The Great Beyond: Higher Productivity, Parallel Processors and the Extraordinary Search for a Theory of Expression

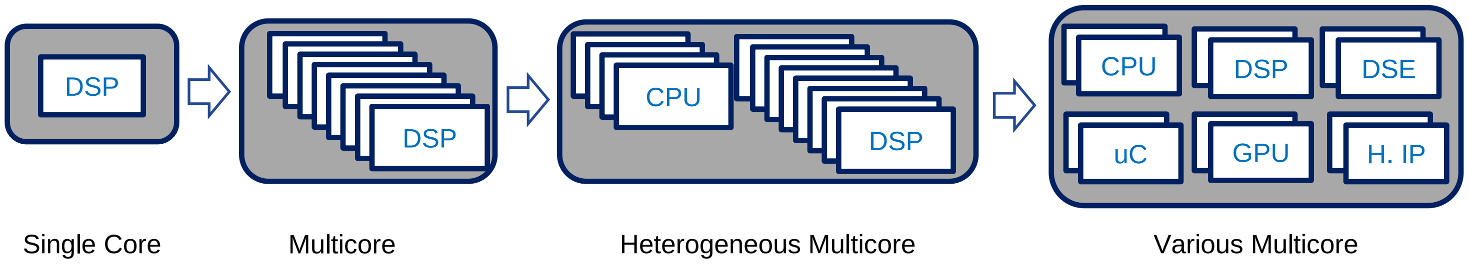
Alan S. Ward

Distinguished Member Technical Staff,
Manager, Multicore Development Tools

Texas Instruments

Title Inspiration:
"The Great Beyond: Higher Dimensions, Parallel Universes and the Extraordinary Search for a Theory of Everything",
Paul Halpern

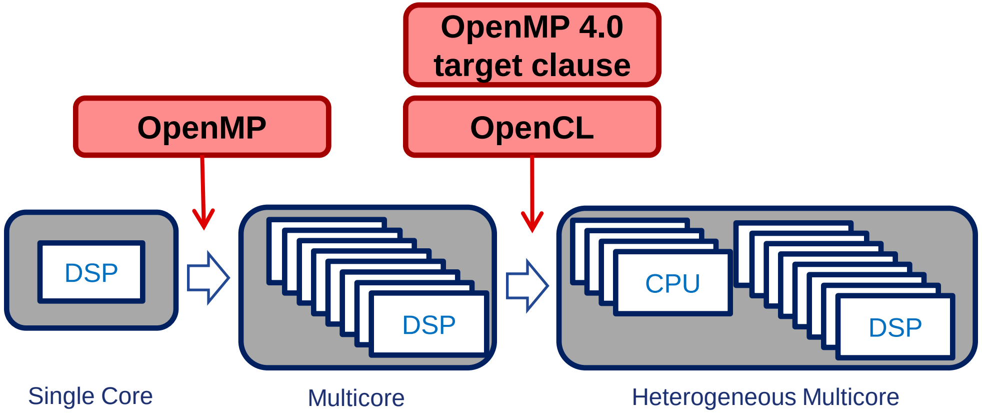
SoC Trends



- Power, Performance, and Area (Cost) is optimized through specialization and replication.
  - The business case is clear !
- The cost:
  - Increased software complexity
  - Specialized developer skills
  - Reduced application portability
- The goal:
  - Keep the benefits of hardware specialization and replication,
  - And ~~eliminate~~ reduce the delta cost!

DSP Digital Signal Processor
DSE Domain Specific Engine
H. IP Hardware IP block
uC Microcontroller

Maintaining Software Investment / Facilitating OpenCL Adoption



Single Core to Multicore

- OpenMP introduced
- New software application can run single or multicore

Multicore to Heterogeneous Multicore

- OpenCL introduced, but ….
  - What about existing code?
  - What about OpenMP in existing code?
  - What about malloc/free in existing code?
  - What about ???

An answer of “rewrite using “pure” OpenCL” was rejected

- Additional cost for status quo !
- Additional code base as the OpenCL version would not backward run on the multicore platforms.

Simple solution (examples)

- Allow OpenCL C code to call standard C code (including OpenMP enabled C code)
- Provide a means for dynamic heap allocation (all memory spaces) that does not conflict with OpenCL runtime allocations.

OpenCL C calling Standard C

```
const char *kern_src = " kernel void oclwrapper(global char * buf, int size) { alg(&buf[get_group_id(0)*size], size); } ";

Program::Sources source(1, make_pair(kern_src, strlen(kern_src)));
Program  program = Program(context, source);
program.build(devices, "ccode.obj“);
```

Standard C function

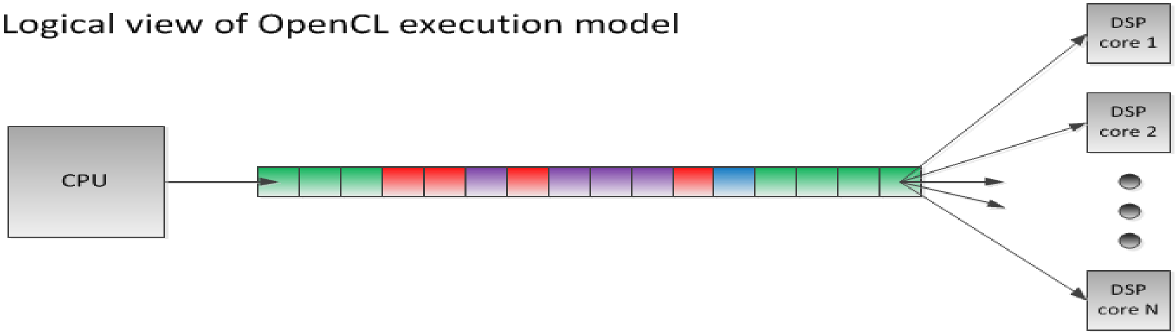
Resolved by this object code,
Passed as a build option

- The standard C Code is pre-compiled outside the OpenCL context and the resultant object filename is simply passed as an option to the OpenCL C build method.
  - Could use 1.2 separate compile and link model
  - However, current implementation is 1.1 conformant and we wished to us the 1.1 C++ bindings unmodified.

- If the alg function is OpenMP enabled
  - The OpenMP runtime is embedded in our OpenCL runtime, so nothing further is needed on the build side.
  - On the run side, user must ensure parallelism from OpenCL kernels and parallelism from OpenMP do not conflict
    - Ensured if the kernel is submitted to an “in order” queue as a task (i.e. 1 work-item)

TI’s Logical View of OpenCL execution

Logical view of OpenCL execution model



Color Key

- Barrier – Not executed, but cannot be popped from queue until all DSP cores are free. Adjacent barriers behave like one.
- Workgroup – Popped from queue and executed on one free dsp core.
- Coherency – Explicit cache coherency operations if needed. These are popped off queue and executed by all DSP cores.
- Task – Popped from queue and executed on one free dsp core. These contain embedded coherency operations.

Queue Patterns for different kernel enqueue methods

enqueueNDRangeKernel(Queue, ...)

enqueueTask(InOrderQueue, ...)

enqueueTask(OutOfOrderQueue, ...)

OpenCL C calling Std C calling malloc/free

```
const char *kern_src = " kernel void oclwrapper(global char * buf, int size)
                         {
                             __heap_init_ddr(buf, size);
                             std_c_app();
                         } ";
```

Initialize a heap that can be used in subsequent code

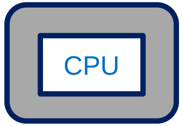
- Unadorned malloc/free are available
  - But, to a size limited heap.
  - Did not want to partition available memory between OpenCL managed and malloc managed.
  - Did not want to have devices send malloc/free requests to the host
- Created adorned malloc/free
  - Using additional built-in functions
    - __heap_init_ddr, __malloc_ddr, __free_ddr
    - __heap_init_msmc, __malloc_msmc, __free_msmc
    - __heap_init_l2, __malloc_l2
  - DDR and MSMC heaps persist for the lifetime of the buffer containing the heap
  - L2 heaps persist for the lifetime of a kernel invocation

A Different View of OpenCL:

OpenCL Reduces Software Complexity ?

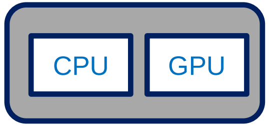
It depends on your frame of reference !

If this is your frame of reference

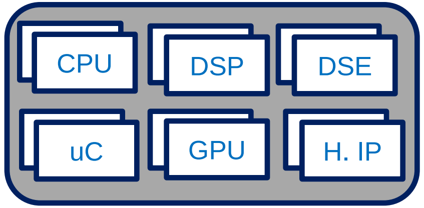


No

If this is your frame of reference



or



Yes

Custom Device feature extends OpenCL control

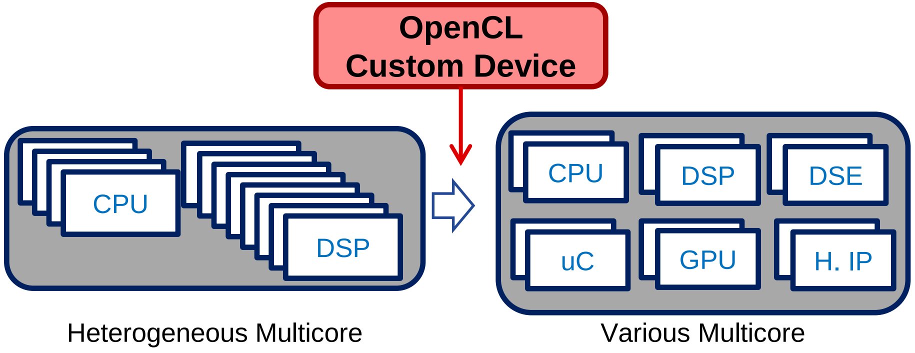
Three Categories of non OpenCL C capability

- uC, microcontrollers
  - No support floating point, (emulated at cost)
- DSE, Domain Specific Engine
  - Specialized ISA, not generally programmable
  - Can be programmed with a DSL
- H. IP, Hardware IP blocks
  - Fixed function
  - May have controls, configurations
  - Consumes and/or Produces

Still useful to leverage OpenCL buffers, events on these alternative devices.

Custom Device allows them to be programmed with either:

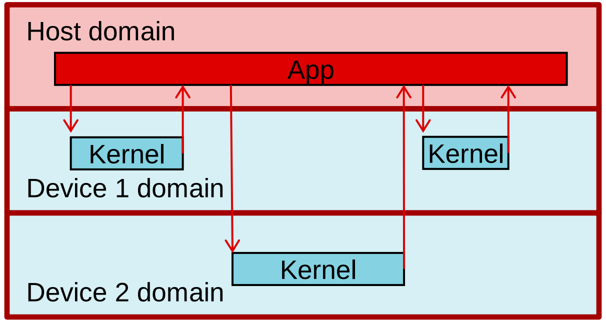
- An OpenCL C subset
- A DSL
- Selection from a set of fixed functions.



OpenCL execution model:
A fit for Classical Embedded?

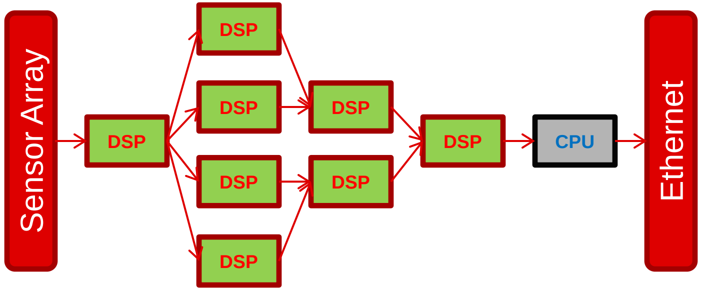
Typical OpenCL applications execute in a master-worker model.

- Host is responsible for execution, scheduling, and data availability.

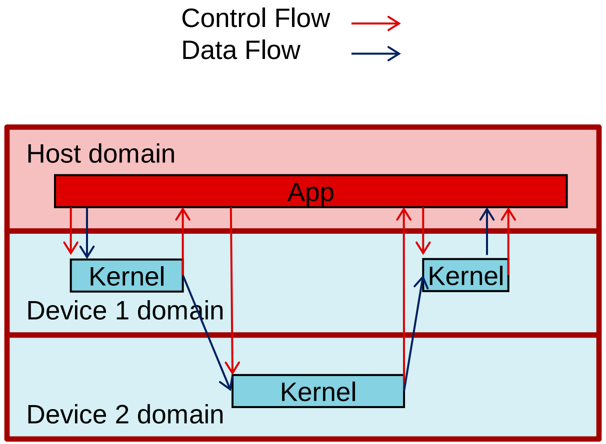


Typical Embedded execution is a data flow model.

- Distributed control and execution
- The algorithm is partitioned into multiple blocks.
  - Each block is assigned to a device compute unit.
  - The output of one block is input directly to the next block.
  - A block is stimulated awake by data ready
- Partition the algorithm to optimize performance
- The flow typically repeats on a regular basis



OpenCL execution model:
A fit for Classical Embedded?



In a shared virtual memory domain:

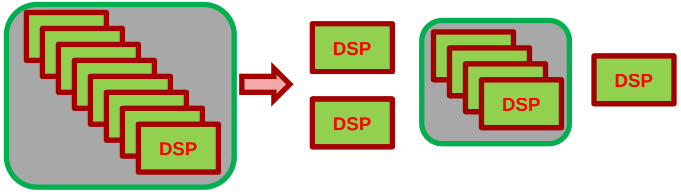
- The data can flow direct
- No communication hops through host required

OpenCL 2.x added a number of features that assist a Data Flow Model:

- Pipes
- Shared virtual memory, in general
- Fine grained virtual memory, memory ordering rules and atomics
- Device side kernel enqueue

OpenCL 1.2 added Device Partitioning

- Which allows a static partition of algorithmic blocks to reserved portions of a device.



But, What about ?

- Using the OpenCL 2.0 feature set
  - We can implement the data flow model within a device,
  - In a power efficient manner.

- But, what about data flow across devices?
  - Can't use device-side enqueue, for example
  - Perhaps?
  - Power efficient?