OpenCL

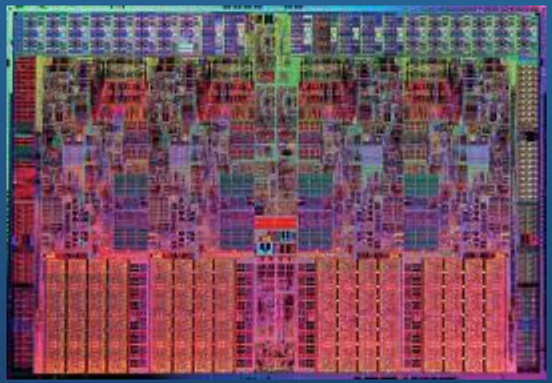
Using the CPU as a Compute Device

Matt Sellitto
Dana Schaa
Northeastern University
NUCAR

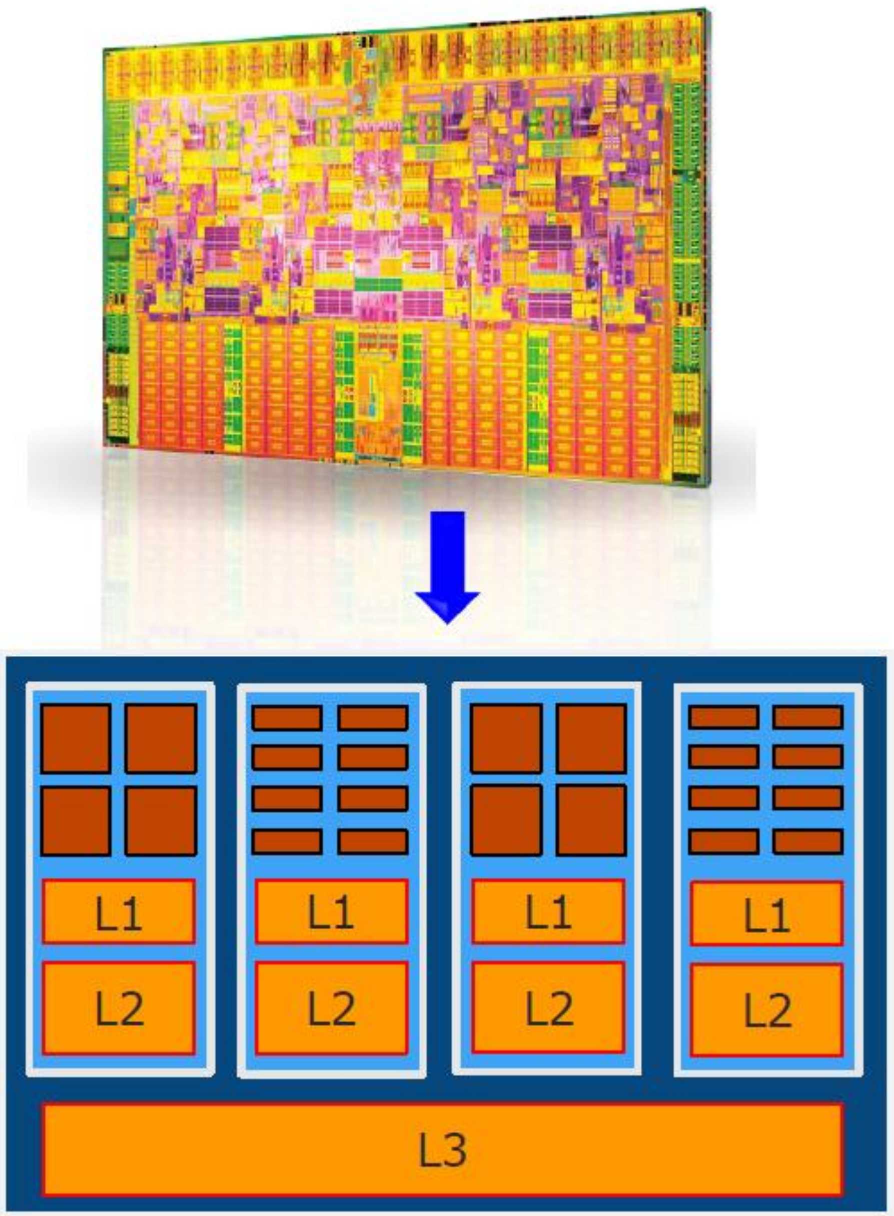
OpenCL and CPUs

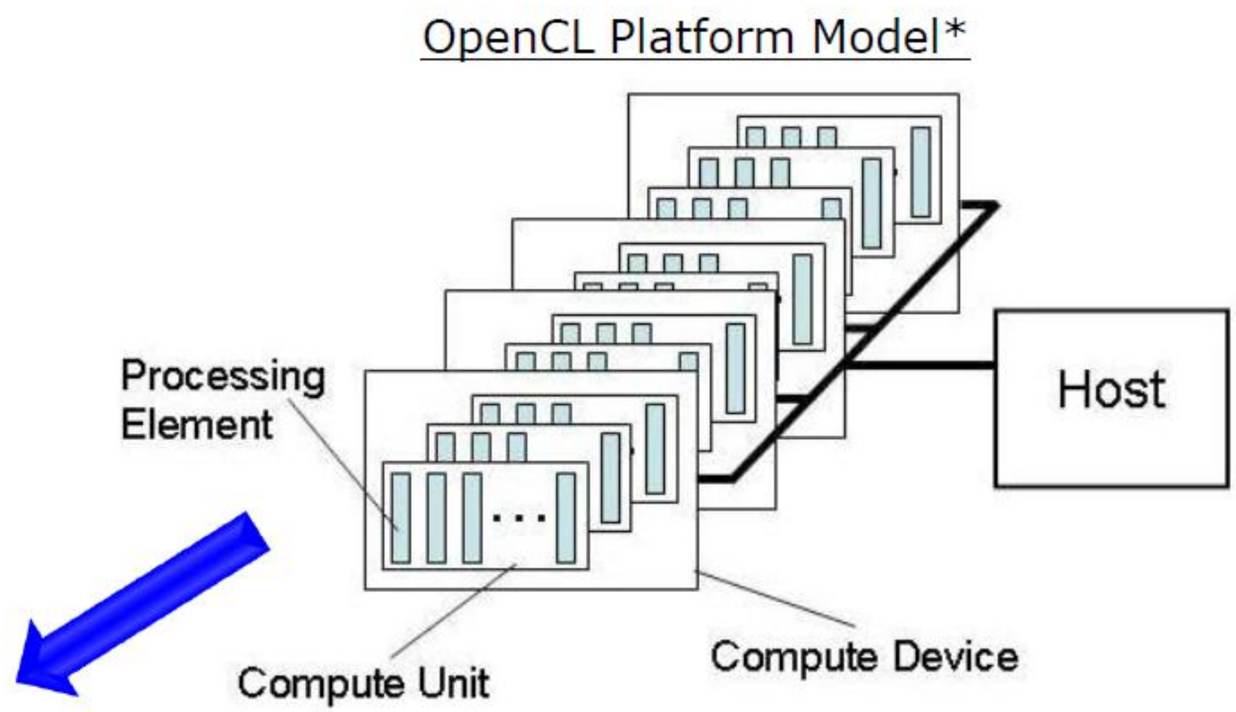
- OpenCL can use CPUs as a compute device just it can for GPUs.
- AMD and Intel’s OpenCL implementations support X86 CPUs (each one works with any CPU that has SSE3 +)
- The goal of heterogenous programming should not be just to offload work from the CPU but to make the best use of the available resources in the system.

OpenCL



OpenCL and CPUs





Core™ i7 975

- 8 Compute Units
  - Quad Core + SMT
- 4/8/16 Processing Elements per Unit
  - 128bit XMM registers
  - Data type determines # of elements…
- (32K L1 + 256K L2) Per Core, 8M L3 Shared
  - Not part of OpenCL platform model, but useful ☺

Points to note when working with OpenCL and CPUs

- There is no local memory, CPUs cache is utilized in OpenCL just like any normal CPU program.
  - Accesses to global memory may hit in the cache
  - No reason to use __local memory.
- Images give no performance benefit.
  - Just for convenience.
- CPUs will naturally be better at code that does a lot of branching as compared to GPUs.
- Better off using the CL_MEM_USE_HOST_PTR or CL_MEM_ALLOC_HOST_PTR flags when calling clCreateBuffer()
  - This tells the OpenCL library not to duplicate storage on the host side.
  - Since OpenCL “host” memory and OpenCL “device” memory are one in the same when using the CPU as a device.

OpenCL Data Parallelism + CPUs

- Explicit Data Parallelism
  - Use OpenCL Vector datatypes in each work-item
  - Tune vector width to that of underlying hardware
  - Combine with task-parallelism to exploit multiple cores
  - Requires More Tuning
- Implicit SIMD Parallelism
  - Write kernel as a "scalar program"
  - Use datasizes natural to your algorithm
  - OpenCL will automatically map these to the cores and SIMD units
  - Number of processing elements changes based on datatype used
- Both of these approaches work in OpenCL

OpenCL Explicit Data Parallelism on CPU

- Can use explicit parallel data structures like vectors to take advantage of the CPUs SIMD units.
- In this case arrays of floats become array of float4s, each vector addition can be performed in a SIMD fashion by the CPUs SIMD units.
- Must tune to specific device to maximize performance.

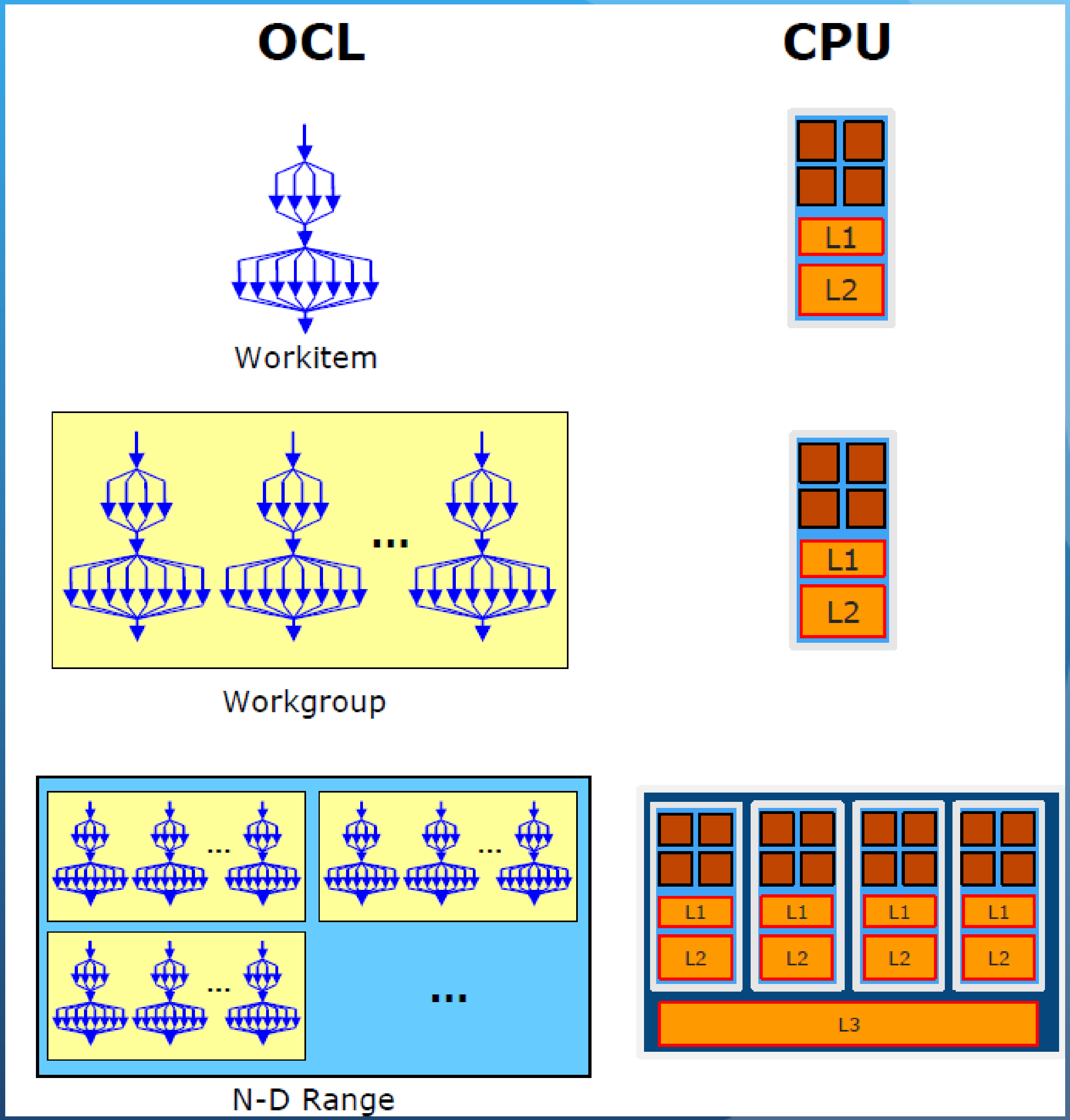
```
float a[N], b[N], c[N];
for (i=0; i<N; i++)
    c[i] = a[i]*b[i];

<<< the above becomes >>>>

float4 a[N/4], b[N/4], c[N/4];
for (i=0; i<N/4; i++)
     c[i] = a[i]*b[i];
```

OpenCL Explicit Data Parallelism on CPU

- Each work-item uses explicit SIMD operations to take advantage of the CPU vector units.
- Each work-group operates in a single compute-unit (HW thread)
- Several work-groups are executing over the entire compute-device.

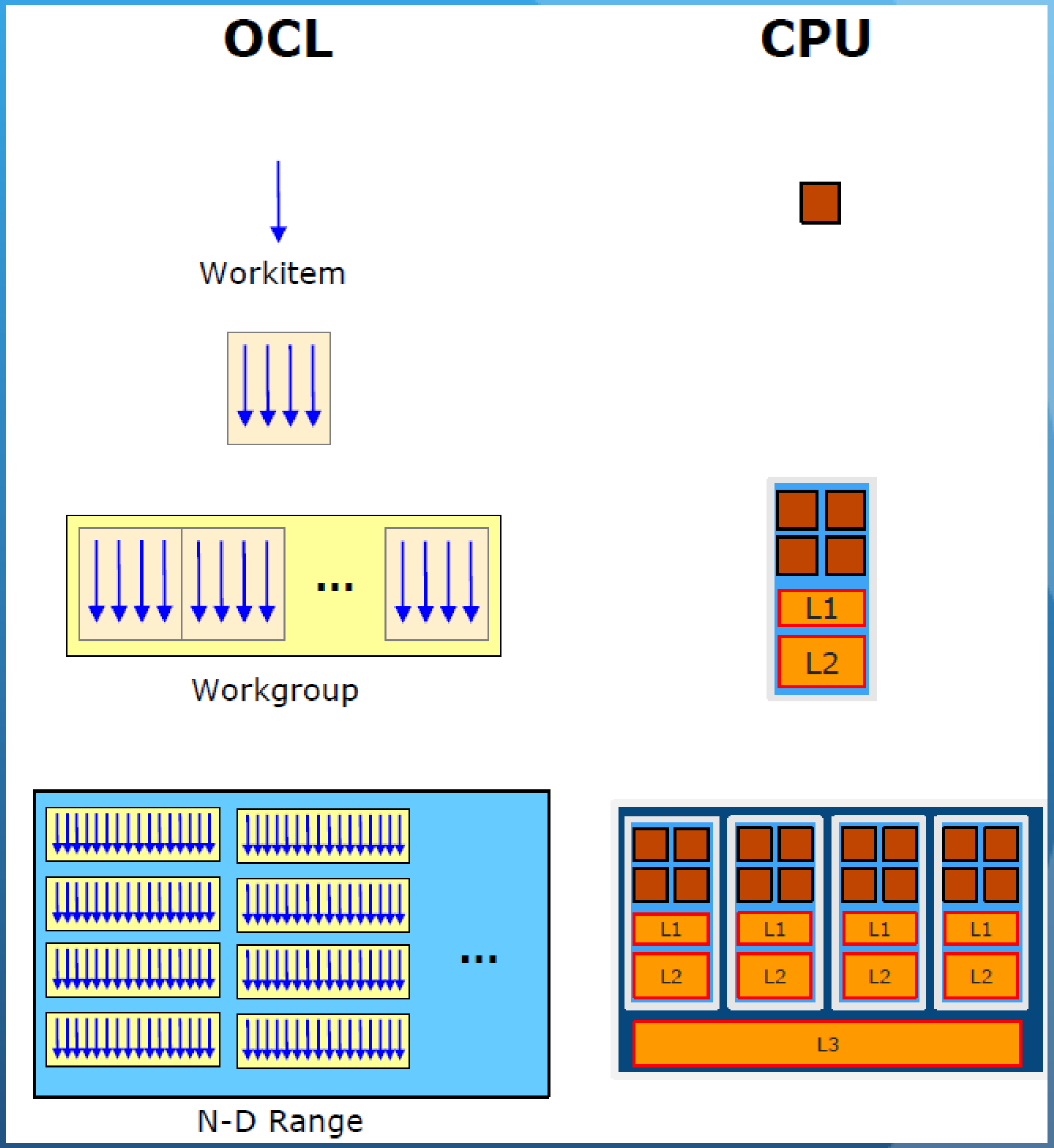


OpenCL Implicit Data Parallel on CPU

- Implicit SIMD Parallelism involved writing code as a "scalar" program the same as the GPU
- Advanced OpenCL compiler techniques are required to map the different threads to the CPU to maximize performance.
- Easier to code, but requires a good compiler.
  - Goal of the Intel OpenCL compiler is to use these techniques.
  - Intel Implicit Vectorization Module

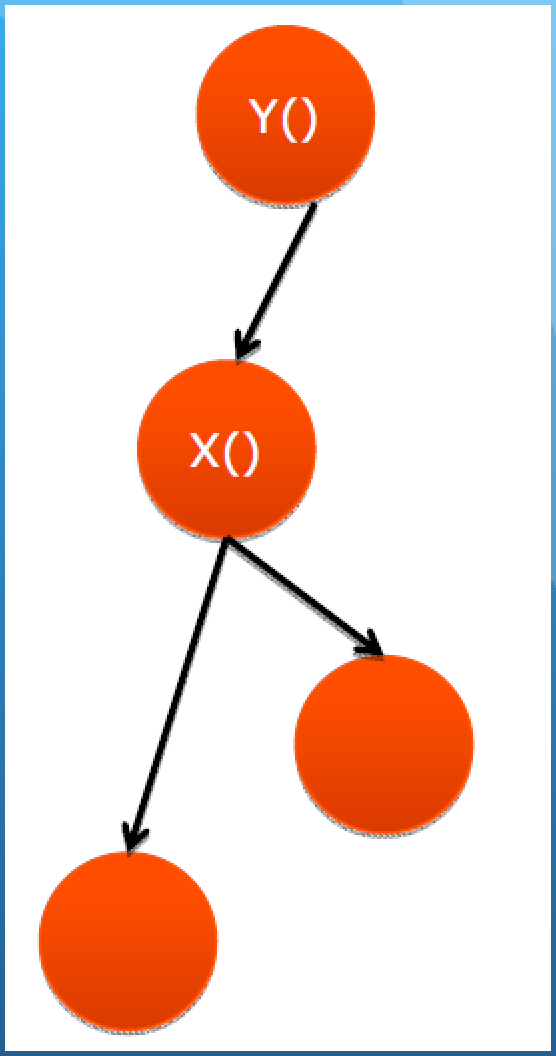
OpenCL Implicit Data Parallel on CPU

- Each work-item maps to a lane in the CPUs SIMD unit.
- Work-items are executed together in SIMD-width groups to utilize the SIMD units effectively.
- Work-group size should be a multiple of SIMD-width work-items (usually 4)



Task Level Parallelism

- OpenCL can also use CPUs for task-level parallelism.
- If you need to execute an operation that does not do the same operation on thousands of different data objects then a "task" may be more appropiate.
- Use OpenCL events to create task graph.
- Tasks can be coded to take advantage of SIMD hardware (by using vectors).
- Tasks only use one compute unit.



clEnqueueTask

- clEnqueueTask() is used to enqueue a task for execution on a device:

```
cl_int    clEnqueueTask (cl_command_queue command_queue,
                         cl_kernel kernel,
                         cl_uint num_events_in_wait_list,
                         const cl_event *event_wait_list,
                         cl_event *event)
```

- Tasks can be thought of as kernels with a single work-group and a single work-item.
- Used the same way clEnqueueNDRangeKernel()

To Do

- Modify the vectorAddition program to run on the CPU.
- Use explicit SIMD parallelism in the kernel by using OpenCL vectors to increase performance.

References

- Most of this presentation was derived from the Intel presentation at SIGGRAPH 2010 “Optimizing OpenCL on CPUs” by Offer Rosenberg