optimal execution at 80% offload
  - Would vary depending on application, processor, and computation size

# Software Challenges with Integrated CPU-GPU

- **Scheduling** – when to use the big core (or CPU) v/s little cores (or GPU)?
  - Heavily depends on the type of application
    - Offloading 80% computation on GPU was optimal in case of matrix multiplication, but it might be suboptimal in case of vector addition
      - Why?
    - Offloading percentage would be different for different kind of workloads (IO-bound, memory-bound, CPU-bound, etc.)
  - Depends on optimization criteria (time, or energy, or best of both, etc.)
- **Programming** – manually partitioning the workload would hurt programmer's productivity
- **Solution** – runtime assisted scheduling!

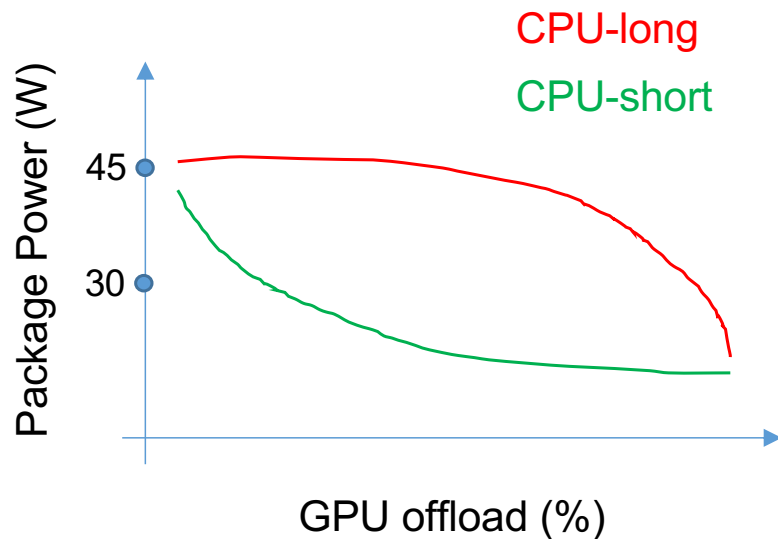
# Energy-Aware Scheduler for Integrated CPU-GPU<sup>1</sup>

- Classification of work-load into eight categories during actual execution (i.e., online profiling)
  - Memory-bound or Compute-bound
    - Ratio of LLC misses to total instructions retired
  - CPU or GPU suitable

Workload Category	CPU Time	GPU Time
CPU biased	Short	Long
GPU biased	Long	Short
Balanced short workload	Short	Short
Balanced long workload	Long	Long

[1] <https://dl.acm.org/doi/abs/10.1145/2854038.2854052>

# Energy-Aware Scheduler for Integrated CPU-GPU<sup>1</sup>



- Power usage with CPU long v/s short
  - Assume one job consumes 45W package power when only using the CPUs, and the same job consumes 30W when only using the GPU
  - Increasing the GPU offload from 0-100%
    - Power usage drops slowly for CPU-long
      - Concave shaped curve
    - Power usage drops quickly for CPU-short
      - Convex shaped curve

[1] <https://dl.acm.org/doi/abs/10.1145/2854038.2854052>

# Energy-Aware Scheduler for Integrated CPU-GPU<sup>1</sup>

- **How to do an online profiling in a running application?**
  - Classify the work-load by running a few iterations of the for-loop on both CPU & GPU simultaneously (as categorized in Slide #20)
  - Compare the findings of the online profiling either with a pre-trained processor specific model (on-the-fly is also possible, and is discussed in several papers)
    - A one-time characterization of the processor's power usage by measuring the power used by different kinds of workloads by varying the value of **alpha** ( $\alpha \rightarrow$  percentage of tasks offloaded to GPU)
      - Measured once on each experimental machine
    - This would give a power curve  $P(\alpha)$  for different categories of programs
      - $P(\alpha)$  is captured in form of polynomial expression

[1] <https://dl.acm.org/doi/abs/10.1145/2854038.2854052>

# Energy-Aware Scheduler for Integrated CPU-GPU<sup>1</sup>

- Steps for energy aware runtime scheduling of parallel\_for

1. If  $\alpha$  exists for this computation
  - Schedule iterations over CPU-GPU using this known  $\alpha$  value
    - Any applications falling in this category that we have already used before?
2. If  $\alpha$  does not exist for this computation then perform repetitive profiling to determine the minimum value of the target object function (objective could be energy, EDP, etc.)
  - a) Divide fixed sized  $N_{\text{chunks}}$  between CPU-GPU by changing  $\alpha$  in the range 0...100% (where,  $N_{\text{total}} \% N_{\text{chunks}} = 0$ )
    - If  $N_{\text{total}}$  is too small then exit this loop and launch entire computation on CPU only
  - b) Profile the CPU and GPU executions (CPU & GPU throughputs, memory bandwidth, etc.)
  - c) Characterize work-load and determine corresponding power curve
  - d) Increment  $\alpha$  in step of 0.1 and repeat the above steps until half of the  $N_{\text{chunks}}$  remaining

How to choose the value of  $N_{\text{chunks}}$ ?

Recall, Intel GPUs have EUs akin to a CPU-core, and are also N-way SMT with wide vector units

[1] <https://dl.acm.org/doi/abs/10.1145/2854038.2854052>



# Reading Materials

- Amdahl's law in the multicore era
  - [https://www.cs.washington.edu/mssi/2008/talks/hill\\_msr\\_uw\\_0808.pdf](https://www.cs.washington.edu/mssi/2008/talks/hill_msr_uw_0808.pdf)
- OpenCL 2.0 shared virtual memory overview
  - <https://www.intel.com/content/www/us/en/developer/articles/technical/opencl-20-shared-virtual-memory-overview.html>
- A black-box approach to energy-aware scheduling on integrated CPU-GPU Systems
  - <https://dl.acm.org/doi/abs/10.1145/2854038.2854052>

# Next Lecture

- Power management in multicore processors
- Quiz-3
  - Syllabus: Lectures 15-17