Virtual Machines are increasingly being deployed on Desktop PC’s, but because many existing VMM technologies are mostly targeted at servers, graphical performance is a stumbling block to many modern applications such as games, simulation, and video-conferencing. Existing high-performance display systems sacrifice safety and provide applications with direct hardware access, but the aim of our work is to build a display system with Virtual Machine quality safety and isolation, while retaining performance comparable to less safe “direct” approaches. We are developing Blink, a Display System for Virtual Machines, the core component of which is a JIT-compiler for extended OpenGL programs which execute safely inside the address space of the display server.

The display system is hard to multiplex in a way that is both efficient and safe, especially for demanding applications such as 3D games or realtime video. This is evidenced by the fact that the major operating systems all provide “direct” avenues of contacting the GPU, largely without operating system involvement. However, some level of cooperation between applications is necessary, so that multiple applications are able to share a single screen without exceeding their screen space, and without causing the GPU to crash by unsafely intermingling hardware command streams. Current systems trust clients to behave correctly, and thus trade safety for performance, but Virtual Machine systems cannot afford to make such trade-offs. The ability to run untrusted code sandboxed within a VM is the main motivation for desktop deployment, so the choice is between either limiting the VM display model to a dumb but safe 2D framebuffer model, or a model which provides safe access to accelerated graphics hardware.

Our new display system, called Blink, multiplexes OpenGL content coming from multiple untrusted Virtual Machines onto a single Graphics Processing Unit (GPU). Blink does not allow clients to program the graphics card GPU directly, but instead provides a Virtual Processor (VP) abstraction to which they can program. VP programs execute within the context of Blink, and in turn program the GPU on behalf of the client. Blink employs JIT-compilation and simple static analysis of VP programs, with increased flexibility and a reduction in processing and context switching overheads as results. Visible jitter is reduced because client code executes independently of the client VM timeslice. Blink is targeted at paravirtualized VM’s—it supplies virtualized applications with an OpenGL-like abstraction that they can program to, but does not aim to be a transparent adaptation layer for existing windowing systems such as X11. Instead, Blink supplies the mechanisms on which such a layer can be constructed if needed.

Our results show that this model can lead to fewer context switches than traditional client/server approaches, can eliminate redundant copying and clearing of graphics buffers, and reduce the visible effects of scheduler timing variance. In many cases, the cost of compilation is completely amortized over a large number of invocations, and even when this is not the case, the overhead added by JIT compilation is a modest 20% compared to native execution.