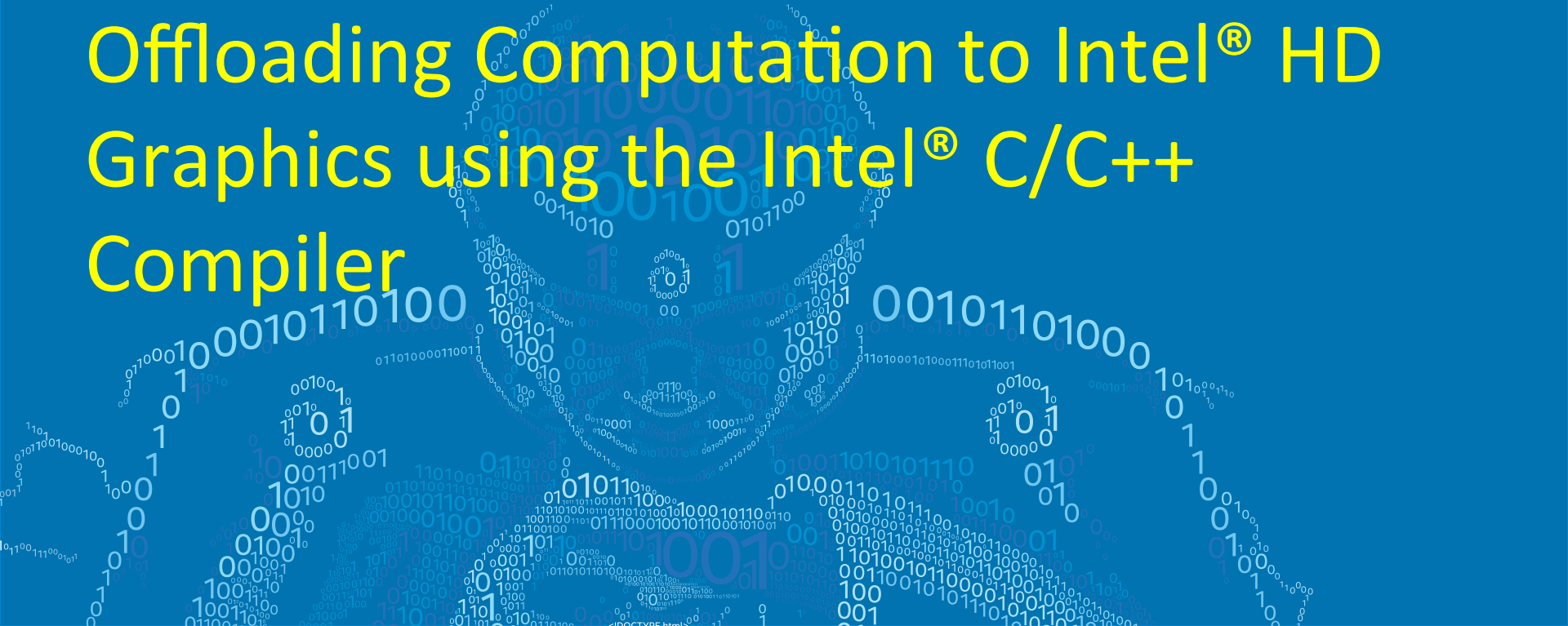
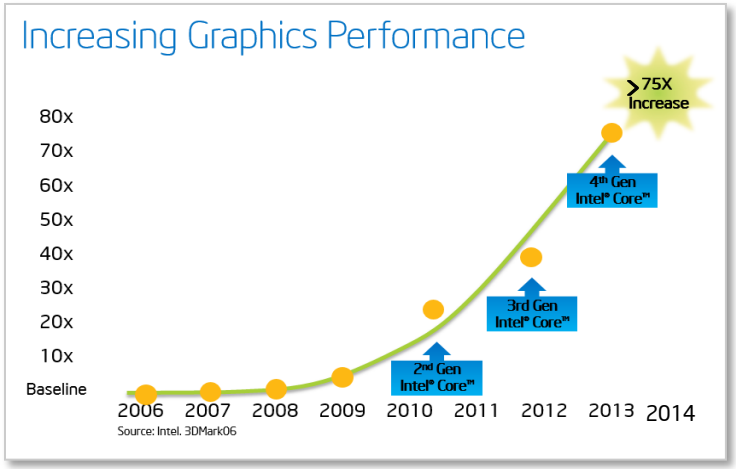
Intel® Software Conference 2015

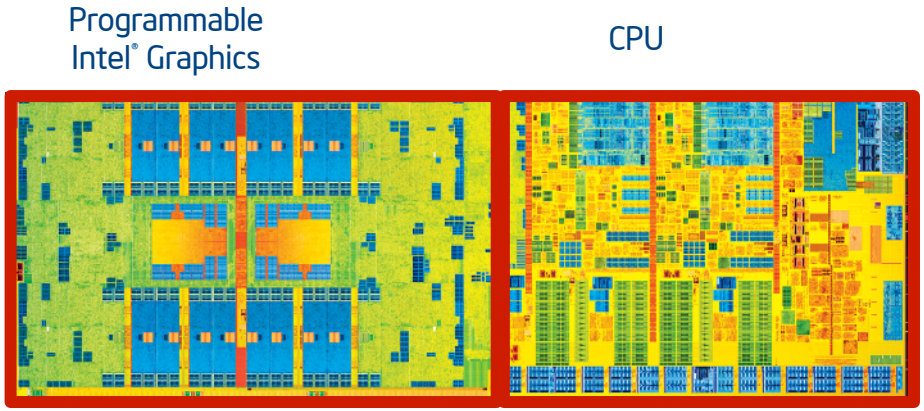
Offloading Computation to Intel® HD Graphics using the Intel® C/C++ Compiler



Intel® Processor Graphics



- Intel® Processor Graphics: 3D Rendering, Media, and Compute
- Graphic and cores physically share memory
- Some products are near TFLOPS performance (SP FP)
- The foundation is a highly threaded, data parallel compute engine
- Dedicated area keeps growing



Intel Processor Graphics is a key Compute Resource

Notice

Intel® Processor and Intel® Graphics

Intel® Graphics	Year	3rd generation Intel® Core™ Processor (Ivy Bridge)
Gen 6	2011	2rd generation Intel® Core™ Processor (Sandy Bridge)
Gen7	2012	3rd generation Intel® Core™ Processor (Ivy Bridge)
	2013	Intel® Atom Processor (Bay Trail)
Gen 7.5	2013	4th generation Intel® Core™ Processor (Haswell)
Gen8	2014/15	5th generation Intel® Core™ Processor (Broadwell)

Comparison Recent Generations Intel® Processor Graphics

	Gen 7	Gen 7.5			Gen 8	
	Intel® HD Graphics 4000	Intel® HD Graphics	Intel® HD Graphics 4200/ 4400/ 4600	Intel® Iris™ Pro Graphics 5200, Intel® Iris™ Graphics 5100, Intel® HD Graphics 5000	Intel® HD Graphics 5500, Intel® HD Graphics 5300,	Intel® Iris™ Graphics 6100, Intel® HD Graphics 6000,
APIs	DirectX* 11.0 DirectX Shader Model 5.0 OpenGL* 4.0 OpenCL* 1.1	DirectX* 11.1 DirectX Shader Model 5.0 OpenGL* 4.2 OpenCL* 1.2			DirectX* 12, 11.2 DirectX* Shader Model 5.0 OpenGL* 4.3 OpenCL* 2.0	
Execution Units (EUs)	16	10	20	40	24	48
FP ops/clck	256	160	320	640	384	768

Note: FP ops/clck:

MAD op (MUL+ADD) x 4-lane SIMD for 32bit x #FPUs x #EUs == 2 * 4 * 2 * #EUs == **16 x #EUs**

Peak GFLOPs for Processor Graphics (Selection)

Theoretical peak Performance: Scales with Clock Speed and number of Execution Units

16

	Mobile Processor Graphics Product Examples	Number of Execution Units	Graphics Clock, max	Theoretical Peak Compute GFLOPS
Gen 8 	Intel® Iris™ 6100	48	1.1	845
	Intel® HD Graphics 6000	48	1.0	768
Gen 8	Intel® HD Graphics 5500	24	1.0	384
	Intel® HD Graphics 5300	24	0.85	326
Gen 7.5	Intel® HD Graphics 4400	20	1.1	354

Note: Additional compute GFLOPS available in CPU cores.

Peak GFLOPS scale with: Num EUs & Clock Speed

Comparison to Core Performance

Single precision floating point performance for 3GHz frequency:

Intel® 2nd and 3rd Generation Core™ processors (Sandy Bridge, Ivy Bridge)

8/(AVX vector) * (MUL + ADD) x 3 x 10**9 = 48 GFLOPS/Core

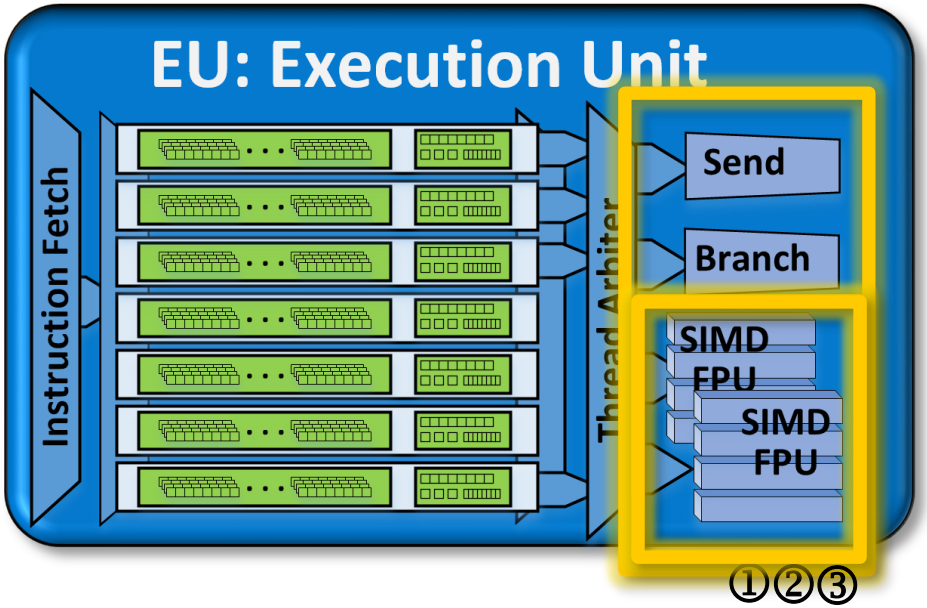
Intel® 4th and 5th Generation Core™ processors (Haswell, Broadwell)

8/(AVX vector) x 2 FMA x (MUL+ADD) x 3 x 10**9 = 96 GLOPS/Core

That is:

The latest high-end Intel® Graphic models provide even more (theoretical) compute resources (SP FP) than 8 processors cores of the latest CPU architecture !!

EU: Instructions & FPUs



① Instructions:

- Reads 2 or 3 src registers
- Writes 1 dst register
- Instructions are variable width SIMD.
- Logically programmable as 1, 2, 4, 8, 16, 32 wide SIMD
- SIMD width can change back to back w/o penalty
- Optimize register footprint, compute density

② 2 Arithmetic, Logic, Floating-Pt Units

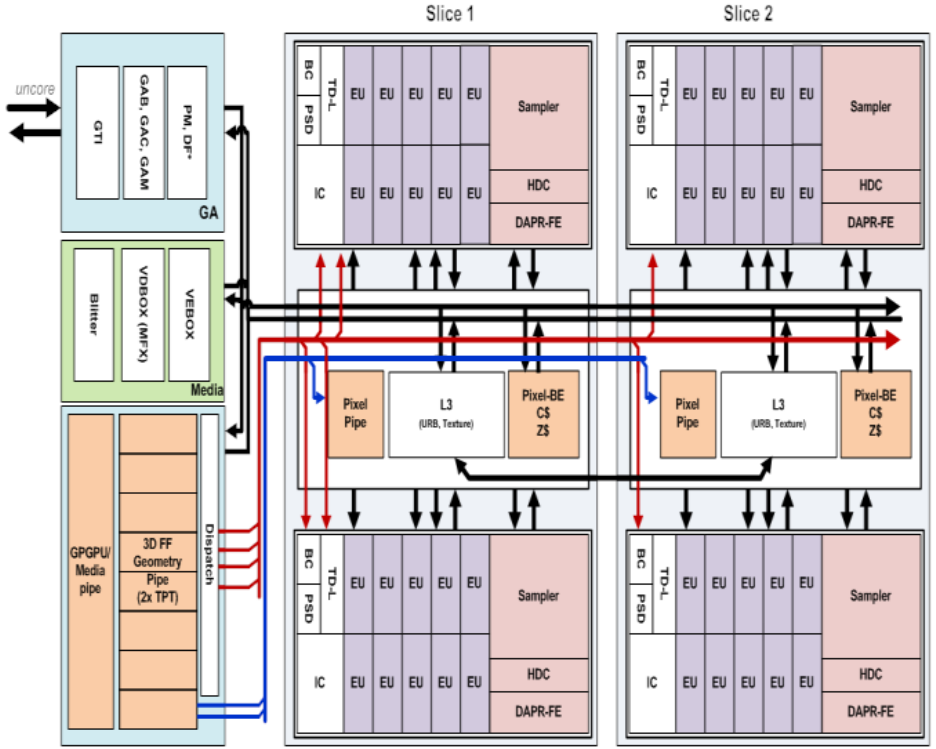
- *Physically* 4-wide SIMD, 32-bit lanes

③ Min FPU instruction latency is 2 clocks

- SIMD-1, 2, 4 float ops: 2 clocks
- SIMD-8 float ops: 2 clocks
- SIMD-16 float ops: 4 clocks
- SIMD-32 float ops: 8 clocks

FPUs are fully pipelined across threads: instructions complete every cycle.

GT3 Top Level Block Diagram (4th generation Intel® Core™ Processors)



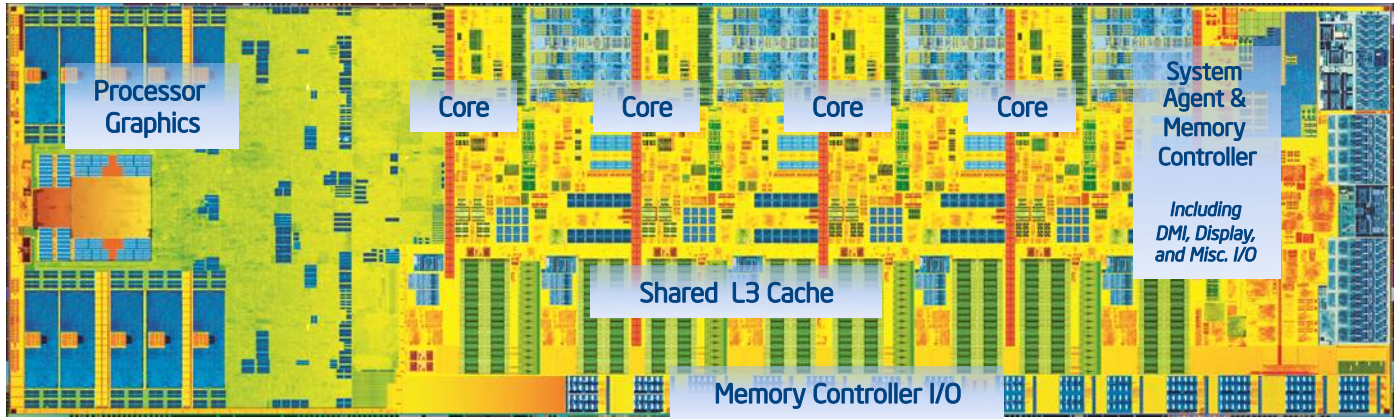
- 5 EUs Per row
- 2 rows per subslice
- 2 subslices per slice
- 2 slices (40 EUs total) in GT3
- 7 Threads Per EU
  - 280 threads in GT3
- 128 Registers per thread
  - 4KB per thread!
  - 1120KB in regfile in GT3
- 32K IC in each ROW (5EUs)
- 256KB data cache per slice (L3 only)

Intel® Compiler Support for Intel® Graphics Technology

Also called ‘GFX Compiler’ or ‘GT Compiler’

Why C/C++ Compiler Support for Intel® Graphics ?

- Latest Intel® Core processors and above have significant compute power (CPU + GFX)
- Better utilization of silicon area on the processor (performance & power)
- Simple Cilk-based offload model : Extension of standard C/C++ language, not a new language
- Supports incremental source code adaption (different from other models like OpenCL)
- All advantages of Intel® Compilers (vectorization, optimization)



Ease of Porting to GFX – the Idea

Original Host code:

```
void v_add(float *a, float *b, float *c)
{
  for(int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
  return;
}
```

Parallel Host code using Intel® Cilk™ Plus:

```
void v_add(float *a, float *b, float *c)
{
  cilk_for(int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
return;
}
```

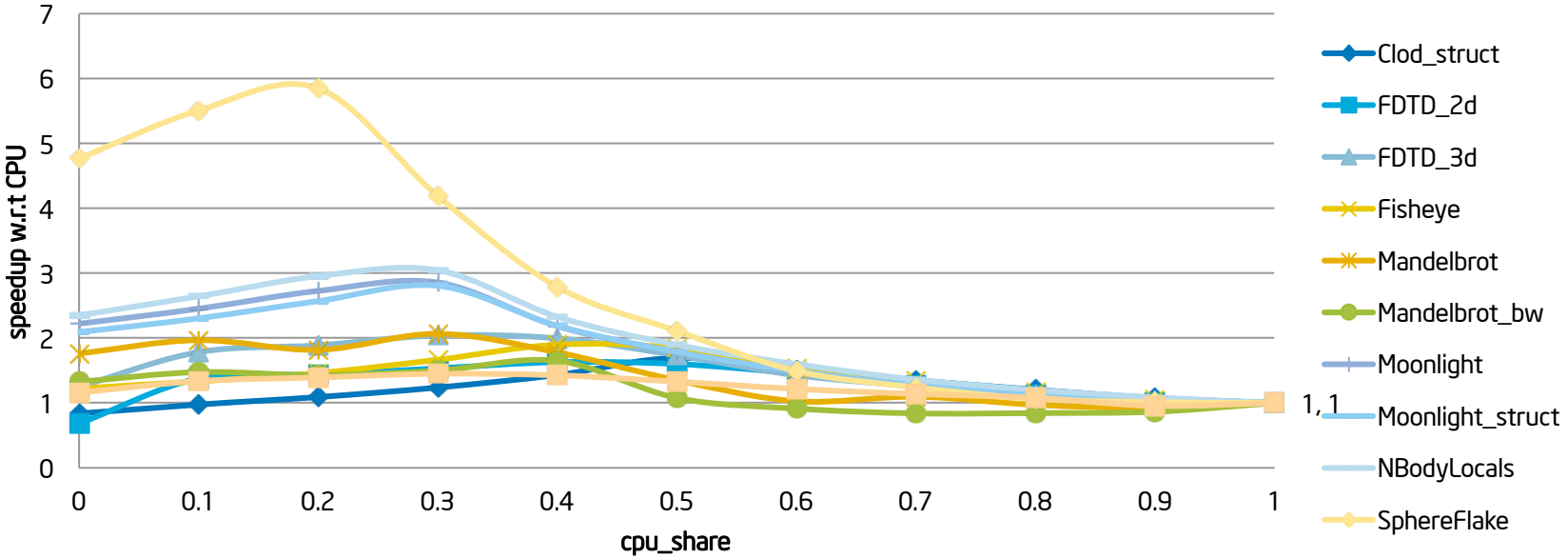
Offloading function body to GFX:

```
void v_add(float *a, float *b, float *c)
{
  #pragma offload target(gfx) pin(a, b,
c:length(N))
  cilk_for(int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
  return;
}
```

Creating GFX kernel for asynchronous offload:

```
__declspec(target(gfx_kernel))
void v_add(float *a, float *b, float *c){
   cilk_for(int i = 0; i < N; i++)
     c[i] = a[i] + b[i];
  return;
}
```

Performance gain using GFX+CPU compute power



Left to right -> Pure GPU execution to Pure CPU execution
0 – Pure GPU execution mode , 1- Pure CPU execution mode
0.1 – 10% workload on CPU and 90% on GPU
0.9 – 90% workload on CPU and 10% on GPU

System Specification:

OS : Windows Server 2012
Processor: 4th generation Intel® Core™ i7-4770 3.5 GHz
Memory : 16 GB
Processor Graphics : Intel® HD Graphics 4600
Intel® HD Graphics Driver: 15.36.7.64.3960
Compiler version : Intel ® C++ Compiler XE 15.0
Compiler option : -std=c++11 -xAVX

Offload Models

Synchronous Offload

- Annotate data parallel code section with #pragma offload target(gfx)
- Annotate functions invoked from the offloaded sections and global data with __declspec(target(gfx))
- Host thread waits for the offloaded code to finish execution
- Constraint - #pragma offload target(gfx) statement should be followed by a cilk_for loop
- Compiler automatically generates both host side as well as GFX code

Synchronous offload : Vector addition

```
void vector_add(float *c, float *a, float *b)
{
#pragma offload target(gfx) pin(a, b, c:length(ARRAYSIZE))
        cilk_for(int i = 0; i < ARRAYSIZE; i++)
        {
                c[i] = a[i] + b[i];
        }
}
```

Synchronous offload – Matrix Multiplication Kernel

```
void matmul_tiled(float A[][K], float B[][N], float C[][N]) {
#pragma offload target(gfx) \
    pin(A:length(M)) pin(B:length(K)) pin(C:length(M))
    cilk_for (int m = 0; m < M; m += TILE_M) {                    //
        cilk_for (int n = 0; n < N; n += TILE_N) {                //
            // (c) Allocate current tiles for each matrix:
            float atile[TILE_M][TILE_K], btile[TILE_N], ctile[TILE_M][TILE_N];
            #pragma unroll
            ctile[:][:] = 0.0;                                    // initialize result ti

            for (int k = 0; k < K; k += TILE_K) {                 // (d) calculate 'dot produc
                #pragma unroll
                atile[:][:] = A[m:TILE_M][k:TILE_K];              // (e
                #pragma unroll
                for (int tk = 0; tk < TILE_K; tk++) {             // (f
                    btile[:] = B[k+tk][n:TILE_N];                 // (g
                    #
                }
            }
            #pragma
            C[m:TILE_
        }
    }
}
```

Pin clause ensures the data is available in the main memory for GPU access.

cilk_for annotation for "nested for loops" will collapse the iteration space of the nested loops

Use Intel® Cilk™ Plus Array Notation to explicitly generate vector instructions rather than traditional scalar instructions.

#pragma unroll instructs the compiler to use direct register addressing which is efficient in comparison to indirect register addressing which uses address register as index register

1. TILE_M=16, TILE_K=16, TILE_N=32
2. sizeof(atile) = 1KB, sizeof(btile) = 128 bytes, sizeof(ctile) = 2KB
3. Total size of local variables = 1KB + 2KB + 128 bytes = 3200 bytes < 4KB
4. These three local variables are allocated on GRF since none of the tile addresses escape (address escape means compiler cannot determine the lifetime of the local variable in all cases e.g. address of these arrays are not copied to other variables or passed as arguments)

Asynchronous Offload Support

- An API based offload solution
- By annotating functions with __declspec(target(gfx_kernel))
- Above annotation creates the named kernel functions (Kernel entry points)
- Non-blocking API calls from CPU until the first explicit wait is specified.
- GFX kernels are enqueued into an in-order gpgpu queue
- Explicit control over data transfer, data decoupled from Kernel, data persistence across multiple kernel executions.
- Compiler just generates the GFX code.
- User to explicitly program the host version of offload section

Asynchronous offload – Vector addition

Host Code

```
float *a = new float[TOTALSIZE];
float *b = new float[TOTALSIZE];
float *c = new float[TOTALSIZE];
float *d = new float[TOTALSIZE];

a[0:TOTALSIZE] = 1;
b[0:TOTALSIZE] = 1;
c[0:TOTALSIZE] = 0;
d[0:TOTALSIZE] = 0;

_GFX_share(a, sizeof(float)*TOTALSIZE);
_GFX_share(b, sizeof(float)*TOTALSIZE);
_GFX_share(c, sizeof(float)*TOTALSIZE);
_GFX_share(d, sizeof(float)*TOTALSIZE);

_GFX_enqueue("vec_add", c, a, b, TOTALSIZE); // Non-blocking offload
_GFX_enqueue("vec_add", d, c, a, TOTALSIZE); // Place next kernel in
                                             // in-order queue
_GFX_wait();                                 // wait for all tasks

_GFX_unshare(a);
_GFX_unshare(b);
_GFX_unshare(c);
_GFX_unshare(d);
```

GPU Code

```
__declspec(target(gfx_kernel))
void vec_add(float *res, float *a, float *b, int size){
    cilk_for (int i = 0; i < size; i++)
    {
        res[i] = a[i] + b[i];
    }
    return;
}
```

In 16.0 : Adding (some) OpenMP* 4.0 Offload Support

```
bool Sobel::execute_offload()
{
    int w = COLOR_CHANNEL_NUM * image_width;
    float *outp = this->output;
    float *img = this->image;
    int iw = image_width;
    int ih = image_height;

#pragma omp target map(to: ih, iw, w) \
                   map(tofrom:  img[0:iw*ih*COLOR_CHANNEL_NUM], \
                                outp[0:iw*ih*COLOR_CHANNEL_NUM])
#pragma omp parallel for collapse(2)
    for (int i = 1; i < ih - 1; i++) {
        for (int k = COLOR_CHANNEL_NUM; k < (iw - 1) * COLOR_CHANNEL_NUM; k+
+) {
            float gx = 1 * img[k + (i - 1) * w -1 * 4]
                     + 2 * img[k + (i - 1) * w +0 * 4]
                     + 1 * img[k + (i - 1) * w +1 * 4]
                     - 1 * img[k + (i + 1) * w -1 * 4]
                     - 2 * img[k + (i + 1) * w +0 * 4]
                     - 1 * img[k + (i + 1) * w +1 * 4];
            float gy = 1 * img[k + (i - 1) * w -1 * 4]
                     - 1 * img[k + (i - 1) * w +1 * 4]
                     + 2 * img[k + (i + 0) * w -1 * 4]
                     - 2 * img[k + (i + 0) * w +1 * 4]
                     + 1 * img[k + (i + 1) * w -1 * 4]
                     - 1 * img[k + (i + 1) * w +1 * 4];
            outp[i * w + k] = sqrtf(gx * gx + gy * gy) / 2.0;
        }
    }

    return true;
}
```

Optimization Notice

Usability:

- Only a subset is supported
- ‘tofrom’ and ‘to’ maps to ‘pin’
- -qopenmp-offload=gfx must be used to change the compiler default omp target from MIC to GFX

gfx_sys_check tool

- Checks current platform and S/W configuration for Gen offload capability
- Reports platform configuration details critical for user support
- Works on all supported platforms
- Distributed together with the compiler
- -v for more details

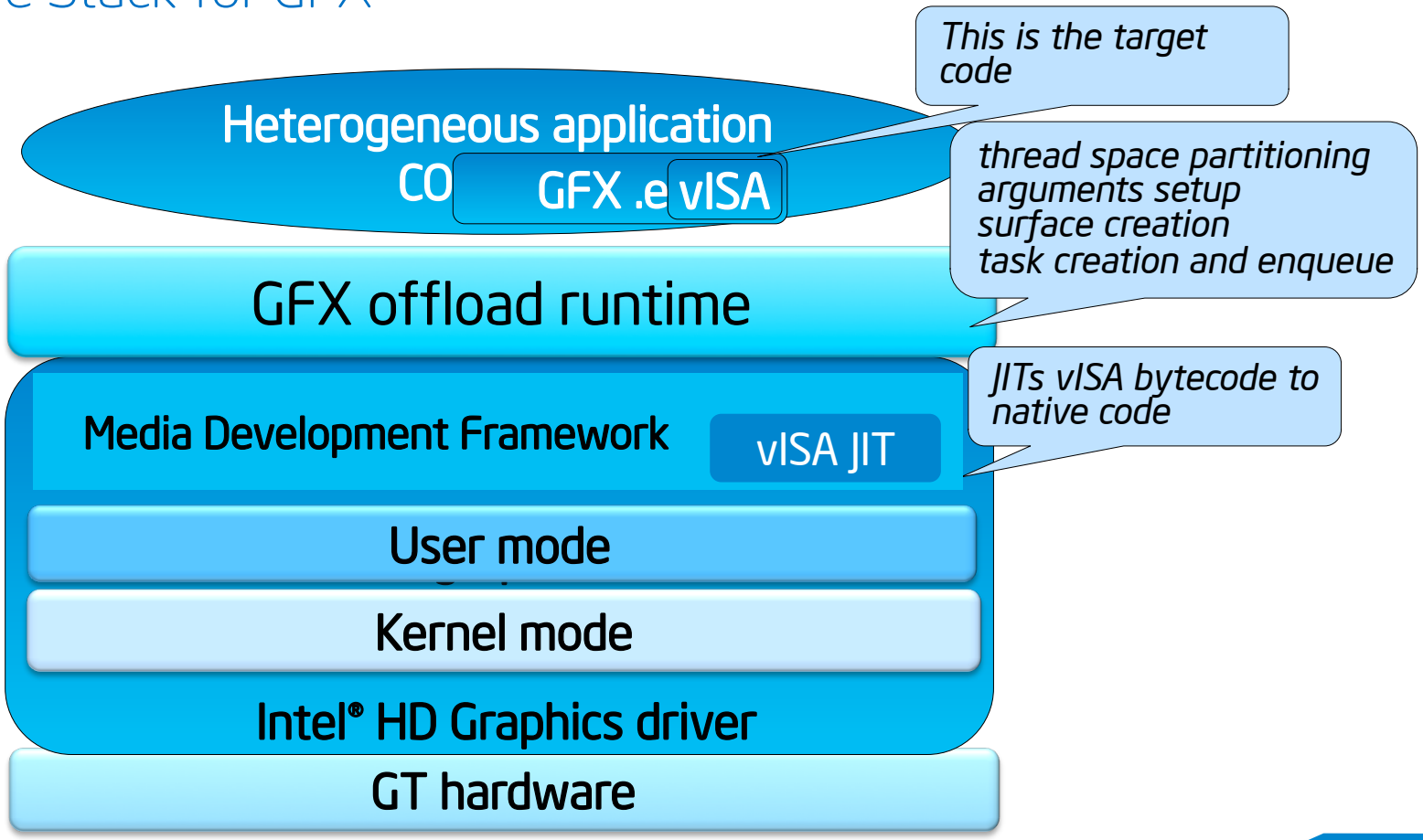
```
% gfx_sys_check -v

Checking CPU
 processor signature: 306a9
       extended family: 0
        extended model: 3
                  type: 0
           family code: 6
          model number: a
           stepping ID: a

Checking OS
 Linux | 3.8.0-23-generic | #34~precise1-Ubuntu SMP Wed May 29 21:12:31
UTC 2013 | x86_64

Checking display
 DISPLAY variable is not set, using ':0'
 X11 mode
libva info: VA-API version 0.34.0
libva info: va_getDriverName() returns 0
libva info: Trying to open /usr/lib/x86_64-linux-gnu/dri/
iHD_drv_video.so
libva info: Found init function __vaDriverInit_0_32
libva info: va_openDriver() returns 0
 VA version: 0, 34
 Driver description: 16.3.2.1.27925-ubit

Checking Cm, executing test program
 cm_version=300
  RT Dll version: (3.0.0.1017)
 JIT Dll version: (3.0.0.1017)
 CAP_KERNEL_COUNT_PER_TASK=4
. . .
 CAP_GPU_CURRENT_FREQUENCY=0
 executing vISA 3.0 CmRT reported execution time 29360 nanosec
 skipped executing vISA 3.1
```

Limitations of Offload Models

- Main language restrictions
  - No exceptions, RTTI, longjmp/setjmp, VLA, variable parameter list, indirect control flow (virtual functions, function pointers, indirect calls and jumps)
  - No shared virtual memory (in 15.0, will be added in 16.0)
  - No pointer or reference typed globals
  - No OpenMP* or Intel® Cilk™ Plus tasking
- Runtime limitations
  - No ANSI C runtime library except math SVML library.
  - Inefficient 64-bit float and integer (due to HW limitations)

Code generation options

Software Stack for GFX



Native Code Generation for Processor Graphics

- Initial one time JITing overhead for each offload region can be avoided.
- For those customer who know their targets in advance.
- Compiler option used for targeting processor graphics is –mgpu-arch=arch (on Linux*) and /Qgpu-arch:arch (on Windows*).

ISA target value	Intel® Core™ Processors	Intel® Pentium® Processors	Intel® Celeron® Processors
ivybridge	3rd Generation Intel® Core™ Processors	Processor Model Numbers: 20xx	Processor Model Numbers: 10xx
haswell	4th Generation Intel® Core™ Processors	Processor Model Numbers: 3xxx	Processor Model Numbers: 29xx

Debugger Support

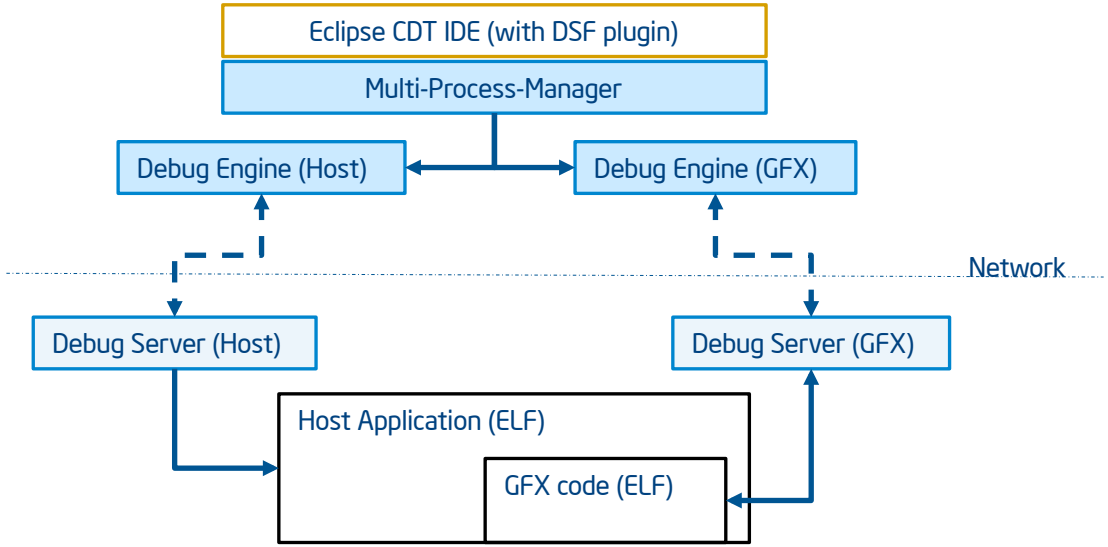
System Overview for Debugger

Single IDE Debugging

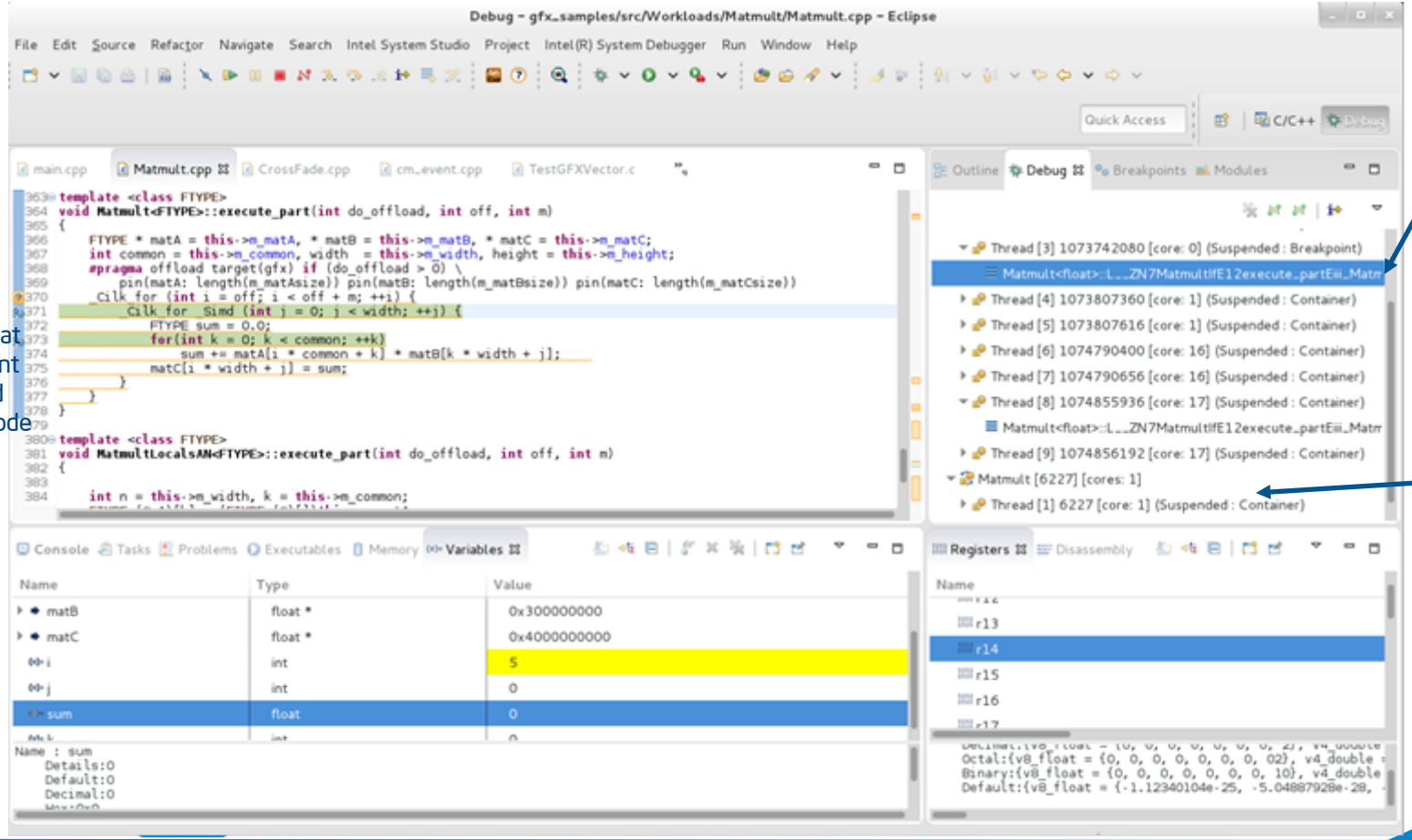
- Debugging starts with host code
- Seamless addition of GFX threads as they become available
- Switching between host and GFX code during single session

Remote Debugging

- Since debugging on the GPU may block screen output, remote debugging is required



Usage Details



Stopped at breakpoint In shared source code

GFX threads

Host thread

Limitations

Initial Limitations:

- No call stack for GFX code
- No display of memory based variables

Current Hardware Limitations:

- Running GFX code is not interruptible, it needs to exit or hit a breakpoint
- No data breakpoints

Other:

- Performance drop of executed code

Target Hardware and OS support

Target Hardware Support

Compiler 15.0:

3rd and 4th Generation Intel® Core™ Processors. These processors comes with either Intel® HD Graphics, Intel® Iris™ Graphics or Intel® Iris™ Pro Graphics.

- Intel® Pentium® Processors with processor model numbers 20xx and 3xxx.
- Intel® Celeron® Processors with processor model numbers 10xx and 29xx.
- Intel® Xeon® Processor E3-1285 v3 and E3-1285L v3 (Intel® C226 Chipset) with Intel® HD Graphics P4700
- For more information on the processors, please refer to http://ark.intel.com

Compiler 16.0:

- Plus 5th Generation Intel® Core™ Processers – all models with Intel® HD Graphics and Intel® Iris Graphics Shared local memory
- Shared virtual memory support as provided by Gen 8 hardware
- Not (!!) all in initial beta release – will be added in later beta update like support for Linux*

Target OS support

15.0

- Windows* 32/64 bit. On Windows* 7 (DX9) requires an active display, batch jobs are not supported. Above restriction is relaxed in Windows* 8 and Windows Server 2012*
- Linux* 64 bit
  - Ubuntu 12.04 (Linux kernel numbers: 3.2.0-41 for 3rd generation Intel® Core™ Processors and 3.8.0-23 for 4th generation Intel® Core™ Processors)
  - SLES11 SP3 (Linux kernel numbers: 3.0.76-11 for both 3rd and 4th generation Intel® Core™ Processors)
- No OS X* and Android* support as of now.

16.0:

- Plus CentOS 7.0 / Redhat 7.0 (not in initial beta – will be added later)

Comparison to OpenCL

OpenCL	GFX offload
Detecting platform and devices in each platform using clGetPlatformIDs() and clGetDevicesIDs() APIs. Supports this feature because OpenCL support in different GPUs shipped by different vendors.	Detection of integrated GPU is done by Intel® Graphics Technology offload runtime automatically.
The program to be run on GPU needs to be put in separate compilation unit (.cl) and build explicitly for target GPU from application host code using clCreateProgramWithSource() and clBuildProgram() APIs.	As long as the source code is annotated right with previously mentioned compiler hints, the existing code is ready to be run on integrated GPU.
Explicitly a command queue for the GPU using clCreateCommandQueue() API.	The command queue is created by the GFX offload runtime.
Kernels have to created from the program build for target using clCreateKernel() API at runtime.	The kernels are automatically created in both Synchronous offload and Asynchronous offload during the build process and not during the runtime.
Kernel invocation is not intuitive rather clEnqueueNDRangeKernel() API is used with the kernel name, number of work groups and number of work items/work group as parameters.	No code change required at the call site. Handled by GFX offload runtime.
Kernel function arguments are passed using clSetKernelArg() API	No code change required at the call site. Handled by GFX offload runtime.
Result from GPU after kernel computation is copied on to host memory using clEnqueueReadBuffer() API.	Synchronous offload – Copy of values are handled by the GFX runtime Asynchronous offload – Explicit copy from GPU to host required.
Explicitly release all kernel objects, memory objects, command queue, program and context object using clReleaseKernel(), clReleaseMemObject(), clReleaseCommandQueue(), clReleaseProgram() and clReleaseContext() APIs.	The release of task objects and buffer objects are done by the GFX runtime automatically.
Portable solution. Works with any GPU.	Non-portable solution. Works only for integrated GPU.

References

- GFX H/W specs documentation - https://01.org/linuxgraphics/documentation
- Intel® C++ Compiler 15.0 User's Guide documentation
  - And documents of 16.0 beta compiler (release notes, user guide)
- Intel® Cilk™ Plus Webpage – http://cilkplus.org
- Vectorization Essentials using Intel® Cilk™ Plus - https://software.intel.com/en-us/articles/vectorization-essentials
- Getting Started with compute offload for Intel® Graphics Technology
- How to offload computation to Intel® Graphics Technology
- Code generation options for Intel® Graphics Technology

Legal Disclaimer & Optimization Notice

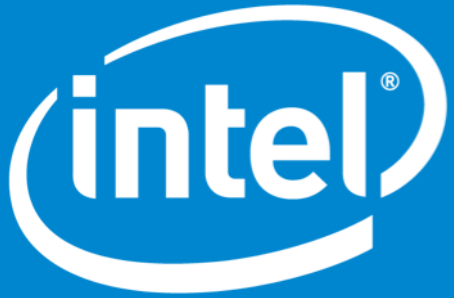
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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