IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
AUSTIN DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No: 1:17-cv-1148
	§	
vs.	§	PATENT CASE
	§	
ADVANCED MICRO DEVICES, INC.,	§	
	§	
Defendant.	§	
	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against Advanced Micro Devices, Inc. ("Defendant" or "AMD") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Delaware corporation with a place of business at One AMD Place, P.O. Box 3453, Sunnyvale, CA 94088-3453. Defendant may be served with process in this judicial district by serving its registered agent for service of

process: The Corporation Trust Company, Corporation Trust Center, 1209 Orange St., Wilmington, DE 19801.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District. For instance, on information and belief, Defendant has a regular and established place of business at both 7171 Southwest Parkway, Austin, TX 78735 and 1340 Airport Commerce Dr, Suite 500, Austin, TX 78741.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The '546 Patent is valid, enforceable, and was duly issued in full compliance

with Title 35 of the United States Code.

13. On information and belief, Defendant has infringed and continues to infringe one or more claims, including at least Claim 1, of the '546 Patent by making, using, importing, selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the '546 Patent. Defendant has infringed and continues to infringe the '546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software development packages including, without limitation, AMD Accelerated Parallel Processing (APP) SDK, and any similar products ("Product"), which infringe at least Claim 1 of the '546 Patent.

15. The Product is a framework (e.g., a software development kit) that is configured to create embedded software for multiple hardware modules (e.g., versions of a graphic processing unit (GPU), such as ATI Radeon HD 5780 GPU, or a CPU such as the AMD Phenom IIx4 processor or other similar processors). Defendant and/or its customers use the Product to produce embedded software. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

1 Overview

The AMD APP SDK v2.8 is provided to the developer community to accelerate the programming in a heterogeneous environment. The package consists of samples that serve as examples for a wide class of developers on different facets of heterogeneous programming.

The AMD APP SDK package contains the runtime for CPU hardware only. The GPU runtime is included in the Catalyst driver.

There are no changes to the SDK 2.8 Developer or Sample packages installation.

For Microsoft® Windows® platforms, the AMD APP SDK installer installs the following packages on your system by default (unless you choose to customize the install):

1. AMD APP SDK CPU Runtime package.
2. AMD APP SDK Developer package. This includes:
   - the OpenCL™ compiler,
   - pointers to the latest versions of the developer documentation. (See the AMD APP SDK v2 folder in the All Programs panel of Windows Start. This also contains links to the AMD Math Libraries.)
3. AMD APP SDK v2 Samples package. This includes:
   - sample applications,
   - sample documentation.

Source: http://developer.amd.com/download/AMD_APP_SDK_Installation_Notes.pdf

1.2 OpenCL Overview

The OpenCL programming model consists of producing complicated task graphs from data-parallel execution nodes.

In a given data-parallel execution, commonly known as a kernel launch, a computation is defined in terms of a sequence of instructions that executes at each point in an N-dimensional index space. It is a common, though by not required, formulation of an algorithm that each computation index maps to an element in an input data set.

The OpenCL data-parallel programming model is hierarchical. The hierarchical subdivision can be specified in two ways:

- Explicitly - the developer defines the total number of work-items to execute in parallel, as well as the division of work-items into specific work-groups.
- Implicitly - the developer specifies the total number of work-items to execute in parallel, and OpenCL manages the division into work-groups.

OpenCL's API also supports the concept of a task dispatch. This is equivalent to executing a kernel on a compute device with a work-group and NDRange containing a single work-item. Parallelism is expressed using vector data types implemented by the device, enqueuing multiple tasks, and/or enqueuing native kernels developed using a programming model orthogonal to OpenCL.

Source: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf

1.3 Programming Model

The OpenCL programming model is based on the notion of a host device, supported by an application API, and a number of devices connected through a bus. These are programmed using OpenCL C. The host API is divided into platform and runtime layers. OpenCL C is a C-like language with extensions for parallel programming such as memory fence operations and barriers. Figure 1.1 illustrates this model with queues of commands, reading/writing data, and executing kernels for specific devices.

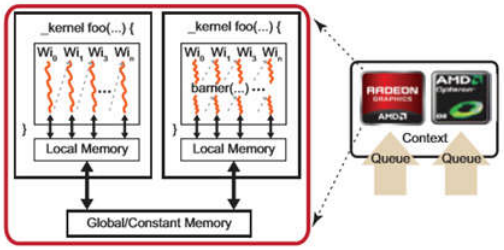


Figure 1.1 OpenCL Programming Model

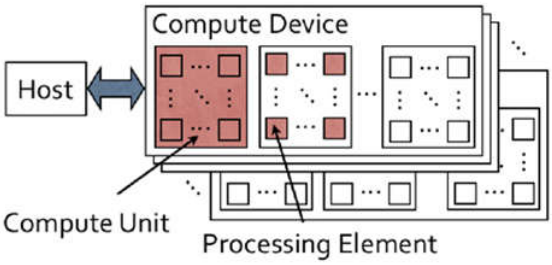


Figure 2: OpenCL Platform Model

Source: http://developer.amd.com/wordpress/media/2013/11/MediaSDK_User_Guide.pdf

The AMD APP SDK CPU runtime installation for Windows adds the variable AMDAPPSDKROOT to your environment. This points to the location where you have installed the SDK development package. The Windows installer also adds the locations of the OpenCL dynamic libraries to your system PATH variable, so applications know where to find it.

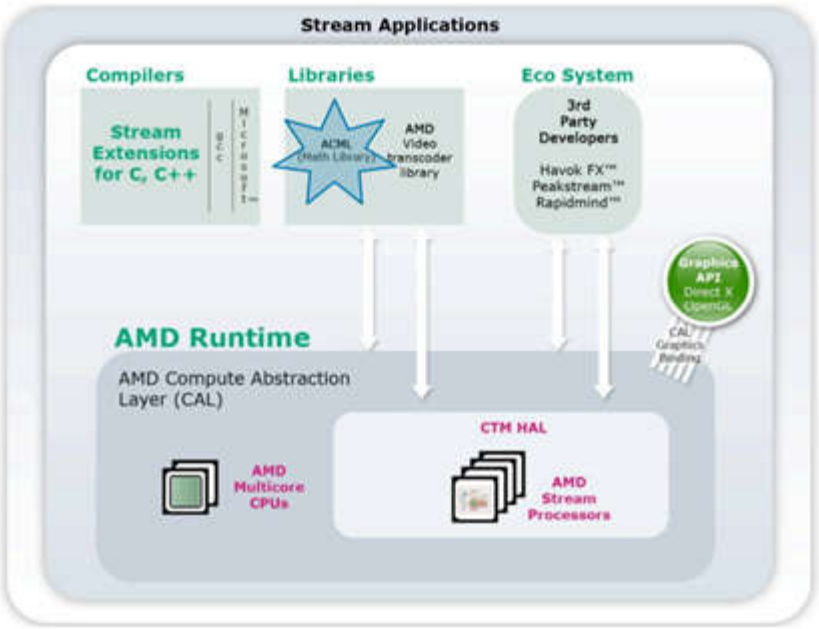
The AMD APP SDK Samples installation includes the following folders:

- bin - This includes pre-built binaries and dynamic libraries for running AMD APP samples.
- lib - This contains AMD APP SDK utility libraries to which sample applications link.
- include - This contains the header files for utilities and tools used by the samples.
- samples - This contains sample applications for OpenCL 1.1 and OpenCL 1.2, C++ AMP, Bolt, and Aparapi.
- make - This contains the definitions and rules for make.

The AMD APP SDK Samples installer for Windows adds the variable AMDAPPSDKSAMPLESROOT to your environment. This points to the location where you have installed the SDK Samples package.

Source: http://developer.amd.com/wordpress/media/2013/11/MediaSDK_User_Guide.pdf

16. The Product provides one or more generic application handler programs (AMD APP SDK provides a Compute Abstraction Layer (CAL) to optimize Graphic Processing Unit (GPU) such as the ATI Radeon™ HD 5870 GPU which further includes a Hardware Abstraction Layer(HAL) for device specific and driver like interface. CAL also includes Compute Kernels to provide data parallelism and generic functions.) The generic programs comprise computer program code for performing generic application functions common to multiple types of hardware modules used in a communication environment (e.g., the generic code provides common and generic functions to multiple hardware modules, such as versions of a graphic processing unit (GPU), such as ATI Radeon HD 5780 GPU, or a CPU such as the AMD Phenom IIx4 processor or other similar processors). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



Source: https://developer.amd.com/wordpress/media/2013/02/07-CTM-overview.pdf

AMD ACCELERATED PARALLEL PROCESSING

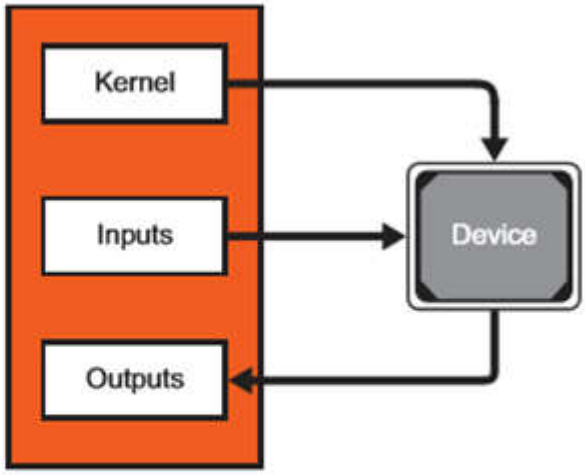
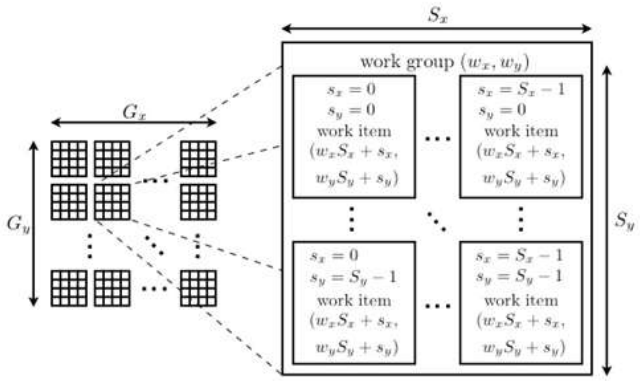


Figure 1.3 CAL Device and Memory

Source:http://developer.amd.com/wordpress/media/2012/10/AMD_CAL_Programming_Guide_v2.0.pdf

Kernels

As mentioned, OpenCL kernels provide data parallelism. The kernel execution model is based on a hierarchical abstraction of the computation being performed. OpenCL kernels are executed over an index space, which can be 1, 2 or 3 dimensional. In Figure 3, we see an example of a 2 dimensional index space, which has Gx * Gy elements. For every element of the kernel index space, a work-item will be executed. All work items execute the same program, although their execution may differ due to branching based on data characteristics or the index assigned to each work-item.



Source: http://developer.amd.com/resources/articles-whitepapers/opencl-and-the-amd-app-sdk/

17. The Product includes generating specific application handler code to associate the generic functions with the specific functions at a device driver for at least one of the types of hardware modules. For example, in addition to the Compute Kernels provided by CAL, AMD APP SDK also includes specific application generic function code that is specific to the application (device Graphic Processing Unit (GPU) such as the ATI Radeon™ HD 5870 GPU, or a CPU, such as the AMD Phenom™ II x4 processor, or other similar processors). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

CAL	Compute Abstraction Layer. A device-driver library that provides a forward-compatible interface to AMD Accelerated Parallel Processing compute devices. This lower-level API gives users direct control over the hardware: they can directly open devices, allocate memory resources, transfer data and initiate kernel execution. CAL also provides a JIT compiler for AMD IL.

1.1 CAL System Architecture

A typical CAL application includes two parts:

- a program running on the host CPU (written in C/C++), the *application*, and
- a program running on the stream processor, the *kernel* (written in a high-level language, such as AMD IL).

The CAL API comprises one or more stream processors connected to one or more CPUs by a high-speed bus. The CPU runs the CAL and controls the stream processor by sending commands using the CAL API. The stream processor runs the kernel specified by the application. The stream processor device driver program (CAL) runs on the host CPU.

Figure 1.2 is a block diagram of the various CAL system components and their interaction. Both the CPU and stream processor are in close proximity to their local memory subsystems. In this figure:

- Local memory subsystem – the CAL local memory. This is the memory subsystem attached to each stream processor. (From the perspective of CAL, the Stream Processor is local, and the CPU is remote.)
- System memory – the single memory subsystem attached to all CPUs.

CPUs can read from, and write to, the system memory directly; however, stream processors can read from, and write to:

Source: http://developer.amd.com/wordpress/media/2012/10/AMD_CAL_Programming_Guide_v2.0.pdf

6.3.2.2 AMD APP SDK example

In the AMD APP SDK sample, `addMul2d` is a generic function that uses generic address spaces for its operands. The function computes the convolution sum of two vectors. Two kernels compute the convolution: one uses data in the `global` address space (`convolution2DUsingGlobal`); the other uses the `local` address space (`sepiaToning2DUsingLocal`). The use of a single function improves the readability of the source.

```
float4 addMul2D (uchar4 *src, float *filter, int2 filterDim, int
width)
{   int i, j;
    float4 sum = (float4)(0);
    for(i = 0; i < (filterDim.y); i++)
    {
            for(j = 0; j < (filterDim.x); j++)
            {
```

Source: http://developer.amd.com/wordpress/media/2013/11/MediaSDK_User_Guide.pdf

18. The Product generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, AMD APP generates system-specific application handler code

by defining a specific element such as functions and data structures corresponding to specific GPU such as the ATI Radeon™ HD 5870 GPU, or a CPU, such as the AMD Phenom™ II x4 processor, or other similar processors. When specific functions are written for handling defined specific elements, the specific functions must be registered. APP SDK accordingly contains data structures that register and embed the required functions. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

1.3 Programming Model

The OpenCL programming model is based on the notion of a host device, supported by an application API, and a number of devices connected through a bus. These are programmed using OpenCL C. The host API is divided into platform and runtime layers. OpenCL C is a C-like language with extensions for parallel programming such as memory fence operations and barriers. Figure 1.1 illustrates this model with queues of commands, reading/writing data, and executing kernels for specific devices.

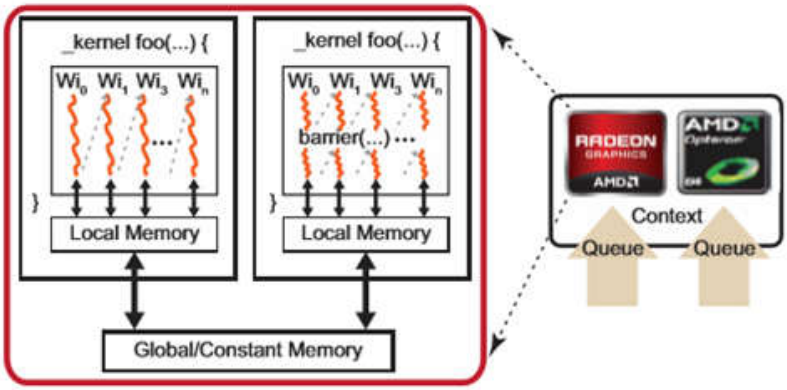


Figure 1.1 OpenCL Programming Model

Source: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf

For example, consider an application that must perform CPU computations in the application thread and also run another kernel on the stream processor. The following code shows one way of doing this.

```
// Launch GPU kernel
```


Copyright © 2010 Advanced Micro Devices, Inc. All rights reserved.


```
CALevent e;
if(calCtxRunProgram(&e, ctx, func, &rect) != CAL_RESULT_OK)
    fprintf(stderr, "Error in run kernel\n");

// Wait for the GPU kernel to finish
while(calCtxIsEventDone(ctx, e) == CAL_RESULT_PENDING);

// Perform CPU operations _after_ the GPU kernel is complete
performCPUOperations();

// Map the output resource to application data pointer
calResMap((CALvoid**)&fdata, &pitch, outputRes, 0);
```

The following code implements the same operations as above, but probably finishes more quickly since it executes the CPU operations in parallel with the stream kernel.

```
// Launch GPU kernel
CALevent e;
if(calCtxRunProgram(&e, ctx, func, &rect) != CAL_RESULT_OK)
```

Source: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf

OpenNI Libraries –

The GestureRecognition AMD APP SDK OpenCV-CL sample makes use of OpenNI libraries to extract video frames. OpenNI framework is an open source SDK used for the development of 3D sensing middleware libraries and applications. The OpenNI SDK can be downloaded from http://structure.io/openni.

You must set the following environment variables:

1. For 32-bit builds, add `OPENNI2_REDIST` to the `PATH` environment variable.
   For 64-bit builds, add `OPENNI2_REDIST64` to the `PATH` environment variable.
2. Header and library paths will be added by the OpenNI Windows installer. If those paths are missing from the system, then the user must set the following environment variables:
   i. For 32-bit platforms:
      Set `OPENNI2_INCLUDE` to `<<OPENNI-INSTALL_PATH>>\Include`
      Set `OPENNI2_LIB` to `<<OPENNI-INSTALL_PATH>>\Redist`
      Add `OPENNI2_LIB` to the `PATH` environment variable
   ii. For 64-bit platforms:
      Set `OPENNI2_INCLUDE64` to `<<OPENNI-INSTALL_PATH>>\Include`
      Set `OPENNI2_LIB64` to `<<OPENNI-INSTALL_PATH>>\Redist`
      Add `OPENNI2_LIB64` to the `PATH` environment variable

The GestureRecognition sample currently works on only Windows platforms.

Source: http://developer.amd.com/wordpress/media/2013/11/MediaSDK_User_Guide.pdf

19. When a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. AMD and/or its customers compile the generic functions using OpenCL compiler and the specific functions are then build using CL Build program supported by APP SDK. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

3.2.2.2 Compiling and linking the program separately

In this method, two separate steps are performed to generate the device executable. First, program objects are compiled by using the **clCompileProgram** API (for details, see the OpenCL specification); then the compiled programs are linked together to generate the final executable by using the **clLinkProgram** API (for details, see the OpenCL specification). This method is particularly useful--and is the only way--to link a previously-compiled program. By using this method, users can link their program objects with external program objects to build the final program object.

Both the APIs support similar options (depends on whether one is compiling or linking) as the options in **clBuildProgram**, to control the compiler and linker. For details about the options supported by each API, see the respective API description section in the OpenCL specification.

Compiling the program –

The user must compile each program object separately. This step may be a little tedious if a source program depends on other header files. In that case, separate program objects corresponding each header file must be created first. Then, during compilation, those header programs must be passed as embedded headers along with the intended program object.

The software includes the following components:

- OpenCL compiler and runtime
- Debugging and Performance Profiling Tools – AMD CodeXL.
- Performance Libraries – clMath and other OpenCL accelerated libraries for optimized NDRange-specific algorithms.

The latest generations of AMD GPUs use unified shader architectures capable of running different kernel types interleaved on the same hardware.

Source: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf

3.2.2 Building the program executable from the program objects

After the program object is created (from either sources or binaries), the program must be built for the targeted devices and the device executables must be generated. The executables are generated mainly in two ways:

- Building (compile and link) the program in a single step (using **clBuildProgram**)
- Compiling and linking the program separately (using **clCompileProgram and clLinkProgram**)

3.2.2.1 Building the program in a single step

The most common way of building program objects, this method uses a single API, **clBuildProgram**, for both compiling and linking the program. For additional details about this API, see the OpenCL specification.

Example(s):

Suppose a program object has been created as follows:

```
cl_program program = clCreateProgramWithSource(context, 1, &source,
&length, NULL);
```

Next, the program object can be built for all the devices in the context or for a list of selected devices.

- To build the program **for all the devices**, "NULL" must be passed against the target device list argument, as shown below:
  ```
  clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
  ```
- To build **for any particular GPU device or a list of devices**.:
  ```
  int nDevices = 0;
  clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 0, NULL,
  &nDevices);
  ```

Source: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_User_Guide2.pdf

20. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

21. Defendant's actions complained of herein are causing irreparable harm and

monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined and restrained by this Court.

22. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees, attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: December 11, 2017

Respectfully submitted,

/s/ Jay Johnson

JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A