摘要

## 课题背景

Docker®容器通常用于在多台机器上无缝部署基于CPU的应用程序。有了这个用例，容器就是硬件无关的和平台无关的。使用NVIDIA GPU显然不是这样，因为它使用专门的硬件，并且需要安装NVIDIA驱动程序。因此，Docker Engine本身并不支持具有容器的NVIDIA GPU。

为了解决这个问题，出现的一个早期解决方案是在容器内完全重新安装NVIDIA驱动程序，然后在启动容器时传递对应于NVIDIA GPU（例如/dev/nvidia0）的字符设备。但是，该解决方案是脆弱的：主机驱动程序的版本必须与容器中安装的驱动程序版本完全匹配。 Docker图像无法共享，必须在每台机器上本地构建，从而打破了Docker的主要优势之一。

为了使Docker图像在使用NVIDIA GPU的同时便携式，nvidia-docker使用的解决方案是使图像与NVIDIA驱动程序无关。启动目标机器上的容器时，将安装所需的字符设备和驱动程序文件。

容器将应用程序包装到隔离的虚拟环境中，以简化数据中心部署。 通过包括所有应用程序依赖关系，如二进制文件和库，应用程序容器可在任何数据中心环境中无缝运行。

Docker是领先的集装箱平台，现在可用于集成GPU加速应用程序。 这意味着您可以轻松地集成和隔离加速应用程序，无需任何修改，并将其部署在任何支持GPU的基础设施上。 管理和监控加速数据中心从未如此简单。

## 文献综述

## 研究目标

本课题将研究基于金融IC卡标准的市民卡多应用平台的系统设计与实现。系统将实现IC卡应用生命周期管理、卡片生命周期管理、卡片安全域管理、灵活的行业应用接入、动态应用加载、并可对接其他TSM平台，如人行TSM、银联TSM、移动TSM等，实现应用共享和移动支付。

本课题将研究高性能计算系统在虚拟化技术中的实现。

# 研究内容

# 主要研究方案、技术路线与可行性分析

## 研究方案

## 技术路线

## 可行性分析

# 预期研究成果和（或）创新点

# 工作基础和条件

# 研究进度安排

# 参考文献

1. 绪论
   1. 课题的背景介绍
   2. 国内现状与国外研究进展
   3. 论文主要目标
   4. 本文组织结构
2. 系统相关技术研究
   1. 虚拟化技术概述
   2. 基于Docker的容器技术
   3. 高性能计算平台
   4. 系统开发环境
   5. 本章小结
3. 高性能计算模型研究与设计
   1. CUDA编程模型
   2. 合作组织(CG)
   3. MPS
   4. 动态平行度
   5. NVML
   6. GPU P2P
   7. GPUDirect
   8. 本章小结
4. 系统在容器内的设计与实现
   1. 硬件平台结构
   2. Docker架构平台
      1. Docker概况
      2. Docker卷
      3. Docker扩展应用
   3. 软件系统组织结构
5. 算法在高性能计算系统中的实现与测试
   1. Cholesky分解算法在系统中实现
   2. LU分解算法在系统中实现实现
   3. PageRank算法在系统中实现
   4. Linpack测试容器内系统性能
6. 结论与展望
   1. 总结
   2. 展望

摘要

问题是什么。我们做了什么。大概怎么做的。做的还不错。

在高性能计算系统中构建和部署基于图形处理加速器的软件环境是一项极具挑战性的任务。高性能应用程序必须可靠地运行在多个平台和环境中，并在解决复杂的软件堆栈依赖关系的同时充分利用图形处理加速器的资源。容器通常用于在多台机器上无缝部署基于CPU的应用程序。有了这个用例，容器就是硬件无关的和平台无关的。使用英伟达图形处理加速器显然不是这样，因为它使用专用的硬件，并且需要安装英伟达驱动程序。因此，Docker Engine本身并不支持将英伟达图形加速器封装在容器内部。

容器是一种轻量级的虚拟化技术，具备便携式，易于构建和部署，占用空间小，运行时间少等优势。通过将应用程序及其环境封装到标准的软件单元中来解决部署复杂环境的问题。为了使Docker图像在使用英伟达图形加速器的同时具备便携式和跨平台的特点，容器使用的解决方案是使图像与英伟达驱动程序无关。启动目标机器上的容器时，将安装所需的字符设备和驱动程序文件。

本文研究分析了基于Volta架构的图形处理器和统一计算设备架构，如合作组织，动态平衡度等编程模型。笔者通过在实际工作中遇到的问题和需求为Docker提供了运行时的扩展应用，它无缝隙的支持基于Volta架构的图形处理器，并在容器内能实现所有最新CUDA提供的技术方案，它为容器化应用程序提供了访问图形加速器和主机系统的专用机制，从而有效地解决了高性能计算资源在容器内的移植性问题。

在实际工作中，优化后的容器将应用程序包装到隔离的虚拟环境中，通过此项技术，大大简化了CUDA和操作系统在数据中心环境部署工作，运维和测试人员可以轻松地集成和隔离加速应用程序，无需任何修改，并将其部署在任何支持GPU的基础设施上,极大地节省了资源。

关键字：GPU，Docker，高性能计算，容器，虚拟化

Abstract

绪论

第一章

1.1 课题背景介绍

长期以来计算机科研人员最为担忧的是软件的可移植性问题。起初在20世纪50年代后期，人们创造出COBOL编程语言，这主要用来降低软件移植到新平台上去的复杂性和成本[1]。自那时起已经有60多年了，目前人们仍然在努力应对这些问题的根本性挑战。在高性能计算（HPC）领域，人们越来越多的注意力聚焦在便携性上，这是因为在不损失高水平计算性能的前提下，高性能应用程序通常需要被运行在各种各样的平台和环境中。

从最初人们在COBOL中的工作，许多创新使得我们更加关注多年来软件的可移植性。让我们考虑下面的几个方面：编程语言（例如COBOL，Fortran，C，C++，Python），便携式库（例如Boost C++，PETSc），通用操作系统（例如嵌入式系统到超级计算机的Linux），计算机平台标准（例如IBM-PC,基于x86的架构），软件模式（例如通过可重用的软件组件的可移植性）。 尽管这些努力已经成为人们处理软件的重要工具，但是由于对于如：性能，资源限制，依赖机器的功能，与操作系统有关的要求，软件库和各种工具的可用性日益增长的需求使得软件可移植性成为一个极为复杂的问题。因此，开发人员通常会将可移植性作为与其他要求的权衡[3]。这里我们考虑使用Hadoop的方法，Hadoop是一个开源框架用作大数据集的分布式处理和存储，通过抽象出平台软件和硬件的细节来实现可移植性。如[3]中所讨论的，Hadoop中对软件层的抽象就是一个十分有针对性的例子，它在性能和可移植性之间做了权衡。

在高兴能计算需求的背景下，人们要求程序同时能兼顾高性能和可移植性。特别是在超级计算机架构中这项要求特别具有挑战，以为配置和软件环境在系统和供应商之间差异很大。

我们要求高性能软件能适应各种不同的系统环境。工程师们最需要考虑的是在跨超级计算机站点移植代码任务时候产生的时间和金钱成本。然而，大多数情况下，用户和开发人员都希望将其时间花在新的科学研究和软件开发商，而不是在平台和环境之间移植应用程序。因此，简化和加速应用程序的移植的工作流程可以大大提高用户和开发人员的生产力。

虚拟化技术在过去十多年中呈现快速增长的趋势，它展现出可以胜任轻松移植和部署 应用程序，尤其在云环境中被使用的最为广泛。下面我们先看一个在虚拟化环境中和实体机上系统软件架构的区别。

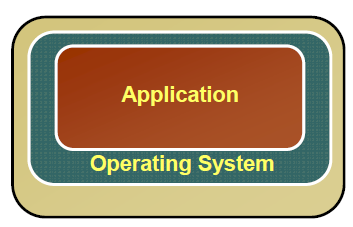


图1.1 实体系统

作为直接在部署在硬件上的系统，它具备以下几个特点：

1. 每台机器都需要至少一个操作系统
2. 软件和硬件紧密耦合
3. 在同一机器上运行多个应用程序通常会产生冲突
4. 资源利用不充分

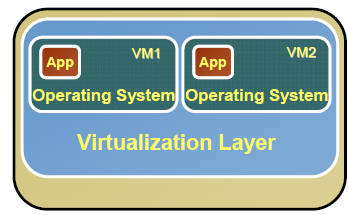


图1.2 典型虚拟化系统

当我们把环境部署在虚拟化系统中后，它具备下列特点：

1. 虚拟机打破了操作系统和硬件之间1比1的依赖关系
2. 通过将操作系统和应用程序封装成虚拟机将其视作一个单一的单元来管理
3. 虚拟环境之间良好的隔离性
4. 将软件和系统封装进虚拟环境后，硬件被独立出来。我们可以在任何硬件环境下运行程序。

从上面图例我们可以看到所谓虚拟化，就是通过将操作系统的部分或者全部和软件环境与应用程序运行所依赖的底层硬件解耦，这项技术提高了软件的可移植性和可维护性。计算机虚拟化技术运行虚拟化程序无需修改也可以在其他系统上完美运行，从而实现了更快的软件移植工作流程。近些年来，由于关于虚拟化应用程序的性能问题，人们在高性能系统中采用虚拟化技术受到了限制和制约[4]。令人欣慰的是这些问题中的一部分已经通过容器虚拟化技术来解决[5][6]。容器虚拟化被视作为更灵活，轻便，更易于部署的基于虚拟化管理程序的虚拟化替代方案。相比于虚拟机技术，容器虚拟化将容器实例运行在本机系统内核上。

这篇论文讨论了一种基于容器虚拟化的方案，提出一种基于Docker的应用扩展来使用专用的图形加速处理作为计算系统的底层基础硬件并且不损失性能，为满足高性能计算系统需求而设计的一种解决方案。

1.2 国内现在与国外研究进展

1.2.1 虚拟化技术现状

上世纪60年代开始，美国的计算机学术界就有了虚拟技术思想的萌芽。1959年克里斯托弗（ChristopherStrachey）发表了一篇学术报告，名为《大型高速计算机中的时间共享》（TimeSharinginLargeFastComputers），他在文中提出了虚拟化的基本概念，这篇文章也被认为是虚拟化技术的最早论述。L.W. Comeau 和 R.J. Creasy 创造性地设计了一种名为 CP-40 的新型操作系统，该操作系统实现了虚拟内存和虚拟机。

虚拟化技术在 20 世纪 60 年代首次出现，由IBM 率先实施：对大型机进行逻辑分区以形成若干独立虚拟机的一种方式。这些分区允许大型机进行“多任务处理”：同时运行多个应用程序和进程。原因是当时大型机是十分昂贵的资源，因此设计虚拟化技术来进行分区，作为一种充分利用投资的方式，解决了大型机的僵化和使用率不足的情况。

在 20 世纪 80 年代和 90 年代，由于客户端-服务器应用程序以及价格低廉的x86服务器和台式机组成了分散的计算机架构，大型机上的虚拟化技术处于停滞不前的状态。

在20世纪，虚拟化技术基本上都是服务器虚拟化，进入了21世纪，随着IT的发展，虚拟化的思路被借用到服务器以外的领域(包括存储，网络，桌面应用等)，形成了各种各样的虚拟化技术。本文重点从桌面应用来看，开始出现了应用虚拟化（也称桌面虚拟化）的技术，该技术把应用程序的人机交互逻辑（应用程序界面、键盘及鼠标的操作、音频输入输出、读卡器、打印输出等）与计算逻辑隔离开来，客户端无需安装软件，通过网络连接到应用服务器上，计算逻辑从本地迁移到后台的服务器完成，实现应用的快速交付和统一管理。

1.2.2 高性能计算平台现状

自1946年第一台电子计算机ENIAC问世至今，超级计算机的发展已先后经历了5个阶段或5代，即早期的单处理器巨型机、向量处理系统、大规模并行处理系统、共享内存处理系统和机群系统。

1983年，中国第一台被命名为“银河”的亿次巨型电子计算机在国防科技大学诞生。它的研制成功向全世界宣布：中国成了继美、日等国之后，能够独立设计和制造巨型机的国家。

1992年，国防科技大学研制出银河－II通用并行巨型机，峰值速度达每秒10亿次，主要用于中期天气预报。

1993年，国家智能计算机研究开发中心（后成立北京市曙光计算机公司）研制成功曙光一号全对称共享存储多处理机，这是国内首次以基于超大规模集成电路的通用微处理器芯片和标准UNIX操作系统设计开发的并行计算机。 1995年，曙光公司又推出了曙光1000，峰值速度每秒25亿次浮点运算，实际运算速度上了每秒10亿次浮点运算这一高性能台阶。曙光1000与美国Intel公司1990年推出的大规模并行机体系结构与实现技术相近，与国外的差距缩小到5年左右。

1997年，国防科技大学研制成功银河－III百亿次并行超级计算机系统，峰值性能为每秒130亿次浮点运算。

1997至1999年，曙光公司先后在市场上推出曙光1000A，曙光2000－I，曙光2000－II超级服务器，峰值计算速度突破每秒1000亿次浮点运算。 1999年，国家并行计算机工程技术研究中心研制的神威I计算机，峰值运算速度达每秒3840亿次，在国家气象中心投入使用。

2004年，由中科院计算所、曙光公司、上海超级计算中心三方共同研发制造的曙光4000A实现了每秒10万亿次运算速度。

2008年，“深腾 7000”是国内第一个实际性能突破每秒百万亿次的异构机群系统，Linpack溪能突破每秒106.5亿次。

2008年，曙光5000A实现峰值速度230万亿次、Linpack值180万亿次。作为面向国民经济建设和社会发展的重大需求的网格超级服务器，曙光5000A可以完成各种大规模科学工程计算、商务计算。

2009年10月29日，中国首台千万亿次超级计算机“天河一号”诞生。这台计算机每秒1206万亿次的峰值速度和每秒563．1万亿次的Linpack实测性能，使中国成为继美国之后世界上第二个能够研制千万亿次超级计算机的国家。 2011年11月，中国第一台实测性能超千万亿次的超级计算机曙光“星云”在国家深圳超算中心全面开通运行。“星云”系统峰值为每秒3000万亿次，实测Linpack性能达到每秒1271万亿次，是中国第一台、世界第三台实测双精度浮点计算超过千万亿次的超级计算机。

由日本政府出资、富士通制造的巨型计算机“K Computer”目前落户于日本理化研究所，并成功从中国手中夺回运算速度排行榜第一的宝座。“K Computer”目前的运算速度为每秒8000万亿次，而到2012年其完全建成时，运算速度将达到每秒一万万亿次。“K Computer”比现居第二的中国超级计算机速度快出约3倍，甚至比排名第2至第6的计算机运算速度总和还要快。

进入新世纪，随着研制高端计算机系统的诸多关键技术被攻克(尤其是机群技术)，我国自行研制的高端计算机系统已开始形成自己的品牌系列和一定的市场规模，其发展呈现星火燎原之势头。近两年，随着“神威”、“银河”、“曙光”、“深腾”、“天梭”等一批知名产品的出现，使我国成为继美、日之后第三个具备高端计算机系统研制能力的国家，被誉为世界未来高端计算市场的“第三股力量”。

1.2.3 容器技术现状

Docker利用Linux 的一些内核机制例如 [cGroups](https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt)、命名空间和 [SElinux](http://selinuxproject.org/page/Main_Page) 来实现容器之间的隔离。起初 Docker 只是 [LXC](https://linuxcontainers.org/) 容器管理器子系统的前端，但是在 0.9 版本中引入了 [libcontainer](http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/)，这是一个原生的 go 语言库，提供了用户空间和内核之间的接口。

容器技术需要解决最为核心的问题是针对软件的创建、发布和运行。它通过将运行环境和应用程序打包到一起，来解决部署的环境依赖问题，整整做到跨平台的发布和使用。容器会比虚拟机更高效，因为它们能够分享一个内核和分享应用程序库。

最初有一家叫DotCloud的法国公司提供PaaS服务，他能对支持多种语言的运行环境，如Java、Python、Ruby等。可是在PaaS领域有太多巨头已经布局，DotCloud考虑如果不开源，很难与巨头竞争，所有就干脆将Docker项目开源，至少能在开源社区得到个好名声。2013年3月，Docker正式以开源形式发布，此举让容器领域有了新的春天，截止2015年11月，Docker在Github上收到25600个赞，超过6800次克隆，以及超过1100名代码贡献者，成为20个最具影响力的Github开源项目。目前，世界上几乎所有的科技公司都在拥抱以Docker为代表的虚拟化生态圈。

1.2.4 异构系统现状

当前图形设备处理器易于获得并且价格可以被广泛接受，被认为是搭建异构系统的理想硬件设备。由于GPU硬件架构的特点，如：晶体管数量庞大，只有少数逻辑控制和缓存部分，特别适合用来做高任务量且逻辑相对独立的矩阵运算。所以CPU是专门为顺序串行处理而优化的几个核心组成，而GPU则有用一个数以千记的更小、更高效的核心组成的大规模并行计算架构。

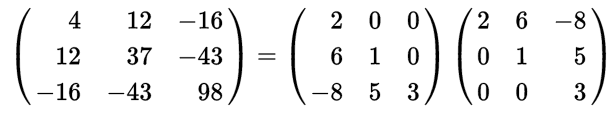
NVIDIA公司于2007年发明了基于CUDA的并行编程生态系统，它对C语言进行了扩展，实现了异构系统模型，因为CUDA提供了一整套开发生态系统，包括基于编译器，调试器，调优器，集成开发环境，和各类加速应用数学库等，让基于GPU的编程变得从此简单。

在2017年GPU技术会议上，NVIDIA向全球发布了CUDA 9，最新版本的CUDA提供了空前强大的并行计算平台和编程模型。比如合作组织，它重新定义了在kernel内部线性启动机制。还有针对NPP运行库的优化，相比Intel公司的多核至强系列处理器，在图像处理方面的性能提升了20-100倍。

1.3 论文主要目标

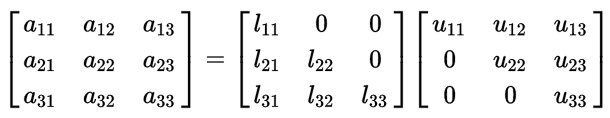
算法研究：

Cholesky定义：给定一个对称正定矩阵A，则存在主对角元素全为正数的下三角矩阵L满足：A=LLT，通常称下三角矩阵L为矩阵A的Cholesky因子。其中L是具有实数和正对角项的下三角矩阵，LT表示L的共轭转置。每个Hermitian正定矩阵[27]（以及每个实值对称正定矩阵）都具有独特的Cholesky分解。如果矩阵A是Hermitian和正半定的，则如果允许L的对角条目为零，则它仍然具有形式A = LLT的分解。下面对称实矩阵的Cholesky分解：



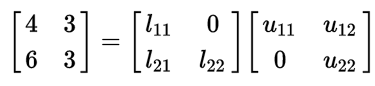
Cholesky分解主要用于线性方程的数值解Ax=B，如果A是对称和正定的，那么我们可以通过首先计算Cholesky分解A = LLT来求解Ax = b。然后通过正向替换求解Ly = b，最后通过反向替换求解x的LTx=y。对于可以置于对称形式的线性系统，Cholesky分解（或其LDL变体）是一种可以被选择的方法，因为它具有优异的效率和数值稳定性。与LU分解相比，它的效率大约快两倍。

LU定义：令A为方阵。 LU分解是指A的分解，具有适当的行和/或列排序或排列成两个因素，即下三角矩阵L和上三角矩阵U，A=LU，在下三角矩阵中，对角线以上的所有元素都为零，在上三角矩阵中，对角线以下的所有元素均为零。 例如，对于3×3矩阵A，其LU分解看起来像这样：

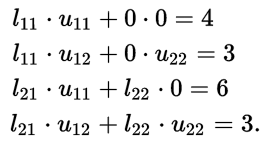


在矩阵中没有正确的排序或排列，因式分解可能无法实现。 例如，很容易通过扩展矩阵乘法来验证a11=l11u11。 如果a11=0，则l11u11中至少有一个必须为零，这意味着L或U是单数。如果A是非奇异的（可逆的），这是不可能的，可以通过简单地重新排序A的行来移除它，以便排列的矩阵的第一个元素不为零。

我将以下2乘2矩阵进行因式分解来举例：



找到这个简单矩阵的LU分解的一种方法是通过检查简单求解线性方程。扩展矩阵乘法如下所示，



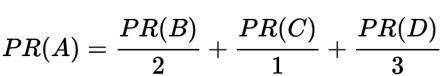
这个方程式是不确定的。 在这种情况下，L和U矩阵的任何两个非零元素都是解的参数，并且可以任意设置为任何非零值。 因此，为了找到唯一的LU分解，有必要对L和U矩阵进行一些限制。 例如，我们可以方便地要求下三角矩阵L成为单位三角矩阵（即将其主对角线的所有条目设置为一个）。

PageRank算法研究：PageRank是一种链接分析算法，它为诸如万维网的超链接文档集的每个元素分配数字加权，目的是“测量”其集合内的相对重要性[28]。该算法可以应用于具有相互引用和引用的实体的任何集合。它分配给任何给定元素E的数字权重称为E的PageRank，由PR(E)来表示。PageRank是基于由所有万维网页面作为节点和超链接创建的网页的数学算法，考虑到权限集线器，如cnn.com或usa.gov。等级值表示特定页面的重要性。一个页面的超链接将被视为支持投票。一个页面的PageRank被递归地定义，并且取决于链接到它的所有页面的数量和PageRank度量（“传入链接”）。由多页与较高PageRank相关联的页面本身具有高排名。

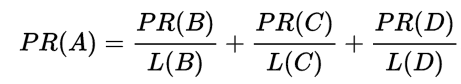
如果系统中唯一的链接是从B，C和D到A，每个链接将在下一次迭代时将0.25 PageRank转移到A，总共为0.75。



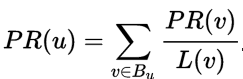
假设页面B链接到页面C和A，则页面C具有到页面A的链接，页面D链接到所有三个页面。因此，在第一次迭代时，页面B将其现有值的一半（0.125）转移到页面A，将另一半（0.125）传送到页面C。页面C将其所有现有值0.25传递到唯一 页面链接到A.由于D有三个出站链接，它会将其现有价值的三分之一（或大约0.083）转移到A.在此迭代完成后，第A页的PageRank约为0.458。



换句话说，出站链接授予的PageRank等于文档自己的PageRank得分除以出站链接数L()：



在一般情况下，任何页面的PageRank值可以表示为：



参考文献

1. Benedicic L, Cruz F A, Madonna A, et al. Portable, high-performance containers for HPC[J]. 2017.
2. Han T D, Abdelrahman T S. hiCUDA: High-Level GPGPU Programming[J]. IEEE Transactions on Parallel & Distributed Systems, 2010, 22(1):78-90.
3. Giunta G, Montella R, Agrillo G, et al. A GPGPU Transparent Virtualization Component for High Performance Computing Clouds[J]. 2010.
4. Walters J P, Chaudhary V, Cha M, et al. A Comparison of Virtualization Technologies for HPC[C]// International Conference on Advanced Information NETWORKING and Applications. IEEE, 2008:861-868.
5. Reaño C, Silla F. A Performance Comparison of CUDA Remote GPU Virtualization Frameworks[C]// IEEE International Conference on CLUSTER Computing. IEEE, 2015:488-489.
6. Nanda S, Chiueh T C. A Survey on Virtualization Technologies[J]. Rpe Report, 2005.
7. Vinaya M S, Vydyanathan N, Gajjar M. An evaluation of CUDA-enabled virtualization solutions[C]// IEEE International Conference on Parallel Distributed and Grid Computing. IEEE, 2013:621-626.
8. Boettiger C. An introduction to Docker for reproducible research[J]. Acm Sigops Operating Systems Review, 2015, 49(1):71-79.
9. Landaverde R, Zhang T, Coskun A K, et al. An Investigation of Unified Memory Access Performance in CUDA.[C]// High Performance Extreme Computing Conference. IEEE Conf High Perform Extreme Comput, 2014:1-6.
10. Brandt J, Chen F, Sapio V D, et al. Combining Virtualization, resource characterization, and Resource management to enable efficient high performance compute platforms through intelligent dynamic resource allocation[C]// IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, 2010:1-8.
11. Soltesz S, Pötzl H, Fiuczynski M E, et al. Container-based operating system virtualization[J]. Acm Sigops Operating Systems Review, 2007, 41(3):275-287.
12. Cramwinckel J, Singor S, Varbanescu A L. FiNS: A Framework for Accelerating Nested Simulations on Heterogeneous Platforms[M]// Euro-Par 2015: Parallel Processing Workshops. Springer International Publishing, 2015.
13. Merkel D. Docker: lightweight Linux containers for consistent development and deployment[J]. 2014, 2014(239).
14. Sanders J, Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming[M]. Addison-Wesley Professional, 2010.
15. Nvidia Corporation. Cg-3.1\_April2012\_ReferenceManual. 2012.
16. Cook S. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs[M]. Elsevier, MK, 2012.
17. Potluri S, Hamidouche K, Venkatesh A, et al. Efficient Inter-node MPI Communication using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs[C]// International Conference on Parallel Processing. IEEE, 2013:80-89.
18. Naumov M. Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS[J]. 2011.
19. N Corporation. CUDA-GDB: User Manual. 2017.
20. N Corporation. NVML.PDF. 2017.
21. N Corporation. CUDA Compiler Driver NVCC. 2017.
22. Dowd K. High performance computing[M]// High-performance computing /. Kluwer Academic/Plenum Press, 1993:39-55.
23. Lombardi F, Pietro R D. CUDACS: Securing the Cloud with CUDA-Enabled Secure Virtualization[C]// International Conference on Information and Communications Security. Springer-Verlag, 2010:92-106.
24. Corp I. Intel Virtualization Technology for Directed I/O[J].
25. Brodtkorb A R, Dyken C, Hagen T R, et al. State-Of-The-Art In Heterogeneous Computing[J]. Scientific Programming, 2015, 18(1):1-33.
26. Peng Wang. UNIFIED MEMORY ON P100. 2017.
27. Herrod S. The future of virtualization technology[C]// International Symposium on Computer Architecture. IEEE, 2006:352-352.
28. Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[C]// Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003:164-177.
29. 王伟, 郭绍忠, 王磊,等. 一种基于CPU-GPU异构计算的混合编程模型[J]. 信息工程大学学报, 2010, 11(6):674-678.
30. 沈聪, 高火涛. 使用 GPU 加速计算矩阵的 Cholesky 分解[J]. 计算机应用与软件, 2016, 33(9):284-287.
31. 陈波. 基于CPU-GPU异构平台的性能优化及多核并行编程模型的研究[D]. 中国科学技术大学, 2011.
32. 伍阳. 基于Docker的虚拟化技术研究[J]. 信息技术, 2016(1):121-123.
33. 梁添. 基于GPU的稀疏矩阵运算优化研究[D]. 华中科技大学, 2012.
34. 李繁. 基于GPU的高性能并行优化算法研究[D]. 大连理工大学, 2014.
35. 赵春晖, 许云龙, 黄辉. 基于LU分解的稀疏目标定位算法[J]. 电子与信息学报, 2013, 35(9):2234-2239.
36. 张玉洁. 基于多GPGPU并行计算的虚拟化技术研究[D]. 南京航空航天大学, 2015.
37. 汪恺, 张功萱, 周秀敏. 基于容器虚拟化技术研究[J]. 计算机技术与发展, 2015(8):138-141.
38. 温研, 王怀民. 基于本地虚拟化技术的隔离执行模型研究[J]. 计算机学报, 2008, 31(10):1768-1779.
39. 张云洲. 虚拟化环境下的GPU通用计算关键技术研究[D]. 南京航空航天大学, 2014.
40. MA Ye, 马业, YUAN Jia-Bin,等. 面向CUDA的GPU虚拟化研究及其应用[C]// 2012全国高性能计算学术年会. 2012.
41. 吕相文. 高性能计算云环境下GPU并行计算技术及应用研究[D]. 南京航空航天大学, 2015.
42. 陈国良, 毛睿, 蔡晔. 高性能计算及其相关新兴技术[J]. 深圳大学学报(理工版), 2015(1):25-31.
43. 王勇超. 高性能计算集群技术应用研究[D]. 西安理工大学, 2007.
44. 陈建平, Jerzy Wasniew ski. Cholesky分解递归算法与改进[J]. 计算机研究与发展, 2001, 38(8):923-926.
45. 金明心. 搜索引擎PageRank算法研究及其改进[D]. 西安理工大学, 2015.
46. 卢风顺, 宋君强, 银福康,等. CPU/GPU协同并行计算研究综述[J]. 计算机科学, 2011, 38(3):5-9.