一种采用检查点机制的动态主动GPU抢占机制

介绍

GPU由于其大规模并行处理能力，已经在高性能计算，机器学习和科学计算等领域得到广泛应用。这些方面的计算如今以服务的形式存在于数据中心或云上，而GPU则可作为共享基础硬件资源提供给不同的用户。多任务在GPU中为支持并行服务和任务已经变得必不可少。已经有些主要的硬件特性支持多任务处理，例如Nvidia Kepler体系结构提供的HYPER-Q, AMD支持的命令处理机制。虽然已经有这些机制支持多任务处理，但还需要更多的工作来支持真正的多任务处理机制。

上下文切换是一种在CPU中常用的支持并行性的技术，如今已经被应用到了GPU来支持多任务。CPU的进程相对来说非常轻量化，所以在上下文切换和任务时分应用上非常快速高效。但是，一个CUDA的上下文相比于CPU却是非常巨大的。比如说在NVIDIA GTX980 GPU上，上下文包括每个流多核处理器的256KB的寄存器和96KB的共享内存，对于一个含16个流多核处理器的GPU，其上下文大小达到了5664KB，存储这样大的上下文需要花费大量的存储带宽，并带来严重的性能损失。

之前已经有许多尝试来降低GPU的上下文切换的开销。最早出现的技术是让上下文切换仅在一部分SM上进行，这样其他的SM就能保持执行。被切换的SM回完全停止执行指令，以完成上下文的存取，这些操作对于存储带宽的要求依然非常高。之后出现的方法是，让一部分线程块继续执行直到完成，而只上下文切换一部分线程块。这样可以最大化的利用程序访存和上下文访存的重叠并女性。这个技术被进一步加强扩展，允许在需要抢占时每一个SM内部的不同线程块同时进行执行（直至线程块完成）、丢弃（满足幂等性）、和上下文切换。这些选择均取决于每个线程块对于抢占的终止时间的要求。除了这些工作，一种轻量级的上下文切换技术被设计来降低需要保存在片外存储器的上下文大小。所有这些方法都是通过被动的方法来实现抢占，即只有当抢占请求到来以后才激活所有的操作。因此，如果没有丢弃操作，抢占延迟依然是对性能的一大挑战。

在本章，我们提出了一种动态主动的抢占机制，PEP。这种抢占机制能够大大降低抢占延迟和开销。通过观察kernel从CPU到GPU的发射过程，我们发现kernel的实际执行总是在kernel发射之后。从一个kernel在CPU发射到开始在GPU执行，大约需要几十个毫秒的数量级，我们可以通过预计抢占请求的到达时间来主动准备上下文切换。当抢占的kernel真正到达的时候，需要等待完成的上下文切换工作将变得非常小。因此，需要等待的抢占时间将变得非常短。准备上下文切换的工作我们采用了检查点（checkpoint）的概念。第一个检查点，我们在抢占被预计发生时备份当前的上下文。当真正的抢占请求到达GPU后，仅需再备份变化的上下文。备份变化的上下文相比于完整的kernel的上下文能够节省大量的时间，减少抢占kernel的等待时间。平均来看，总的需要备份的上下文不大于所有的上下文。我们还观察到分配的上下文在线程块的生命周期里并不是完全激活的。所以，我们为寄存器设置脏位，来表明该寄存器是否是激活的。只有有效的寄存器才会被备份，这大大减少了需要存储的上下文大小。此外，我们设计了一个动态实时调度策略来确定抢占方法。小的kernel将要继续执行直到结束，而大的kernel需要采用checkpoint（上下文切换）来进行抢占。这个算法可以达到最小延迟和开销。

我们的贡献主要包括：

1. 我们研究了kernel发射的过程，观察到抢占的事件可以被预测。
2. 我们引入了一种主动的抢占技术来减少抢占的kernel等待上下文切换的时间。采用主动checkpoint技术，当真正的抢占请求到来时，只有一小部分的脏上下文需要被存储。
3. 我们使用了一种相对简单的脏数据存储技术来减少上下文大小，这可以减少不必要的上下文存储。
4. 我们开发了一种相对更加精确的线程块执行时间和上下文切换时间估算方法，设计了实时动态选择算法来确定采用的抢占方法。我们可以完成小kernel和大kernel的抢占，并使之达到最短延迟和最小开销。

我们实验评测了PEP，并与之前最好的抢占工作Chimera在集中不同类型的测试集里比较。我们的实验结果显示，相比之前的工作Chimera，我们可以将平均抢占延迟从8.9us降低到3.6us。我们采用的简单的上下文大小减少技术，将需要存储的上下文从完整的上下文大小减少了16.1%。PEP的总开销，即平均线程块切换延迟相比Chimera减少了6.3%。