DEBUGGING SHADERS ON INTEL® GPUS: PAST AND PRESENT

Mircea Gherzan <mircea.gherzan@intel.com>
Fabian Schnell <fabian.schnell@intel.com>

GNU Tools Cauldron 2016, Hebden Bridge

Overview of the Intel® Processor Graphics Architecture

- Hardware thread with own registers grouped in register files:
  - General Register File (**GRF**): 128 general purpose registers (r0 … r127), 32 bytes each, byte addressable
  - Architecture Register File (**ARF**): PC, accumulators, index & predicate registers
- Execution Unit (**EU**): group of hardware threads sharing:
  - Functional units (SIMD FPU)
  - Memory access units
  - Branching units
- **Subslice**: array of EUs sharing instruction/L1/L2 caches & a thread dispatcher
- **Slice**: array of subslices sharing Fixed Function Units & L3 cache

Overview of the Intel® Processor Graphics Architecture (2)

- Shared physical memory (zero copy CPU/GPU data sharing)
- 5th Generation Intel® Core™ and later: shared virtual memory (pointer sharing)
- Instruction Set Architecture:
  - 32 byte instructions, 16 byte “compact” instructions
  - SIMD control flow (1 program counter per SIMD lane)
  - Predication for every instruction (enable/disable individual lanes)
- Documented:
  - Programmer’s Reference Manual (up to and including 6th Generation Intel® Core™)
  - Compute Architecture, Graphics API dev guide

Supported Compute & 3D APIs

- Intel® Cilk™ Plus
- OpenMP 4.0
- Khronos OpenCL™ 2.0
- Microsoft* C++ AMP
- Khronos OpenGL™ 4.3 (with GL-Compute)
- Microsoft* DirectX* 12 (with Computer Shader)
- Khronos Vulkan™

Intel® Cilk™ Plus for Intel® Processor Graphics

- Supported by the Intel® Parallel Studio XE 2017 and System Studio 2017
- Synchronous model base on compiler pragmas

```
#pragma offload target(gfx) pin(a, b:length(size))
    _Cilk_for(int i = 0; i < size; i++)
        a[i] *= b[i];
```

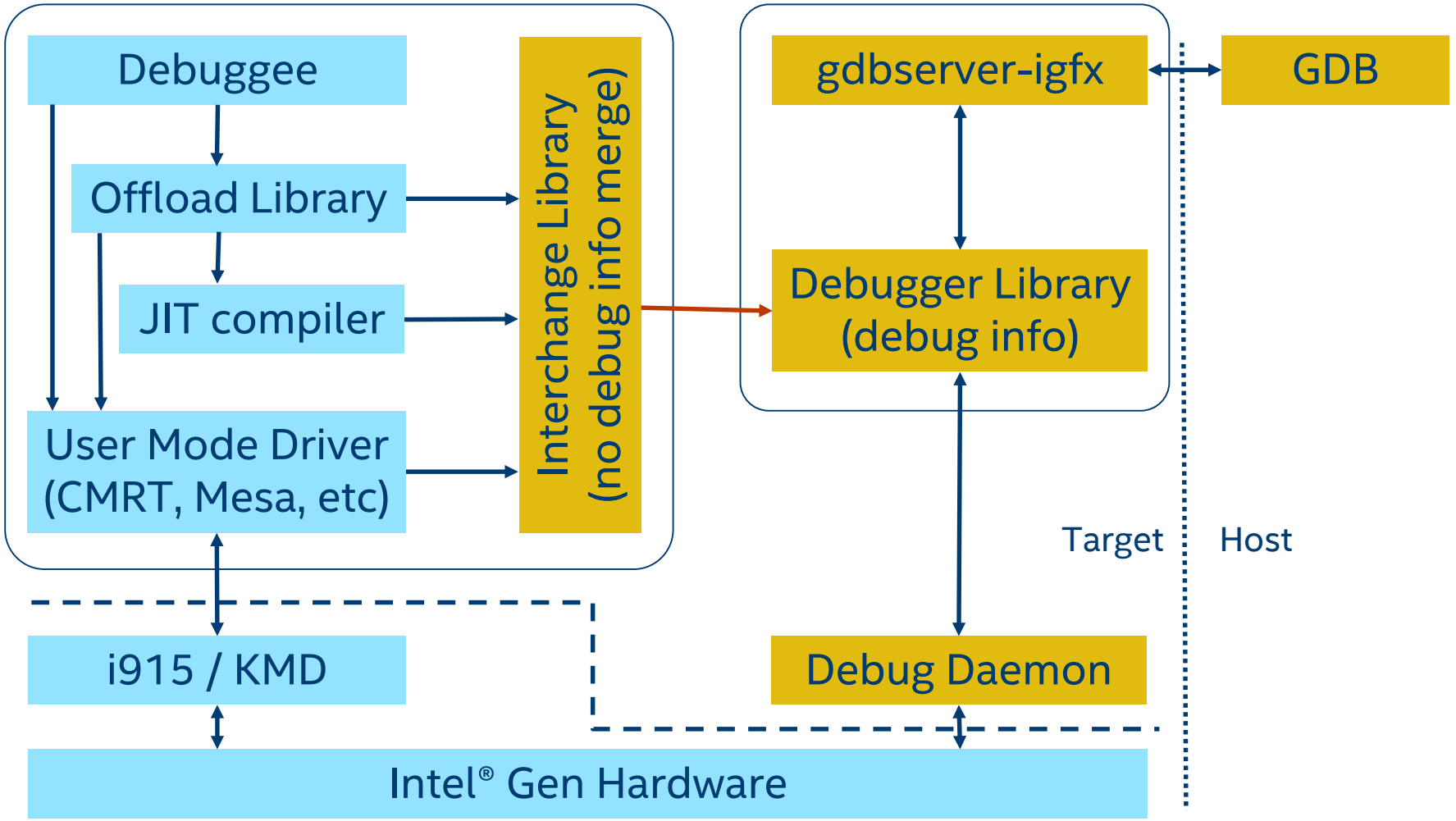
- Async model, explicit declaration of shaders (kernels) and submission

```
__declspec(target(gfx_kernel))
void shader(float *ptr, int sz) { _Cilk_for(int i=0; i<sz; i++) { … } }

_GFX_share(cpu_side_ptr, size);

int task_id = _GFX_enqueue("shader", cpu_side_ptr, size);

_GFX_wait(task_id, -1)
```

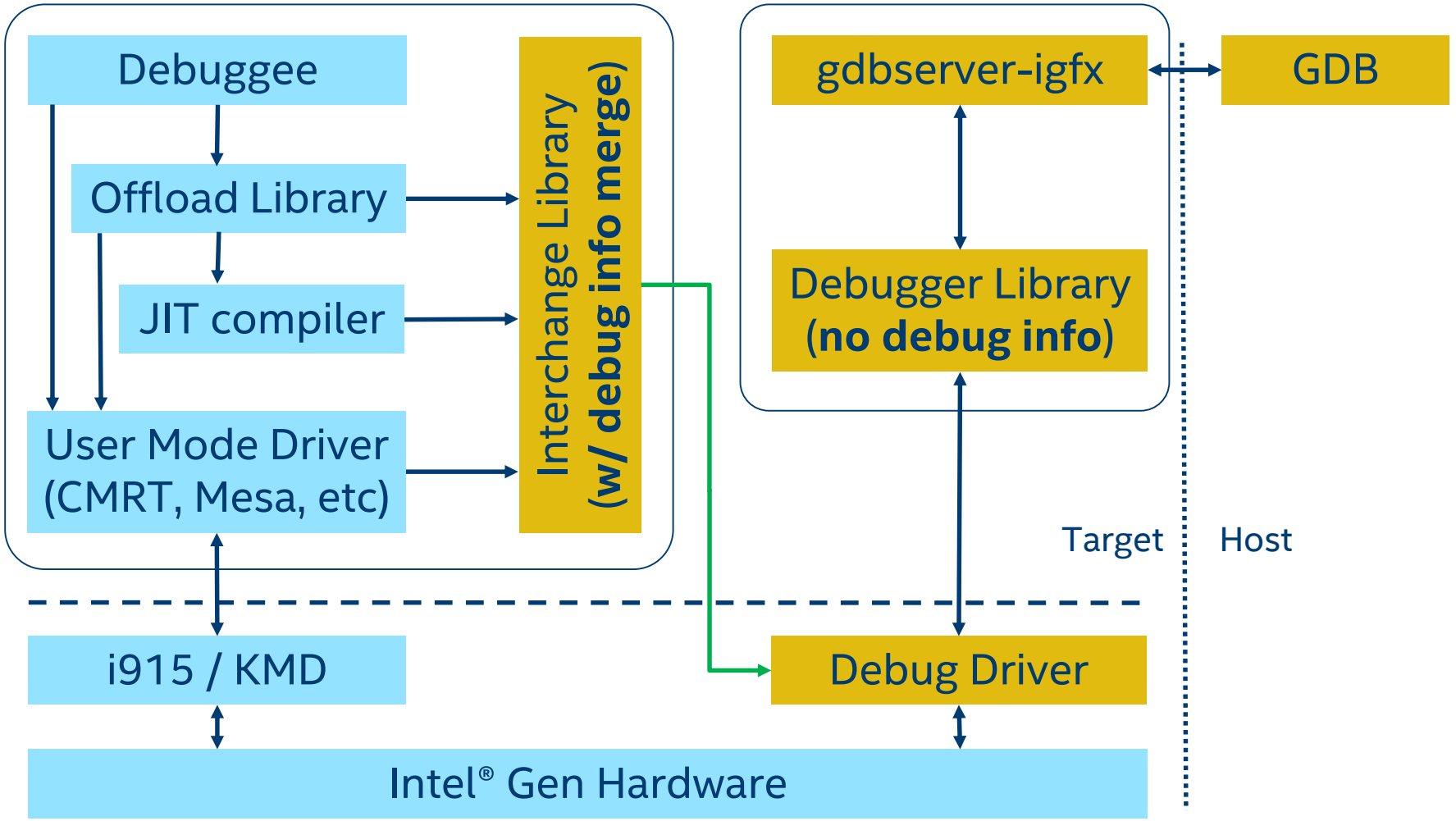
Shader Debugger: What's New?

- Microsoft Windows support
- All debugger stack components are now open-source:
  - Userspace libraries (debugger library, interchange library)
  - Kernel-mode driver for Linux (replacing the /dev/mem-based daemon)
    - No more requirements for the kernel configuration
  - Kernel-mode driver for Windows
- Rework of the debugger stack architecture:
  - Old architecture focused on the proprietary Intel® Cilk™ Plus /OpenCL offload stack (intermediary ISA + JIT compiler)
  - New architecture friendlier toward open-source runtimes (Mesa for OpenGL, Beignet for OpenCL)
    - Runtimes that compile shader languages directly to Gen ISA, no JIT compiler involved
- Support for debugging 3D shaders, not just compute shaders

Debugger Stack Architecture: Past



Debugger Stack Architecture: Present



Architecture Details

- Session-centric model:
  - Debugging session == running debugger instance
  - Session ID: PID of the process launching the shaders **or** user-controlled
- Client-server model at the debug driver level:
  - Server == debugger handling multiple shaders (pseudo-unique shader ID)
  - Clients == host application that submits shaders to the GPU
- Client: are my shaders going to be debugged? Whom shall I ask?
  - Interchange provides the API for checking if a debugger is active (based on the ID)
- Client: OK, a debugger is active, what to do now?
  - Compile the shaders without optimization and, if possible, emit DWARF to disk
  - Assign an pseudo-unique shader ID and notify the debugger (via the interchange) about the upcoming submission (ID, shader binary)

GDB Implementation Details

- New ELF machine type for Intel® Processor Graphics
- Port of GDB completely decoupled from the host/x86 side:
  - XML descriptions for the ARF and GRF registers
  - Supports the GPUs of the 4th, 5th, 6th and 7th Generations of Intel® Core™ processors
  - No new GDB commands
- Only supporting the “stop the world” synchronous model

Challenges in Porting GDB to Gen

- Segmented memory model:
  - Most addresses relative to a specific “base address”
  - Solution: encode “memory type” in the upper address bits
- Large number of threads in most scenarios (> 150):
  - Most threads might hit the same breakpoint
- DWARF: new operator needed because of the indirection based on two registers
- The General Register File (GRF) can be treated as both register and memory space

Requirements and Limitations

- No local debugging (host == target) with a local desktop compositor
  - Normal Windows operation
  - Linux with accelerated compositors (workaround via software rendering)
- GDB:
  - Backtrace support not available

Summing up

- New architecture of the debugging stack
- All components now open-source
- Proper kernel-mode debug driver for Linux
- Support for Windows
- 3D shader support
- More enabled runtimes (e.g. Beignet)

Q&A

Legal Disclaimer & Optimization Notice

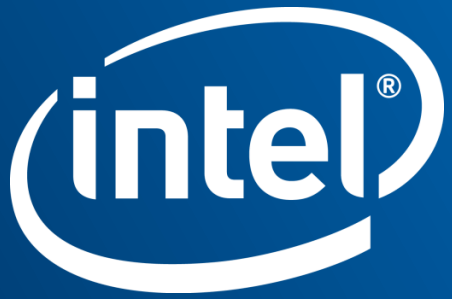
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®