The present invention extends to methods, systems, and computer program products for providing domain, hull, and geometry shaders in a para-virtualized environment. As such, a guest application executing in a child partition is enabled use a programmable GPU pipeline of a physical GPU. A vGPU (executing in the child partition) is presented to the guest application. The vGPU exposes DDI(s) of a rendering framework. The DDIs enable the guest application to send graphics commands to the vGPU, including commands for utilizing a domain shader, a hull shader, and/or a geometry shader at a physical GPU. A render component (executing within the root partition) receives physical GPU-specific commands from the vGPU, including commands for using the domain shader, the hull shader, and/or the geometric shader. The render component schedules the physical GPU-specific command(s) for execution at the physical GPU.
UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

** declarations under Rule 4.17:**

— as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(a))
BACKGROUND

1. Background and Relevant Art

[0001] Computer systems and related technology affect many aspects of society. Indeed, the computer system's ability to process information has transformed the way we live and work. Computer systems now commonly perform a host of tasks (e.g., word processing, scheduling, accounting, etc.) that prior to the advent of the computer system were performed manually. More recently, computer systems have been coupled to one another and to other electronic devices to form both wired and wireless computer networks over which the computer systems and other electronic devices can transfer electronic data. Accordingly, the performance of many computing tasks is distributed across a number of different computer systems and/or a number of different computing environments.

[0002] Some computer systems are configured to provide para-virtualized execution environments, which allow guest software to share hardware devices of a single computer system in an isolated manner. Generally, para-virtualized execution environments provide a plurality of partitions, supported by a hypervisor. Each partition comprises an isolated execution environment for software executing therein. The partitions generally include a root partition and one or more child partitions. The root partition runs a host operating system and manages a virtualization stack. The root partition may gain access to physical devices. Each child partition hosts guest software (e.g., guest operating systems and guest applications). Child partitions are provided access to physical devices through virtual devices and software interfaces of the hypervisor.

[0003] Some para-virtualized execution environments provide child partitions (and guest software executing therein) with para-virtualized access to one or more physical graphics processing units ("GPUs"). Each implementation of para-virtualized access to physical GPUs typically supports a single rendering framework. As such, guest software may be unable to access one or more GPU functions of a physical GPU if that guest software is executing within a para-virtualized execution environment that does not support those GPU functions. For example, if the para-virtualized execution environment supports a legacy version of a rendering framework, then guest software executing within the para-virtualized execution environment cannot leverage GPU functions implemented by newer versions of the rendering framework. This is true even though those functions are supported by the physical GPU.
Thus guest software may need to perform a GPU function not supported by the para-virtualized environment in which it is running. When this occurs, the guest software may rely on using a fully virtualized processing unit, may use a translation layer to perform the GPU function, and/or may use an inferior GPU function that is supported by the para-virtualized environment. Any of these solutions can incur potentially significant performance penalties for the guest software. In some cases, the guest software may be unable to perform the GPU function at all, leading to errors and/or decreased functionality of the guest software.

**BRIEF SUMMARY**

The present invention extends to methods, systems, and computer program products for providing domain, hull, and geometry shaders in a para-virtualized environment. As such, embodiments of the invention provide a rich programmable GPU pipeline to guest applications.

Embodiments include a method for providing a programmable GPU pipeline to a guest application executing in a child partition of a para-virtualized execution environment. A virtual machine session is instantiated. The virtual machine session includes a hypervisor that provides (i) a root partition (which has access to a physical GPU), and (ii) a child partition (which executes a guest application).

A virtualized graphics processing unit ("vGPU"), which also executes within the child partition, is presented to the guest application. The vGPU presents device driver interfaces ("DDIs") of a rendering framework to the guest application as part of a user-mode driver ("UMD"). The DDIs provide an API that enables the guest application to send commands to the vGPU. The API enables the guest application to program a GPU pipeline of the physical GPU to utilize one or more features of the rendering framework, including utilizing a domain shader, a hull shader, and/or a geometric shader.

A render component, which executes within the root partition, receives one or more physical GPU-specific commands from the vGPU. The physical GPU-specific commands are for using the domain shader, the hull shader, or the geometric shader at the physical GPU. The render component schedules the physical GPU-specific command(s) for execution at the physical GPU.

This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.
Additional features and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

**BRIEF DESCRIPTION OF THE DRAWINGS**

In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

**DETAILED DESCRIPTION**

The present invention extends to methods, systems, and computer program products for providing domain, hull, and geometry shaders in a para-virtualized environment. As such, embodiments of the invention provide a rich programmable GPU pipeline to guest applications.

Embodiments include a method for providing a programmable GPU pipeline to a guest application executing in a child partition of a para-virtualized execution environment. A virtual machine session is instantiated. The virtual machine session includes a hypervisor that provides (i) a root partition (which has access to a physical GPU), and (ii) a child partition (which executes a guest application).

A virtualized graphics processing unit ("vGPU"), which also executes within the child partition, is presented to the guest application. The vGPU presents device driver interfaces ("DDIs") of a rendering framework to the guest application as part of a user-
mode driver ("UMD"). The DDIs provide an API that enables the guest application to send commands to the vGPU. The API enables the guest application to program a GPU pipeline of the physical GPU to utilize one or more features of the rendering framework, including utilizing a domain shader, a hull shader, and/or a geometric shader.

[0017] A render component, which executes within the root partition, receives one or more physical GPU-specific commands from the vGPU. The physical GPU-specific commands are for using the domain shader, the hull shader, or the geometric shader at the physical GPU. The render component schedules the physical GPU-specific command(s) for execution at the physical GPU.

[0018] Embodiments of the present invention may comprise or utilize a special purpose or general-purpose computer including computer hardware, such as, for example, one or more processors and system memory, as discussed in greater detail below. Embodiments within the scope of the present invention also include physical and other computer-readable media for carrying or storing computer-executable instructions and/or data structures. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer system. Computer-readable media that store computer-executable instructions are computer storage media (devices). Computer-readable media that carry computer-executable instructions are transmission media. Thus, by way of example, and not limitation, embodiments of the invention can comprise at least two distinctly different kinds of computer-readable media: computer storage media (devices) and transmission media.

[0019] Computer storage media (devices) includes RAM, ROM, EEPROM, CD-ROM, solid state drives ("SSDs") (e.g., based on RAM), Flash memory, phase-change memory ("PCM"), other types of memory, other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer.

[0020] A "network" is defined as one or more data links that enable the transport of electronic data between computer systems and/or modules and/or other electronic devices. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a transmission medium. Transmissions media can include a network and/or data links which can be used to carry desired program code means in the form of computer-executable instructions or data.
structures and which can be accessed by a general purpose or special purpose computer. Combinations of the above should also be included within the scope of computer-readable media.

[0021] Further, upon reaching various computer system components, program code means in the form of computer-executable instructions or data structures can be transferred automatically from transmission media to computer storage media (devices) (or vice versa). For example, computer-executable instructions or data structures received over a network or data link can be buffered in RAM within a network interface module (e.g., a "NIC"), and then eventually transferred to computer system RAM and/or to less volatile computer storage media (devices) at a computer system. Thus, it should be understood that computer storage media (devices) can be included in computer system components that also (or even primarily) utilize transmission media.

[0022] Computer-executable instructions comprise, for example, instructions and data which, when executed at a processor, cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. The computer executable instructions may be, for example, binaries, intermediate format instructions such as assembly language, or even source code. Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the described features or acts described above. Rather, the described features and acts are disclosed as example forms of implementing the claims.

[0023] Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including, personal computers, desktop computers, laptop computers, message processors, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, tablets, pagers, routers, switches, and the like. The invention may also be practiced in distributed system environments where local and remote computer systems, which are linked (either by hardwired data links, wireless data links, or by a combination of hardwired and wireless data links) through a network, both perform tasks. In a distributed system environment, program modules may be located in both local and remote memory storage devices.
Figure 1 illustrates an example computer architecture 100 that facilitates providing a programmable GPU pipeline to a guest application executing in a child partition of a para-virtualized execution environment. Referring to Figure 1, computer architecture 100 includes physical hardware 102. Physical hardware 102 can include any appropriate hardware devices, such as one or more general purpose processors, system memory, and the like. As depicted, physical hardware 102 includes physical GPU 104, and may include a plurality of physical GPUs.

Physical GPU 104 is a processing device configured to perform parallel processing tasks, such as graphics rendering tasks. Physical GPU 104 supports a variety of GPU functions. Some supported GPU functions are fixed functions, and some supported GPU functions are programmable functions. Programmable functions provide for a programmable rendering pipeline (as opposed to a merely fixed rendering pipeline). A programmable rendering pipeline enables the output of one GPU function to become the input of another GPU function. In particular, physical GPU 104 supports both legacy programmable functions (i.e., functions supported by legacy rendering frameworks) and newer programmable functions (i.e., functions supported by newer rendering frameworks). Legacy programmable functions may include vertex shaders and pixel shaders. Newer programmable functions may include domain shaders, hull shaders, and geometric shaders.

Computer architecture 100 also includes hypervisor 106. Hypervisor 106 executes on top of physical hardware 102 and supports a virtualization platform. The virtualization platform provides a plurality of partitions. Each partition provides a logical unit of isolation, in which guest software executes. As depicted, computer architecture 100 includes root partition 108 and child partition 110.

Root partition 108 executes a host operating system, and has direct access to physical hardware 102 (as depicted by root partition 108 appearing over physical hardware 102). Each child partition provides an execution environment for executing guest software (e.g., operating systems and/or applications) and may access physical hardware 102 indirectly in a para-virtualized manner. That is, each child partition provides one or more software interfaces (e.g., virtualized hardware) to guest software. The guest software, in turn, uses the software interface(s) to access physical hardware 102. Computer architecture 100 can include a plurality of child partitions.

As depicted, guest software 112 executes within child partition 110. Guest software 112 comprises an operating system, and/or an application program executing...
within an operating system. Guest software 112 includes or uses graphics runtime 114. Guest software 112 may be an operating system that includes graphics runtime 114, or may be an application executing within the operating system that uses graphics runtime 114 provided by the operating system. Graphics runtime 114 corresponds to one or more graphics rendering frameworks, and provides APIs which enable guest software 112 to request execution of GPU functions of the supported rendering framework(s) (e.g., various shaders).

Child partition 110 provides guest software 112 access to vGPU 116. vGPU 116 virtualizes physical GPU 104, enabling guest software 112 to indirectly access physical GPU 104. As such, vGPU 116 is configured to expose all, or a subset, of the functionality of rendering frameworks supported by graphics runtime 114 to guest software 112, along with any corresponding functionality of physical GPU 104.

In particular, vGPU 116 is configured to expose one or more software interfaces to guest software 112 that enable guest software 112 to call vGPU 116 for accessing GPU functions of physical GPU 101a. vGPU 106, in turn, works in conjunction with render component 132 in root partition 108 to execute any requested GPU functions on physical GPU 104.

As depicted, vGPU 116 includes user-mode driver 118 and legacy user-mode driver 112 executing in a user-mode of child partition 110. Each user-mode driver exposes one or more device driver interfaces ("DDIs") that enable guest software 112 to indirectly request execution of GPU functions at physical GPU 104 through calls to vGPU 116. Legacy user-mode driver 122 exposes legacy DDIs 124 corresponding to GPU functionality that is supported by a legacy rendering framework. For example, legacy DDIs 124 may include DDIs related to fixed GPU functions, or DDIs related to some programmable GPU functions supported by legacy rendering frameworks, such as vertex shaders and/or pixel shaders. As such, legacy user mode driver 122 of vGPU 116 enables graphics runtime 114 to make use of a legacy rendering framework.

User-mode driver 118, on the other hand, exposes DDIs 120 corresponding to GPU functionality that is supported by one or more newer rendering frameworks, and that was not supported by the legacy rendering framework. For example, DDIs 120 may include DDIs related to additional programmable GPU functions, such as domain shaders, hull shaders, and/or geometric shaders. In some embodiments, DDIs 120 expose all DDIs supported by one or more newer rendering frameworks.
In some embodiments, each user-mode driver exposes DDIs from the DirectX® rendering framework from Microsoft® Corporation. For example, legacy user-mode driver 122 may support the version 9 series of DirectX® (e.g., version 9.0c). As such, legacy DDIs 124 may include DDIs from the version 9 series of Direct3D (e.g., DDIs relating to fixed GPU functions and some programmable GPU functions supported by Direct3D version 9, such as vertex shaders and/or pixel shaders). Continuing the DirectX® example, user-mode driver 118 may support one or more newer versions of DirectX®, such as the version 10 and/or the version 11 series of DirectX®. As such, DDIs 120 can include DDIs from one or more of versions 10 and/or 11 of Direct3D (e.g., DDIs relating to domain, hull, and/or geometric shaders). In addition, DDIs 120 may also include Direct2D DDIs. In some embodiments, DDIs 120 include all DirectX® version 10 and 11 DDIs, such that graphics runtime 114 can utilize the full range of DirectX® version 10 and 11 functionality.

In some embodiments, DDIs 120 include the following DDIs related to Direct3D versions 10, 10.1 and 11:

```
Direct3D 10
PFND3D 1ODDI_CALCPRIVATEDEPTHSTENCILSTATESIZE
  pfnCalcPrivateDepthStencilStateSize;
PFND3D 1ODDI_CALCPRIVATETEXTUREQUERYSIZE
  pfnCalcPrivateQuerySize;
PFND3D 1ODDI_CALCPRIVATERASTERIZERSTATESIZE
  pfnCalcPrivateRasterizerStateSize;
PFND3D 1ODDI_CALCPRIVATERENDERTARGETVIEWSIZE
  pfnCalcPrivateRenderTargetViewSize;
PFND3D 1ODDI_CALCPRIVATEAMPLERSIZE
  pfnCalcPrivateSamplerSize;
PFND3D 1ODDI_CALCPRIVATEHARDWARESHADERSIZE
  pfnCalcPrivateShaderSize;
PFND3D 1ODDI_CHECKCOUNTER
  pfnCheckCounter;
```
PFND3D 1ODDI_CHECKCOUNTERINFO
    pfncCheckCounterInfo;
PFND3D 1ODDI_CHECKFORMATSUPPORT
    pfncCheckFormatSupport;
PFND3D 1ODDI_CHECKMULTISAMPLEQUALITYLEVELS
    pfncCheckMultisampleQualityLevels;
PFND3D 1ODDI_CLEARDEPTHSTENCILVIEW
    pfncClearDepthStencilView;
PFND3D 1ODDI_CLEARRENDERTARGETVIEW
    pfncClearRenderTargetView;
PFND3D 1ODDI_CREATEDEPTHSTENCILSTATE
    pfncCreateDepthStencilState;
PFND3D 1ODDI_CREATEELEMENTLAYOUT
    pfncCreateElementLayout;
PFND3D 1ODDI_CREATEGEOMETRYSHADER
    pfncCreateGeometryShader;
PFND3D 1ODDI_CREATEPIXELSHADER
    pfncCreatePixelShader;
PFND3D 1ODDI_CREATEQUERY
    pfncCreateQuery;
PFND3D 1ODDI_CREATERasterizerState
    pfncCreateRasterizerState;
PFND3D 1ODDI_CREATERENDERTARGETVIEW
    pfncCreateRenderTargetView;
PFND3D 1ODDI_CREATESAAMPLER
    pfncCreateSampler;
PFND3D 1ODDI_CREATEVERTEXSHADER
    pfncCreateVertexShader;
PFND3D 1ODDI_DESTROYBLENDSTATE
    pfncDestroyBlendState;
PFND3D 1ODDI_DESTROYDEPTHSTENCILSTATE
    pfncDestroyDepthStencilState;
PFND3D 1ODDI_DESTROYDEPTHSTENCILVIEW
    pfncDestroyDepthStencilView;
PFND3D 1ODDI_DESTROYDEVICE
  pfnDestroyDevice;
PFND3D 1ODDI_DESTROYELEMENTLAYOUT
  pfnDestroyElementLayout;
PFND3D 1ODDI_DESTROYQUERY
  pfnDestroyQuery;
PFND3D 1ODDI_DESTROYRASTERIZERSTATE
  pfnDestroyRasterizerState;
PFND3D 1ODDI_DESTROYRENDERTARGETVIEW
  pfnDestroyRenderTargetView;
PFND3D 1ODDI_DESTROYRESOURCE
  pfnDestroyResource;
PFND3D 1ODDI_DESTROYSAMPLER
  pfnDestroySampler;
PFND3D 1ODDI_DESTROYSHADER
  pfnDestroyShader;
PFND3D 1ODDI_DESTROYSHADERRESOURCEVIEW
  pfnDestroyShaderResourceView;
PFND3D 1ODDI_DRAW
  pfnDraw;
PFND3D 1ODDI_DRAWAUTO
  pfnDrawAuto;
PFND3D 1ODDI_DRAWINDEXED
  pfnDrawIndexed;
PFND3D 1ODDI_DRAWINDEXEDINSTANCED
  pfnDrawIndexedInstanced;
PFND3D 1ODDI_DRAWINSTANCED
  pfnDrawInstanced;
PFND3D1 0DDI_FLUSH
  pfnFlush;
PFND3D 1ODDI_GENMIPS
  pfnGenMips;
PFND3D 1ODDI_IIA_SETINDEXBUFFER
  pfnlaSetIndexBuffer;
PFND3D 10DDI_IA_SETTOPOLOGY
   pfnlaSetTopology;
PFND3D 10DDI_IA_SETVERTEXBUFFERS
   pfnlaSetVertexBuffers;
PFND3D 10DDI_OPENRESOURCE
   pfnOpenResource;
PFND3D 10DDI_QUERYBEGIN
   pfnQueryBegin;
PFND3D 10DDI_QUERYEND
   pfnQueryEnd;
PFND3D1 10DDI_QUERYGETDATA
   pfnQueryGetData;
PFND3D 10DDI_RESETPRIMITIVELD
   pfnResetPrimitiveID;
PFND3D1 10DDI_RESOURCECOPY
   pfnResourceCopy;
PFND3D1 10DDI_RESOURCECOPY
   pfnResourceConvert;
PFND3D 10DDI_RESOURCECOPYREGION
   pfnResourceCopyRegion;
PFND3D 10DDI_RESOURCECOPYREGION
   pfnResourceConvertRegion;
PFND3D 10DDI_RESOURSEISSTAGINGBUSY
   pfnResourceIsStagingBusy;
PFND3D 10DDI_RESOURCENMAP
   pfnDynamicIABufferMapNoOverwrite;
PFND3D 10DDI_RESOURCENMAP
   pfnDynamicConstantBufferMapDiscard;
PFND3D 10DDI_RESOURCENMAP
   pfnDynamicIABufferMapDiscard;
PFND3D 10DDI_RESOURCENMAP
   pfnDynamicResourceMapDiscard;
PFND3D 10DDI_RESOURCENMAP
   pfnStagingResourceMap;
PFND3D 1ODDI RESOURCEMAP
  pfNameResourceMap;
PFND3D 1ODDI RESOURCE READ AFT EWRITEHAZARD
  pfNameResourceReadAfterWriteHazard;
PFND3D 1ODDI RESOURCERESOLVE SUBRESOURCE
  pfNameResourceResolveSubresource;
PFND3D1 ODDI RESOURCE UNMAP
  pfNameDynamicIABufferUnmap;
PFND3D1 ODDI RESOURCE UNMAP
  pfNameDynamicConstantBufferUnmap;
PFND3D1 ODDI RESOURCE UNMAP
  pfNameDynamicResourceUnmap;
PFND3D1 ODDI RESOURCE UNMAP
  pfNameStagingResourceUnmap;
PFND3D1 ODDI RESOURCE UNMAP
  pfNameResourceUnmap;
PFND3D 1ODDI RESOURCE UPDATE SUBRESOURCE US
  pfNameDefaultConstantBufferUpdateSubresourceUP;
PFND3D 1ODDI RESOURCE UPDATE SUBRESOURCE US
  pfNameResourceUpdateSubresourceUP;
PFND3D 1ODDI _SETBLEN DSTATE
  pfNameSetBlendState;
PFND3D1 0DDI _SETCONSTANTBUFFERS
  pfNameVsSetConstantBuffers;
PFND3D1 0DDI _SETCONSTANTBUFFERS
  pfNamePsSetConstantBuffers;
PFND3D1 0DDI _SETCONSTANTBUFFERS
  pfNameGsSetConstantBuffers;
PFND3D1 0DDI _SETCONSTANTBUFFERS
  pfNameDsSetConstantBuffers;
PFND3D 1ODDI _SETDEPTHSTENCILSTATE
  pfNameSetDepthStencilState;
PFND3D 1ODDI _SETINPUTLAYOUT
  pfnlaSetInputLayout;
PFND3D 1ODDI _SETPREDICATION
  pfnSetPredication;
PFND3D 1ODDI _SETSCISSORRECTS
  pfnSetScissorRects;
PFND3D 1ODDI _SETSHADERRESOURCES
  pfnPs SetShaderResources ;
PFND3D1 0DDI SETSHADERRESOURCES
    pfHsSetShaderResources;
PFND3D1 0DDI SETSHADERRESOURCES
    pfDsSetShaderResources;
PFND3D 1ODDI SETTEXTFILTERSIZE
    pfSetTextFilterSize;
PFND3D 1ODDI _SETVERTEXPIPELINEOUTPUT
    pfSetVertexPipelineOutput;
PFND3D 1ODDI _SETVIEWPORTS
    pfSetViewports;
PFND3D 1ODDI _SHADERRESOURCEVIEWREADAFTERWRITEHAZARD
    pfShaderResourceViewReadAfterWriteHazard;
PFND3D 1ODDI _SO_SETTARGETS
    pfSoSetTargets;

Direct 3D 10.1
PFND3D 10 _1DDI _CALCPRIVATEBLENDSTATESIZE
    pfCalcPrivateBlendStateSize;
PFND3D 10 _1DDI _CREATEBLENDSTATE
  pfCreateBlendState;

Direct 3D 11
PFND3D1 1DDI _ABANDONCOMMANDLIST
    pfAbandonCommandList;
PFND3D1 1DDI _CALCDEFERREDCONTEXTHANDLESIZE
    pfCalcDeferredContextHandleSize;
PFND3D1 1DDI _CALCPRIVATECOMMANDLISTSIZE
    pfCalcPrivateCommandListSize;
PFND3D1 1DDI _CALCPRIVATEDEFERREDCONTEXTSIZE
    pfCalcPrivateDeferredContextSize;
PFND3D1 1DDI _CALCPRIVATEDEPTHSTENCILVIEWSIZE
    pfCalcPrivateDepthStencilViewSize;
PFND3D 11DDI _CALCPRIVATEGEOMETRYSHADERWITHSTREAMOUTPUT
    pfCalcPrivateGeometryShaderWithStreamOutput;
PFND3D1 1DDI_CALCPRIVATERESOURCESIZE
    pfncalcPrivateResourceSize;
PFND3D1 1DDI_CALCPRIVATE_SHADERRESOURCEVIEWSIZE
    pfncalcPrivateShaderResourceViewSize;
PFND3D1 1DDI_CALCPRIVATE_TESSELLATION_SHADERSIZE
    pfncalcPrivateTessellationShaderSize;
PFND3D1 1DDI_CALCPRIVATE_UNORDERED_ACCESS_VIEWSIZE
    pfncalcPrivateUnorderedAccessViewSize;
PFND3D1 1DDI_CHECK_DEFERREDCONTEXT_HANDLESIZES
    pfncheckDeferredContextHandleSizes;
PFND3D1 1DDI_CLEAR_UNORDERED_ACCESS_VIEWFLOAT
    pfnClearUnorderedAccessViewFloat;
PFND3D1 1DDI_CLEAR_UNORDERED_ACCESS_VIEWSINT
    pfnClearUnorderedAccessViewUint;
PFND3D1 1DDI_COMMANDLIST_EXECUTE
    pfnCommandListExecute;
PFND3D1 1DDI_COPY_STRUCTURE_COUNT
    pfnCopyStructureCount;
PFND3D1 1DDI_CREATE_COMMANDLIST
    pfnCreateCommandList;
PFND3D1 1DDI_CREATE_DEFERRED_CONTEXT
    pfnCreateDeferredContext;
PFND3D1 1DDI_CREATE_DEPTH_STENCIL_VIEW
    pfnCreateDepthStencilView;
PFND3D1 1DDI_CREATE_DOMAIN_SHADER
    pfnCreateDomainShader;
PFND3D1 1DDI_CREATE_GEOMETRY_SHADER_WITH_STREAM_OUTPUT
    pfnCreateGeometryShaderWithStreamOutput;
PFND3D1 1DDI_CREATE_HULL_SHADER
    pfnCreateHullShader;
PFND3D1 1DDI_CREATE_RESOURCE
    pfnCreateResource;
PFND3D1 1DDI_CREATE_SHADER_RESOURCE_VIEW
    pfnCreateShaderResourceView;
PFND3D1 1DDI_CREATEUNORDERED_ACCESS_VIEW
    pfnCreateUnorderedAccessView;
PFND3D1 1DDI_DESTROYCOMMANDLIST
    pfnDestroyCommandList;
PFND3D1 1DDI_DESTROYCOMMANDLIST
    pfnRecycleDestroyCommandList;
PFND3D1 1DDI_CREATEUNORDERED_ACCESS_VIEW
    pfnDestroyUnorderedAccessView;
PFND3D1 1DDI_DISPATCH
    pfnDispatch;
PFND3D1 1DDI_DISPATCHINDIRECT
    pfnDispatchIndirect;
PFND3D1 1DDI_DRAWINDEXEDINSTANCEDINDIRECT
    pfnDrawIndexedInstancedIndirect;
PFND3D1 1DDI_DRAWINSTANCEINDIRECT
    pfnDrawInstanceIndirect;
PFND3D1 1DDI_RECYCLECOMMANDLIST
    pfnRecycleCommandList;
PFND3D1 1DDI_RECYCLECOMMANDLIST
    pfnRecycleCreateCommandList;
PFND3D1 1DDI_RECYCLECOMMANDLIST
    pfnRecycleCreateDeferredContext;
PFND3D1 1DDI_RELOCATEDEVICEFUNCS
    pfnRelocateDeviceFuncs;
PFND3D1 1DDI_SETRENDERTARGETS
    pfnSetRenderTargets;
PFND3D1 1DDI_SETRESOURCEMINLOD
    pfnSetResourceMinLOD;
PFND3D1 1DDI_SETSHADER_WITH_IFACES
    pfnPsSetShaderWithIFaces;
PFND3D1 1DDI_SETSHADER_WITH_IFACES
    pfnVsSetShaderWithIFaces;
PFND3D1 1DDI_SETSHADER_WITH_IFACES
    pfnGsSetShaderWithIFaces;
PFND3D1 DDI_SETSHADER_WITH_IFACES

PFN HsSetShaderWithlfaces;

PFND3D1 DDI_SETSHADER_WITH_IFACES

PFN DsSetShaderWithlfaces;

In other embodiments, legacy user-mode driver 122 exposes DDIs of a legacy version of another rendering framework, such as OpenGL® from Silicon Graphics, Inc, and user-mode driver 118 exposes DDIs of a newer version of the other rendering framework.

Generally, user-mode driver 118 and legacy user mode driver 112 are each configured to construct hardware contexts and command buffers. In particular, each user-mode driver converts graphic commands issued by guest software 112 (or graphics runtime 114) into hardware-specific commands for execution on physical GPU 104.

As depicted, for example, user-mode driver 118 may receive graphics commands 138, which request that GPU functions (e.g., domain, hull, or geometric shaders; Direct2D functions) be executed at physical GPU 104 on behalf of guest software 112. User-mode driver 118 is configured to convert the received graphics commands into hardware-specific commands (i.e., commands that are specific to physical GPU 104). As part of the conversion, user-mode driver 118 maintains proper hardware context for physical GPU 104. For example, user-mode driver 118 translates logical values for settings affecting a rendering pipeline into values and corresponding physical settings. User-mode driver 118 is also configured to store converted hardware-specific commands in a command buffer. Legacy user-mode driver 122 is similarly configured.

vGPU 116 also includes kernel-mode driver 126 executing in a kernel-mode of child partition 110. Kernel-mode driver 126 is configured to receive command buffers from user mode driver 118 and legacy user mode driver 112 and to construct corresponding direct memory access ("DMA") buffers. When it is time for a DMA buffer to be processed, kernel-mode driver 126 is configured to submit the DMA buffer to physical GPU 104.

As depicted, kernel mode driver 126 includes processing component 128 which is configured to process command buffers from user-mode driver 118 and legacy processing component 130 which is configured to process command buffers from legacy user-mode driver 122. For example, Figure 1 depicts kernel mode driver 126 receiving command buffer 140 from user-mode driver 118. Kernel-mode driver is configured to process command buffers received from user-mode driver 118 using processing.
component 128 to generate a DMA buffer. While not depicted, kernel mode driver 126 can also communicate with graphics runtime 114 directly. As such, one or both of processing component 128 and legacy processing component 130 can process commands received directly from graphics runtime 114. For example, kernel mode driver 126 can receive and process commands related to geometry, hull, and/or domain shaders and Direct2D functions.

[0040] As indicated previously, root partition 108 includes render component 132, which is configured execute any requested GPU functions on physical GPU 104. As depicted, render component 132 includes renderer 134 which is configured to initiate execution of GPU functions corresponding to DDIs 120, and legacy renderer 136 which is configured to initiate execution of GPU functions corresponding to legacy DDIs 120. As such, render component 132 is configured to execute GPU functions from both a newer rendering framework and a legacy rendering framework.

[0041] In particular, render component 132 is configured to schedule any physical GPU-specific rendering commands graphics commands received from vGPU 116 for execution on physical GPU 104, and to create proper context for executing those commands. For example, render component 132 is configured to create any appropriate composition device(s) at physical GPU 104. The composition device typically represents a desktop running in connection with guest software 112.

[0042] As depicted, computer architecture 100 includes communications connections between render component 132 and user-mode driver 118, legacy user-mode driver 122 and kernel-mode driver 126. In addition, computer architecture 100 enables negotiation of communications protocol(s) among these components.

[0043] Negotiation can include determining an appropriate composition device to create at physical GPU 104 so that GPU functions requested by graphics runtime 114/guest software 112 can be accelerated at physical GPU 104. During boot time of a guest operating system within child partition 110, for example, the composition device may be determined to correspond to user-mode driver 118 of a newer rendering framework, or correspond to legacy user-mode driver 122 and a legacy rendering framework. After this initial communications protocol negotiation process, protocol negotiation may be repeated for each guest application run as part of guest software 112 to create an appropriate composition device representing each application. Composition devices can concurrently be created for applications using both newer rendering framework(s) and a legacy rendering framework. As such, computer architecture 100 supports execution of GPU
functions from multiple rendering frameworks at physical GPU 104, including newer rendering frameworks represented by user-mode driver 118 and a legacy rendering framework represented legacy user-mode driver 122.

[0044] Once appropriate composition device(s) is/are determined, each of the supported DDIs are represented by one or more tokens. These tokens are communicated between each user-mode driver and corresponding components at kernel-mode driver 126 and render component 132. Thus, any GPU functions corresponding to any supported DDIs are enabled by computer architecture 100.

[0045] Figure 2 illustrates a flow chart of an example method 200 for providing a programmable GPU pipeline to a guest application executing in a child partition of a para-virtualized execution environment. Method 200 will be described with respect to the components and data of computer architecture 100.

[0046] Method 200 includes an act of instantiating a virtual machine session, including instantiating a hypervisor that provides (i) a root partition having access to the physical GPU, and (ii) the child partition which executes the guest application (act 201). For example, hypervisor 106 is instantiated. Hypervisor 106 is used to provide root partition 108 and child partition 110. Child partition can execute guest software 112.

[0047] Method 200 also includes an act of presenting a vGPU to the guest application, the vGPU executing within the child partition, including presenting a plurality of DDIs of a rendering framework to the guest application as part of a UMD of the vGPU, the plurality of DDIs providing an API that enables the guest application to send commands to the vGPU for programming a GPU pipeline of the physical GPU to utilize one or more features of the rendering framework, including utilizing at least one of: a domain shader, a hull shader, or a geometric shader (act 202). For example, child partition 110 presents vGPU 116 to guest software 112/rendering framework 114. vGPU 116 virtualizes physical GPU 104. vGPU 116 includes user-mode driver 118. User-mode driver 118 exposes DDIs 120, which correspond to a rendering framework that includes one or more of domain shader, a hull shader, or a geometric shader. In some embodiments, DDIs 120 expose all DDIs of one or more rendering frameworks.

[0048] Method 200 also includes an act of a render component executing within the root partition receiving at least one physical GPU-specific command from the vGPU for using one or more of: a domain shader, a hull shader, or a geometric shader (act 204). For example, render component 132, which executes in root partition 108, can receive GPU commands and data from user-mode driver 118 and/or kernel-mode driver 126. These
commands and data can include physical-GPU specific commands that were generated by user-mode driver 118 in response to receiving graphics commands 138 from graphics runtime 114. These physical-GPU specific commands can include commands related to a newer graphics runtime, such as commands related to a domain shader, a hull shader, or a geometric shader.

[0049] Method 200 also includes an act of the render component scheduling the at least one physical GPU-specific command for execution at the physical GPU (act 205). For example, renderer 134 can schedule the received physical-GPU specific commands for execution on physical GPU 104. This can involve setting up appropriate context and composition device(s) at physical GPU 104.

[0050] Accordingly, the present invention provides domain, hull, and geometry shaders in a para-virtualized environment. Thus, a guest application executing in a child partition is enabled use a rich programmable GPU pipeline of a physical GPU.

[0051] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes that come within the meaning and range of equivalency of the claims are to be embraced within their scope.
CLAIMS

1. A computer system including one or more processors and system memory, the computer system also including a physical graphics processing unit ("GPU"), a method for providing a programmable GPU pipeline to a guest application executing in a child partition of a para-virtualized execution environment, the method comprising:

   an act of instantiating a virtual machine session, including instantiating a hypervisor that provides (i) a root partition having access to the physical GPU, and (ii) the child partition which executes the guest application;

   an act of presenting a virtualized graphics processing unit ("vGPU") to the guest application, the vGPU executing within the child partition, including presenting a plurality of device driver interfaces ("DDIs") of a rendering framework to the guest application as part of a user-mode driver ("UMD") of the vGPU, the plurality of DDIs providing an application programming interface that enables the guest application to send commands to the vGPU for programming a GPU pipeline of the physical GPU to utilize one or more features of the rendering framework, including utilizing at least one of: a domain shader, a hull shader, or a geometric shader; and

   an act of a render component executing within the root partition receiving at least one physical GPU-specific command from the vGPU for using one or more of a domain shader, a hull shader, or a geometric shader at the physical GPU; and

   an act of the render component scheduling the at least one physical GPU-specific command for execution at the physical GPU.

2. The method as recited in claim 1, wherein the act of a render component executing within the root partition receiving at least one physical GPU-specific command from the vGPU comprises and act of the render component receiving at least one physical GPU-specific command for using a domain shader.

3. The method as recited in claim 1, wherein the act of a render component executing within the root partition receiving at least one physical GPU-specific command from the vGPU comprises and act of the render component receiving at least one physical GPU-specific command for using a hull shader.

4. The method as recited in claim 1, wherein the act of a render component executing within the root partition receiving at least one physical GPU-specific command from the vGPU comprises and act of the render component receiving at least one physical GPU-specific command for using a geometric shader.
5. The method as recited in claim 1, wherein the UMD converts any graphics commands received from the guest application into corresponding physical GPU-specific commands and stores the corresponding physical GPU-specific commands in a command buffer.

6. The method as recited in claim 5, wherein the act of presenting a vGPU to the guest application comprises an act of presenting a vGPU that includes a kernel-mode driver ("KMD"), the KMD being configured to construct a direct memory access buffer from the command buffer.

7. The method as recited in claim 6, wherein the UMD sends a command buffer containing physical GPU-specific compute shader commands to the KMD.

8. The method as recited in claim 6, wherein instantiating a hypervisor that provides (i) a root partition having access to the physical GPU, and (ii) the child partition which executes the guest application comprises an act of negotiating one or more communications protocols among the UMD, the KMD, and the render component, including determining a type of composition device to instantiate based on a version of the rendering framework supported by the UMD.

9. The method as recited in claim 1, wherein the plurality of DDIs enable the guest application to send graphics commands to the vGPU for programming a GPU pipeline of the physical GPU to utilize all features of the rendering framework.

10. The method as recited in claim 9, wherein the UMD comprises a new UMD that is enabled to execute concurrent with a legacy UMD, and wherein the render component is enabled to communicate with both the new UMD and the legacy UMD, such that both a new three-dimensional rendering framework and a legacy three-dimensional rendering can be used concurrently.

11. A computer program product for use at a computer system, the computer program product for implementing a method for providing GPU-accelerated computing functionality to a guest application executing in a child partition of a para-virtualized execution environment, the computer program product comprising one or more computer storage media having stored thereon computer-executable instructions that, when executed at a processor, cause the computer system to perform the method, including the following:

   instantiate a virtual machine session, including instantiating a hypervisor that provides (i) a root partition having access to a physical GPU, and (ii) the child partition which executes the guest application;
present a virtualized graphics processing unit ("vGPU") to the guest application, the vGPU executing within the child partition, including presenting a plurality of device driver interfaces ("DDIs") of a rendering framework to the guest application as part of a user-mode driver ("UMD") of the vGPU, the plurality of DDIs providing an application programming interface that enables the guest application to send commands to the vGPU for programming a GPU pipeline of the physical GPU to utilize one or more features of the rendering framework, including utilizing at least one of: a domain shader, a hull shader, or a geometric shader; and receive, at a render component executing within the root partition, at least one physical GPU-specific command for using one or more of a domain shader, a hull shader, or a geometric shader at the physical GPU; and schedule the at least one physical GPU-specific command for execution at the physical GPU.

12. The computer program product as recited in claim 11, wherein the UMD converts any graphics commands received from the guest application into corresponding physical GPU-specific commands and stores the corresponding physical GPU-specific commands in a command buffer.

13. The computer program product as recited in claim 12, wherein the act of presenting a vGPU to the guest application comprises an act of presenting a vGPU that includes a kernel-mode driver ("KMD"), the KMD being configured to construct a direct memory access buffer from the command buffer.

14. The computer program product as recited in claim 11, wherein the plurality of DDIs enable the guest application to send commands to the vGPU for programming a GPU pipeline of the physical GPU to utilize functions of both Direct2D and Direct3D.

15. A computer system, the computer system comprising:

one or more processors;

a graphics processing unit ("GPU");
system memory; and

one or more computer-readable storage devices having stored thereon computer-executable instructions representing a virtualized graphics processing unit ("vGPU") and a render component,

wherein the vGPU is configured to execute within a child partition and includes a user-mode driver ("UMD") configured to:
present a plurality of device driver interfaces ("DDIs") of a DirectX rendering framework to a guest application executing in a child partition, the plurality of DDIs providing an application programming interface that enables the guest application to send commands to the vGPU for programming a GPU pipeline of the GPU to utilize one or more of: a domain shader, a hull shader, or a geometric shader of the DirectX rendering framework; and convert vGPU-specific commands to GPU-specific commands; and wherein the a render component is configured to execute in a root partition and to receive at least one GPU-specific command from the vGPU and to schedule the at least one physical GPU-specific command for execution at the GPU while using one or more of: a domain shader, a hull shader, or a geometric shader of the DirectX rendering framework.
Figure 2


203 A Render Component Executing Within The Root Partition Receiving At Least One Physical GPU-Specific Command From The vGPU For Using One Or More Of: A Domain Shader, A Hull Shader, Or A Geometric Shader

204 The Render Component Scheduling The At Least One Physical GPU-Specific Command For Execution At The Physical GPU
INTERNATIONAL SEARCH REPORT

International application No. 
PCT/US2013/020912

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/38(2006.01)i, G06F 9/44(2006.01)i, G06F 9/30(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/38; G06F 17/00; G06T 17/00; G06F 7/00; G06T 15/50; G06T 1/00; G09G 5/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic database consulted during the international search (name of database and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: GPU, shader, domain, hull, geometric, virtualize, rendering

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>US 2006-0161572 Ai (VOGT SEBASTIAN et al.) 20 July 2006 See paragraphs [0010]-[0051] and claims 1-5</td>
<td>1-15</td>
</tr>
<tr>
<td>A</td>
<td>US 2005-0231503 AI (PHENG-ANN HENG et al.) 20 October 2005 See paragraphs [0047]-[0062] and claims 1,8-10</td>
<td>1-15</td>
</tr>
</tbody>
</table>

A. Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" member of the same patent family

Date of the actual completion of the international search 26 April 2013 (26.04.2013)

Date of mailing of the international search report 29 April 2013 (29.04.2013)

Name and mailing address of the ISA/KR
Korean Intellectual Property Office 189 Cheongna-ro, Seo-gu, Daejeon Metropolitan City 302-701, Republic of Korea
Facsimile No. 82-42-472-7140

Authorized officer
KIM, Yong Woong

Telephone No. 82-42-481-5585

Form PCT/ISA/210 (second sheet) (My 2009)
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WO 2008-039950 A1</td>
<td>03.04.2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1691069 A</td>
<td>02.11.2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 1691069 C0</td>
<td>02.11.2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HK 1080974 A1</td>
<td>09.07.2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 7184041 B2</td>
<td>27.02.2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2005-104042 A1</td>
<td>03.11.2005</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2011-0063313 A1</td>
<td>17.03.2011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2011-0063318 A1</td>
<td>17.03.2011</td>
</tr>
</tbody>
</table>