AMD **Accelerated** Parallel Processing TECHNOLOGY

AMD APP SDK v2.4

OpenCL FAQ

1 General Questions

1. **How often can I expect to get AMD APP SDK updates?**

   Developers can expect that the AMD APP SDK may be updated, on average, once per quarter. Actual release intervals may vary depending on available new features and product updates. AMD is committed to providing developers with regular updates to allow them to take advantage of the latest developments in AMD APP technology.

2. **What is the difference between the CPU and GPU components of OpenCL that are bundled with the AMD APP SDK?**

   The CPU component uses the compatible CPU cores in your system to accelerate your OpenCL compute kernels; the GPU component uses the compatible GPU cores in your system to accelerate your OpenCL compute kernels.

3. **What CPUs does the AMD APP SDK v2.4 with OpenCL 1.1 support work on?**

   The CPU component of OpenCL bundled with the AMD APP SDK works with any x86 CPU with SSE3 or later, as well as SSE2.x or later. AMD CPUs have supported SSE3 (and later) since 2005. Some examples of AMD CPUs that support SSE3 (or later) are the AMD Athlon™ 64 (starting with the Venice/San Diego steppings), AMD Athlon™ 64 X2, AMD Athlon™ 64 FX (starting with San Diego stepping), AMD Opteron™ (starting with E4 stepping), AMD Sempron™ (starting with Palermo stepping), AMD Phenom™, AMD Turion™ 64, and AMD Turion™ 64 X2.

4. **What GPUs does the AMD APP SDK v2.4 with OpenCL 1.1 support work on?**

   For the exact list of supported GPUs, see the AMD APP SDK v2.4 System Requirements list at: http://developer.amd.com/stream .

5. **Can my OpenCL code run on GPUs from other vendors?**

   At this time, AMD does not plan to have the AMD APP SDK support GPU products from other vendors; however, since OpenCL is an industry standard programming interface, programs written in OpenCL 1.1 can be recompiled and run with any OpenCL-compliant compiler and runtime.

6. **Can my OpenCL code run on integrated graphics?**

   For the exact list of supported GPUs, see the AMD APP SDK v2.4 System Requirements list at: http://developer.amd.com/stream .

7. **What version of MS Visual Studio is supported?**

   The AMD APP SDK v2.4 with OpenCL 1.1 supports Microsoft® Visual Studio 2008 Professional Edition and Microsoft® Visual Studio 2010 Professional Edition.

8. **Is it possible to run multiple AMD APP applications (compute and graphics) concurrently?**

   Multiple AMD APP applications can be run concurrently, as long as they do not access the same GPU at the same time. AMD APP applications that attempt to access the same GPU at the same time are automatically serialized by the runtime system.

9. **Which graphics driver is required for the current AMD APP SDK v2.4 with OpenCL 1.1 CPU support?**

   For the required graphics driver, see the AMD APP SDK v2.4 System Requirements list at: http://developer.amd.com/stream .

10. **How does OpenCL compare to other APIs and programming platforms for parallel computing, such as OpenMP and MPI? Which one should I use?**

    OpenCL is designed to target parallelism within a single system and provide portability to multiple different types of devices (GPUs, multi-core CPUs, etc.). OpenMP targets multi-core CPUs and SMP systems. MPI is a message passing protocol most often used for communication between nodes; it is a popular parallel programming model for clusters of machines. Each programming model has its advantages. It is anticipated that developers mix APIs, for example programming a cluster of machines with GPUs with MPI and OpenCL.

11. **If I write my code on the CPU version, does it work on the GPU version, or do I have to make changes.**

    Assuming the size limitations for CPUs is considered, the code works on both the CPU and GPU components. Performance tuning, however, is different for each.

12. **Why are there multiple different languages in the AMD APP SDK?**

    One goal of the APP SDK is to provide a full software stack for many different types of programmers. That means if you want performance, AMD provides CAL/IL to do that. If you want to program in the same language across multiple devices from the same source, OpenCL.

13. **What is the precision of mathematical operations?**

    See Chapter 7, “OpenCL Numerical Compliance,” of the OpenCL 1.1 Specification for exact mathematical operations precision requirements.

    http://developer.amd.com/support/KnowledgeBase/Lists/KnowledgeBase/DispForm.aspx?ID=88 .

14. **Are byte-addressable stores supported?**

    Byte-addressable stores are supported.

15. **Are long integers supported?**

    Yes, 64-bit integers are supported.

16. Are operations on vectors supported?

Yes, operations on vectors are supported.

17. Is swizzling supported?

Yes, swizzling (the rearranging of elements in a vector) is supported.

2 Optimizations

18. How do I use constants in a kernel for best performance?

For performance using constants, highest to lowest performance is is achieved using:

- Literal values
- Constant pointer with compile time constants indexing.
- Constant pointer with runtime constant indexing that is the same for all threads.
- Constant pointer with linear access indexing that is the same for all threads.
- Constant pointer with linear access indexing that is different between threads.
- Constant pointer with random access indexing.

19. Why are literal values the fastest way to use constants?

Up to 96 bits of literal values are embedded in the instruction; thus, in theory, there is no limit on the number of useable literals. In practice, the limit is 16K unique literals in a compilation unit.

20. Why does `a * b + c` not generate a mad instruction?

The computation of a*b + c has one rounding after the multiply and another after the addition. Depending on the hardware and the floating point precision, the mad function may round differently, possibly leading to unexpected results.

3 OpenCL Questions

21. What is OpenCL?

OpenCL™ (Open Computing Language) is the first truly open and royalty-free programming standard for general-purpose computations on heterogeneous systems. OpenCL lets programmers preserve their expensive source code investment and easily target both multi-core CPUs and the latest GPUs, such as those from AMD.

Developed in an open standards committee with representatives from major industry vendors, OpenCL gives users a cross-vendor, non-proprietary solution for accelerating their applications on their CPU and GPU cores.

22. How much does the AMD OpenCL development platform cost?

AMD bundles support for OpenCL as part of its AMD APP SDK product offering. The AMD APP SDK is offered to developers and users free of charge.

23. What operating systems does the AMD APP SDK v2.4 with OpenCL 1.1 support?

AMD APP SDK v2.4 runs on 32-bit and 64-bit versions of Windows and Linux. For the exact list of supported operating systems, see the AMD APP SDK v2.4 System Requirements list at: http://developer.amd.com/stream .

24. Can I write an OpenCL application that works on both CPU and GPU?

Applications that program to the core OpenCL 1.1 API and kernel language should be able to target both CPUs and GPUs. At runtime, the appropriate device (CPU or GPU) must be selected by the application.

25. Does the AMD OpenCL compiler automatically vectorize for SSE on the CPU?

The CPU component of OpenCL that is bundled with the AMD APP SDK takes advantage of SSE3 instructions on the CPU.

26. Does the AMD APP SDK v2.4 with OpenCL 1.1 support work on multiple GPUs (ATI CrossFire)?

OpenCL applications can explicitly invoke separate compute kernels on multiple compatible GPUs in a single system. The partitioning of the algorithm to multiple parallel compute kernels must be done by the developer. It is recommended that ATI CrossFire be turned off in most system configurations so that AMD APP applications can access all available GPUs in the system.

ATI CrossFire technology allows multiple AMD GPUs to work together on a single graphics-rendering tasks. This method does not apply to AMD APP computational tasks because it is not compatible with the compute model used for AMD APP applications.

27. Can I ship pre-compiled OpenCL application binaries that work on either CPU or GPU?

By using OpenCL runtime APIs, developers can write OpenCL applications that can detect the available compatible CPUs and GPUs in the system. With this runtime information, applications can dynamically choose on which processors to invoke compute kernels. This lets developers pre-compile applications into binaries that dynamically work on either CPUs or GPUs.

28. Is the OpenCL double precision optional extension supported?

Double precision, an optional OpenCL 1.1 feature, is not supported in the current release of OpenCL bundled with the AMD APP SDK v2.4. An AMD vendor extension for double-precision arithmetic is supported. See the appropriate knowledge base article at:

29. Is it possible to write OpenCL code that scales transparently over multiple devices?

For OpenCL programs that target only multi-core CPUs, scaling can be done transparently; however, scaling across multiple GPUs requires the developer to explicitly partition the algorithm into multiple compute kernels, as well as explicitly launch the compute kernels onto each compatible GPU.

30. What should I do if I get wrong results on the Apple platform with AMD devices?

Apple handles support for the Apple platform; please contact them.

31. Is it possible to dynamically index into a vector?

No, this is not possible because a vector is not an array, but a representation of a hardware register.

32. What is the difference between `local int a[4]` and `int a[4]`?

`local int a[4]` uses hardware local memory, which is a small, low-latency, high-bandwidth memory; `int a[4]` uses per-thread hardware scratch memory, which is located in uncached global memory.

33. Why does using a barrier cause the max kernel work-group size to drop to 64 on HD4XXX chips?

The supported HD4XXX chips do not have a hardware barrier, so the OpenCL runtime cannot execute more than a single wavefront per group to satisfy the OpenCL memory consistency model.

34. How come my program runs slower in OpenCL than in CUDA/Brook+/IL?

When comparing performance, it is better to compare code optimized for our OpenCL platform against code optimized against another vendor's OpenCL platforms. By comparing the same toolchain on different vendors, you can find out which vendors hardware works the best for your problem set.

35. Why can I not use texture on RV7XX devices in OpenCL?

RV7XX devices do not support all of the texture modes and precision requirements that OpenCL requires. Since textures are mapped to images in OpenCL and is an “all or nothing” approach, we do not support images on RV7XX devices; thus, there is no access to textures.

36. Why do read-write images not exist in OpenCL?

OpenCL has a memory consistency model that requires certain constraints (see the *OpenCL Specification* for more information). Since images are special functional hardware units, they are different for reading and writing. This is different from pointers, which for the most part use the same hardware units and can gaurantee the consistency that OpenCL requires.

37. Does prefetching work on the GPU?

No, prefetch on the GPU is a no-op because the GPU does not support pre-fetching.

38. How do you determine the max number of concurrent work-groups?

The maximum number of concurrent work-groups is determined by resource usage. This includes number of registers, amount of LDS space, and number of threads per work group. There is no way to directly specify the number of registers used by a kernel. Each SIMD has a 64-wide register file, with each column consisting of 256 x 32 x 4 registers.

39. Is it possible to tell OpenCL not to use all CPU Cores?

Yes, use the device `fission` extension.

4 OpenCL Optimizations

40. What is more efficient, the ternary operator `?:` or the `select` function?

The select function to the single cycle instruction, `cmov_logical`; in most cases, `?:` also compiles to the same instruction. In some cases, when memory is in one of the operands, the `?:` operator is compiled down to an IF/ELSE block. An IF/ELSE block takes more than a single instruction to execute.

41. What is the difference between 24-bit and 32-bit integer operations?

24-bit operations are faster because they use floating point hardware and can execute on all compute unts. Many 32-bit integer operations also run on all stream processors, but if both a 24-bit and a 32-bit version exist for the same instruction, the 32-bit instruction executes only one per cycle.

5 Hardware Information

42. How are 8/16-bit operations handled in hardware?

The 8/16-bit operations are emulated with 32-bit registers.

43. Do 24-bit integers exist in hardware?

No, there are 24-bit instructions, such as `MUL24`/`MAD24`, but the smallest integer in hardware registers is 32-bits.

44. What are the benefits of using 8/16-bit types over 32-bit integers?

8/16-bit types take less memory than a 32-bit integer type, increasing the amount of data you are able to load with a single instruction. The OpenCL compiler up-converts to 32-bits on load and down-converts to the correct type on store.

45. What is the difference between a GPR and a shared register?

A GPR and a shared (not scratch) register are physically equivalent; the difference is whether the register offset in the hardware is absolute to the register file or relative to the wavefront ID.

46. How often are wavefronts created?

Wavefronts are created by the hardware to execute as long as resources are available. If they are created but cannot execute immediately, they are put in a wait queue where they stay until currently running wavefronts are finished.

47. What is the maximum number of wavefronts?

The maximum number of wavefronts is determined by which resource limits the number of wavefronts that can be spawned. This can be the number of registers, amount of local memory, required stack size, or other factors. Compute shader with local memory usage has a hard cap at 16 wavefronts.

48. Why do I get blue or black screens when executing longer running kernels?

The GPU is not a preemptable device. If you are running the GPU as your display device, ensure that a compute program does not use the GPU past a certain time limit set by Windows. Exceeding the time limit causes the watchdog timer to trigger; this can result in undefined program results.

49. What is the cost of a clause switch?

In general, the latency of a clause switch is around 40 cycles.

50. How can I hide clause switch latency?

By executing multiple wavefronts in parallel.

51. How can I reduce clause switches?

Clause switches are almost directly related to source program control flow. By reducing source program control flow, clause switches can also be reduced.

52. How does the hardware execute a loop on a wavefront?

The loop only ends execution for a wavefront once every thread in the wavefront breaks out of the loop. Once a thread breaks out of the loop, all of its execution results are masked, but the execution still occurs.

53. How does flow control work with wavefronts?

There are no flow control units for each individual thread, so the whole wavefront must execute the branch if any thread in the wavefront executes the branch. If the condition is false, the results are not written to memory, but the execution still occurs.

54. What is the constant buffer size on GPU hardware?

64 kB.

55. What happens with out-of-bound memory operations?

Writes are dropped, and reads return a pre-defined value.

56. Why does 64x1 give bad performance in compute shaders?

One of the reasons is because of how the caches are setup on RV7XX devices. The caches are optimized to work in a tiled mode, not in linear mode (which is the mode compute shaders use). To get optimal cache re-use from the texture in compute shader mode on RV7XX devices, reblock your thread IDs. A16x4, 8x8, or 4x16 should give you good enough blocking to get similiar cache performance as your pixel shader kernel. This is because a cacheline can be thought of as a 4x2 block of data coming in at once. So, for pixel shaders, 64 threads are blocked in a 8x8 block that uses exactly eight cache lines. For compute shaders, your 64x1 block pattern uses 16 cache lines, but only uses half the data in each cache line.

57. What is unique about the LDS in HD4XXX devices, and what are its performance characteristics?

The LDS in the HD 4XXX devices is an owner's write model with limited applications. When used correctly, it has very similiar performance characteristics to the L1 cache, but the user

gains control over what data exists in the memory. The LDS_Transpose sample in the SDK uses the LDS in the HD 4XXX devices very efficiently.

58. How best to think about the graphics cards?

The RV670 (Radeon 3870), has four SIMDs, which are in a simd array. Inside each SIMD are 16 ALU processors (or shader processing units, SPUs), grouped into quads. Each of these ALU processors is composed of five-wide scalar processors (or four-element vectors + one scalar), easily labeled as X, Y, Z, W, and T. X, Y, Z, and W are the elements of the vector; T is the scalar. Each SIMD processes 16 elements per cycle over four cycles, giving a granularity of 64 elements on the RV670. The wavefront size, which is different from chip to chip, is this granularity, and all instructions in a kernel are executed on a group of elements at the wavefront size. This is why it is bad to have flow control in a shader that has a branching pattern different from the wavefront granularity; in this case, part of the wavefront goes down one branch, another part goes down the other branch, and no optimizations can be done because both branches are executed for all threads. When you specify the launch dimensions of the execution domain, the GPU launches a wavefront for every 64 elements to be processed. If the final element does not fill a wavefront, it is launched partially full, possibly loosing some performance. When launched, wavefronts are scheduled in pairs to the SIMD: thread A and thread B then operate on their respective elements in parallel, alternating between using ALU and texture clauses.

6 IL Information

59. How can shared registers be used?

Shared registers can only be used in ALU instructions and not texture/memory instructions. An example would be add `sr1, sr0, sr2`, which takes the value in sr0 and adds it to the value in sr2 and stores it in sr1. This is an atomic instruction.

60. How is LDS different between RV7XX and Evergreen/Northern Islands IL?

In RV7XX IL, you can only have a single LDS buffer. In later architectures, you can have multiple LDS buffers, with each given a specific ID. The compiler then handles the relative offsets for the developers, so that the developer can access each LDS buffer with the same register, but the actual hardware location is different.

For example:

```
dcl_lds_id(0) 1024
dcl_lds_id(1) 1024
lds_load_id(0) r0, r0.0
lds_load_id(1) r1, r0.0
```

The above IL snippet reads address values from 0 and data from 1024 of the LDS memory, even though the offset to the load instruction is the same.

61. What is the most efficient way of writing to memory?

The most efficient method is to use UAVs, which were introduced with Evergreen GPUs and a single UAV was back-ported to be supported on RV7XX devices. A UAV stands for an unordered access view and can be treated equivalent to a C language array that is declared as `int uav0[1 << 32 - 1]`.

62. Does a single UAV work in all cases where a global buffer works?

Although a single UAV is mapped to the global buffer on RV7XX series of chips, it should work for the RV670 chip as well; however, this is not supported and is not guaranteed to work.

63. What does the 'outline' field of a macro mean?

Normally, a macro is inlined into the program, but if `outline` is specified, the macro is turned into a function call instead.

64. How are wavefronts in a work-group scheduled?

All wavefronts in a single work-group are scheduled on the same SIMD.

65. How are work-groups scheduled?

Work-groups are scheduled on SIMDs until the SIMD cannot hold any more work-groups. Once more resources become available on a SIMD, another work-group is scheduled on that SIMD.

66. Can a work-group contain multiple wavefronts?

Yes, a work-group can contain more than a single wavefront. A wavefront is a hardware construct, a work-group is a software construct and it is not a 1-to-1 correspondence.

67. If the work-group is larger than the wavefront size, what does a barrier do?

If the work-group size is larger than the wavefront size, then the work-group consists of multiple wavefronts. The first wavefront hits the barrier and waits until all other wavefronts in the work-group hit the barrier before continuing execution.

68. What happens if the work-group is half the size of a wavefront?

The second half of the wavefront is marked as inactive and no execution that causes side effects or memory accesses occurs.

69. Is LDS memory persistent?

Within a single kernel execution, LDS memory is persistent; however, between kernel executions, memory peristency is not gauranteed.

70. Are there any examples showing blocking in compute shader mode at the IL level?

Yes, look at the lds_transpose sample and the *LDS_Transpose.pdf*.

71. How do I dump IL?

Set the env variable `GPU_DUMP_DEVICE_KERNEL` to 1.

72. How do I bind a UAV Arena?

There are now 1024 arena UAV IDs that are supported; however, they are virtual UAV IDs used to specify to the compiler which instructions correspond to the different memory segments. They all alias to the same hardware UAVs (8-10), which support dword, short, and byte access, respectively. UAVs 9 and 10 are set up for you when you bind UAV 8, so you only need to bind UAV 8. For example, if you have a kernel with three different pointers, you

can create three arena UAVs in the IL; however, they all must point to the same memory resource, but different offsets, 256 byte aligned, within that memory resource. This lets the compiler optimize the different segments of memory differently. So, if segment 1 uses bytes and dwords, segment 2 uses dwords, then segment 1 memory access uses the complete path, and segment 2 access uses the fast path.

73. Can a compute shader outperform a pixel shader?

Yes, it is possible for equivalent kernels to perform better in compute shader than in pixel shader mode. The reason for this is that the pixel shader is part of the graphics pipeline and must share resources with other shader modes. This is not true for compute shaders, which can use the resources from the whole device.

74. When is a shared register access atomic?

A shared register access is only atomic within a single instruction.

75. Is it possible to use all 256 register in a thread?

No, the compiler limits a wavefront to half of the register pool, so there can always be at least two wavefronts executing in parallel.

7 ISA Information

76. What devices is the assembler supported on?

The assembler is only supported on devices that contain the R6XX devices.

77. What are registers R123-R127/T0-T4?

Register numbers R123-R127/T0-T4 are temporary registers, called clause temps, that do not affect register allocation. They are shared between all wavefronts on a SIMD, but are only live for a single ALU CF clause.

78. How do I dump ISA?

Set the env variable `GPU_DUMP_DEVICE_KERNEL` to 2.

8 CAL Information

79. What is CAL?

CAL is a driver layer API similar to OpenGL that provides compute access to the GPU device.

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:
URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Support: developer.amd.com/appsdksupport
Forum: developer.amd.com/openclforum