

**硕士学位论文**

**开题报告**

**报告题目 \_面向异构计算平台的结构化并行程序设计方法研究**

学 号 2120151009

姓 名 廖心怡

导 师 计卫星

研究方向 高性能嵌入式计算

二级学科 计算机软件与理论

一级学科 计算机科学与技术

学 院 计算机学院

**2016年 12月 8 日**

**填表说明**

1．只有学籍状态为注册或悬置的研究生才允许开题。但学籍状态为悬置的研究生只有在完成注册手续之后，开题报告及其评审结果才能被认可。

2．硕士学位论文开题报告封面及一至八项必须用计算机输入、打印。

3．开题报告为A4大小，于左侧装订成册。6硕士研究生应逐项认真填写，各栏空格不够时请自行加页。

4．开题报告经指导教师审阅通过后，由硕士研究生在学科组或更大范围内宣读，并接受专家组质疑、评议。专家组由三名以上高级职称专家组成。开题报告应由硕士生导师为主体组成的评审小组评审。评审合格后，装订，归档，学院留存。

5．硕士研究生应在选题前阅读相关领域的中外文资料，并写出不少于4000字的文献综述报告，引用参考文献的篇数不得低于本学科专业培养方案的规定。文献综述报告应反映国际和国内本领域的研究历史、现状和发展趋势。文献综述报告是开题报告的必要附件，开题报告通过后，由学院留存。

6．“参考文献”著录按照GB7714-87文参考文献著录规则执行。书写顺序为：序号·作者·论文名或著作名·杂志或会议名·卷号、期号或会议地点·出版社·页号·年。

**一 简表**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 研究生简况 | 姓名 | | | 廖心怡 | | | | | | | | | | 性别 | | | | | 女 | | | | 出生年月 | | | | | 1993年5月 | | | | | | |
| 学号 | | | 2120151009 | | | | | | | | | | 入学时间 | | | | | 2015年9月 | | | | 身份证号 | | | | | 500109199305287121 | | | | | | |
| 学科、专业 | | | | | | | | 计算机科学与技术 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 本科毕业时间 | | | | | | | | 2015年6月 | | | | | | | | | | 本科毕业学校 | | | | | 北京理工大学 | | | | | | | | | | |
| 指导小组 | | | | | | | 姓名 | | | | | | | | 职称 | | | | 工作单位 | | | | | | | | | | | | | 签字 | | |
| 导师 | | | | | | | 计卫星 | | | | | | | | 副教授 | | | | 北京理工大学计算机学院 | | | | | | | | | | | | |  | | |
| 小组成员 | | | | | | | 石峰 | | | | | | | | 教授 | | | | 北京理工大学计算机学院 | | | | | | | | | | | | |  | | |
| 刘辉 | | | | | | | | 教授 | | | | 北京理工大学计算机学院 | | | | | | | | | | | | |  | | |
| 王一拙 | | | | | | | | 讲师 | | | | 北京理工大学计算机学院 | | | | | | | | | | | | |  | | |
| 高玉金 | | | | | | | | 讲师 | | | | 北京理工大学计算机学院 | | | | | | | | | | | | |  | | |
| 研究课题 | | 名称 | | | 中文 | | | | | | 面向异构计算平台的结构化并行程序设计方法研究 | | | | | | | | | | | | | | | | | | | | | | | |
| 英文 | | | | | | Research on structured parallel programming methodology for heterogeneous computing | | | | | | | | | | | | | | | | | | | | | | | |
| 类别 | | | 国家项目（ √ ）： 部（省）项目（ ）； 企业项目（ ）；  自拟项目（ ）； 是否兵器类项目（ ）； 是否涉密（否，密级： ） | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 性质 | | | 基础研究（ ）； 应用基础研究（ √ ）； 应用技术研究（ ） | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 与导师课题研究课题的关系 | | | | | | | | 是导师研究课题的一部分（√ ）  与导师研究课题无关（ ） | | | | | | | | | | | | | | | | | | | | | | | | |
| 摘 要 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | | | 随 | | 着 | | | | 计 | 算 | | | 机 | 技 | 术 | | 的 | 不 | 断 | | | 发 | 展 | ， | | 传 | 统 | 并 | | 行 | 编 |
| 程 | 已 | | | 不 | | 能 | | | | 满 | 足 | | | 异 | 构 | 计 | | 算 | 日 | 益 | | | 增 | 长 | 的 | | 需 | 求 | ， | | 而 | 脚 |
| 本 | 语 | | | 言 | | 作 | | | | 为 | 目 | | | 前 | 最 | 为 | | 流 | 行 | 的 | | | 编 | 程 | 语 | | 言 | 之 | 一 | | ， | 其 |
| 较 | 好 | | | 的 | | 移 | | | | 植 | 性 | | | 、 | 对 | 其 | | 它 | 编 | 程 | | | 语 | 言 | 的 | | 黏 | 合 | 性 | | 能 | 够 |
| 有 | 效 | | | 的 | | 使 | | | | 程 | 序 | | | 员 | 开 | 发 | | 出 | 编 | 程 | | | 模 | 型 | 并 | | 验 | 证 | 。 | | 因 | 此 |
| ， | 为 | | | 了 | | 提 | | | | 高 | 异 | | | 构 | 系 | 统 | | 的 | 编 | 程 | | | 效 | 率 | ， | | 本 | 文 | 提 | | 出 | 一 |
| 种 | 面 | | | 向 | | 异 | | | | 构 | 计 | | | 算 | 平 | 台 | | 的 | 结 | 构 | | | 化 | 并 | 行 | | 编 | 程 | 设 | | 计 | 方 |
| 法 | 。 | | | 该 | | 方 | | | | 法 | 使 | | | 用 | 脚 | 本 | | 语 | 言 | ， | | | 能 | 对 | 异 | | 构 | 硬 | 件 | | 系 | 统 |
| 进 | 行 | | | 建 | | 模 | | | | ， | 采 | | | 用 | 结 | 构 | | 化 | 并 | 行 | | | 程 | 序 | 设 | | 计 | 方 | 法 | | ， | 能 |
| 够 | 实 | | | 现 | | 跨 | | | | 语 | 言 | | | 层 | 次 | 的 | | 任 | 务 | 级 | | | 并 | 行 | 和 | | 调 | 度 | 。 | |  |  |
| 关  键  词 | | 1.关键词限3~5个；2.关键词之间用“；”分隔。 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 中文 | | | | | | | | 异构计算；结构化并行；脚本语言；并行编程 | | | | | | | | | | | | | | | | | | | | | | | | |
| 英文 | | | | | | | | heterogeneous computing；structured parallel；script language；parallel programming | | | | | | | | | | | | | | | | | | | | | | | | |

**二 选题依据**

|  |
| --- |
| 2.1. 研究背景  异构计算被视为计算机处理器继单核、多核之后的第三个时代，它将实现使用不同类型指令集和体系架构的计算单元，比如CPU和GPU之间的“协同计算、彼此加速”，从而突破CPU发展的瓶颈，有效解决能耗、可扩展性等问题[1]。异构多核系统的显著特点是在同一个芯片上集成了大量的同构或异构的并行计算单元，并使用片上互连网络NoC(Network-on-chip)将计算内核、特殊加速部件和存储系统连接起来，具有较低的功耗和强大的计算能力，而异构计算系统则可以拥有多个芯片。由于存在多个不同的异构计算单元，异构系统的出现使得系统面临着多个不同的并行程序设计模型，并行编程模型一直是并行计算研究领域中的重点内容[2]，这些模型包括针对通用共享存储系统的PThreads和OpenMP[3]，针对GPU的OpenCL和CUDA等。应用程序设计者不仅关心系统的性能，还关心系统可靠性和实时性，因此软件开发人员需要掌握多种编程模型，熟悉硬件平台和应用领域，这些因素最终使得并行软件设计的复杂性急剧上升。  结构化程序设计由迪克斯特拉（E.W.dijkstra） 在1969年提出，是以模块化设计为中心，将待开发的软件系统划分为若干个相互独立的模块，这样使完成每一个模块的工作变单纯而明确，为设计一些较大的软件打下了良好的基础。而结构化并行编程是类似于使用在串行编程中的结构化控制流模式，正如结构化控制流取代了大多数程序中的goto语句[4]，结构化并行编程模式则可能替代低级和特定体系结构的并行机制，比如线程和向量函数。结构化并行编程有着额外的好处，那就是能够避免一些非决定性的，或者是一些在性能中不可避免的且十分重要的常见异常。  任务并行编程模型是近年来多核平台上广泛研究和使用的并行编程模型[5][6]，其目的是为了简化并行编程并且提高多核利用率。任务并行编程模型的基本单位是并行的任务，它提供了任务的划分和同步的编程接口，这样任务划分和同步的工作都是交给程序员来完成的。用户可以把应用程序划分成大量的细粒度任务。但是，任务之间在执行过程中到底是并行执行还是串行执行、在哪个物理核上执行以及如何实现任务之间的同步则是由运行时系统来完成的。任务并行编程模型的主要优点有：（1）. 为程序员提供了两类并行控制结构，分别是支持规则并行的循环并行控制结构和支持非规则并行的嵌套并行控制结构。这两类并行控制结构均能实现数据并行模式和任务并行模式；（2）. 逻辑任务与物理线程分离[5]。程序员只需考虑如何划分逻辑任务，使用合适的并行控制结构并行逻辑任务，而不需要考虑任务调度和处理器核数，提高编程层次，简化并行编程；运行时系统负责任务调度，每个核对应一个物理线程，每个物理线程会执行许多逻辑任务．这种在用户态空间进行的任务调度大大降低了调度的开销，从而提高了多线程程序的执行效率。同时，运行时系统采用任务窃取调度算法获得负载平衡，提高多核的使用效率。  脚本语言是目前最为流行的编程语言之一，它易于编程的特点使其广受编程人员的喜爱。然而，执行时间并不是在开发中选择语言的唯一标准。在考虑到计算机的体系结构时，由于编程模型本身并没有一个统一的标准，那样使编程模型与硬件的对应关系变得模糊，同时编程人员需要考虑到开发时间带来的消耗。在异构计算系统中，存在着多种多样的设备，但是脚本语言本身与平台无关，具有较好的移植性。此外，脚本语言能够有效的自动管理资源，并且能够作为黏合剂工作在其他语言代码块中。目前脚本语言拥有数量居多的开源库，就拿Python为例，NumPy，SciPy，Matplotlib都是大众熟知并运用广泛的库，因此，编程人员能够用脚本语言迅速的开发出编程模型且进行验证。然而，脚本语言的运行性能较差，但是由于在异构系统中CPU主要是起到控制任务的作用，脚本语言的运行速度足以满足CPU要求的1000个任务每秒的控制速度。  在此研究背景下，为了提高异构系统的编程效率，本文提出了一种面向异构计算平台的结构化并行编程方法。该方法能使用脚本语言支持异构硬件系统进行建模，采用结构化并行程序设计方法，能够实现跨语言层次的任务级的并行与调度。  2.2. 研究现状及发展趋势  2.2.1异构计算的研究现状及发展趋势  关于CPU+GPU的这种最为常见的异构计算平台，目前也有着多种的存在形式。根据是否共享内存可以分为两种情况：一种是CPU与GPU共享内存，如AMD公司推出的的APU，虽然CPU和GPU共享着同一块内存，但是CPU的线程并不能和GPU的线程共享同一块数据。在HPCChina’2012上，AMD公司表示不久的将来他们会完成CPU线程与GPU线程共享数据。另一种情况则是CPU和GPU完全保有独立的存储单元，他们之间使用PCIE通信。Nvidia公司在提高GPU计算单元的访存和计算速度，以及计算单元的个数等方面下功夫。另外，除了GPU这种已经流行的协处理器之外，Intel公司研发的协处理器MIC，也在异构计算的领域中占有一席之地。  开放计算语言[7]（Open Computing Language, OpenCL）是一个由非盈利性组织技术联盟Khronos Group进行管理的异构编程框架。其支持多种层次的并行，可以高效映射到同构或者异构的体系结构上。OpenCL提供了两种并行计算方式，分别是基于任务和基于数据当前支持的CPU包括X86，ARM和PowerPC，同时支持AMD和NVIDIA的显卡。OpenCL是一个为异构设备编写程序的标准框架，它的出现虽然增加了程序在各平台之间的可移植性，但是在多个计算设备协同计算时，多种计算设备的调用、设备间的负载均衡等问题严重制约了它的普及。因此，OpenCL对异构系统下多设备并行计算的研究具有重要的意义。  PyOpenCL是OpenCL的Python编程环境，是OpenCL在Python上的实现。在论文[8]中，作者想要通过无缝执行，从大容量的计算任务中生成低级别C或者类C的源代码，提出了一种名为GPU运行时代码生成（RTCG）的技术，结合两种已有的开源工具包，PyCUDA和PyOpenCL。GPU运行时代码生成作为元程序的一种形式，其主要操作原则为开发人员负责创意和脚本代码，其余的GPU代码相关都交给机器完成。同时，作者开发了PyCUDA和PyOpenCL两种工具包，在基于库的情况下，开发人员可以在Python文件中直接调用相应的库函数来完成CUDA和OpenCL的相应程序开发，然后在GPU上执行。  OpenACC(for Open Accelerators)是一个由Cray，CAPS，Nvidia和PGI公司共同开发的并行计算编程标准。设计该标准的目的是简化异构的CPU/GPU系统的并行编程。和OpenMP一样，程序员可以在C，C++和Fortran源代码上通过添加注释的方法来识别应该加速的区域。和OpenMP4.0以上的版本一样，代码可以同时在CPU和GPU上运行。  2.2.2 结构化并行编程的研究现状和确定性验证  目前确定性并行主要通过两种方法来实现，其一是通过运行时系统来保证程序的确定性执行，例如Kendo[9]、DMP[10]、CoreDet[11]和Grace[12]，这些方法所采用的复杂运行时机制需要更多的系统资源，因此不适合于嵌入式实时系统。另外一种是通过程序设计语言从语义上保证，Robert L. Bocchino等人认为确定性应该是并行程序应具有的属性，并基于Java提出了一种类型推理系统，在用户对程序标注的基础上，通过编译器的静态分析确保程序的确定性执行[13]。该方法依赖于程序员对并行程序的正确标注，而这种标注又是建立在对并行程序正确理解和推理的基础上的。Michael D. McCool建议采用确定性并行模式及其组合来编写并行程序[14]，他所提到的并行模式既包括数据并行模式，例如forall，map-reduce；也包括任务并行，例如CoBegin，Pipeline等。McCool同时指出，并不是所有的并行模式都具有确定性的语义。在确定性程序设计语言出现之前，许多并行程序设计语言从语言级提供了对部分常见并行结构的支持，例如Cilk[15]中的spawn/sync，Habanero Java[9]和X10[16]中的async/finish和forall/foreach。  虽然使用foreach、async/finish等并行结构及其组合编写的并行程序具有确定性的语义，但是不能保证每个程序员都能正确的使用这些并行构造。例如foreach的使用假设每个循环迭代之间是数据无关的，但是实际上不是每个程序都符合这样的要求，因此需要通过静态分析或者动态执行来对数据无关的假设进行再次验证。由于结构化并行程序中并行任务之间具有固定的组合模式，因此能够极大的简化并行程序的推理和分析。Martin Vechev等人基于结构化的Habanero Java程序提出一种静态分析方法[17]。J. Mellor-Crummey针对split-merge程序提出基于Vector Clock的数据竞争检测算法[18]。这两种方法不能找到程序中所有的数据竞争且无法保证程序的确定性执行。Cilk研究小组针对spawn/sync提出Nondeterminator数据竞争检测算法[19]，Habanero Java研究小组针对async/finish提出SPD3算法[15]，良好嵌套的并行Ruby程序所设计的TARDIS算法[20]等都是精确和完全的数据竞争检测算法，即对于给定的输入，可以通过程序的一次执行找到程序中所有的数据竞争。Nondeterminator是一个串行算法，只能通过单个线程的执行来查找数据竞争；而SPD3是以一个针对async/finish设计的并行算法，与Nondeterminator相比具有较好的扩展性。TARDIS是针对DoAll和CoBegin设计的并行数据竞争检测算法。  已有工作在不同并行程序设计语言环境中对部分并行结构分别展开研究，提出了不同的确定性验证机制。现有工作认为并发任务具有相同的优先级，且多采用运行时任务划分与调度机制；嵌入式实时系统中经常需要对视频、音频和其他信息进行流水处理，但是目前并没有关于流水并行Pipeline的确定性定义和相应的数据竞争检测算法；此外，已有工作中尚未发现关于其他具有确定性语义并行结构的确定性验证方法，例如Reduction[21]和Future[22]等  在论文[14]中，作者介绍了算法框架的首次提出[23][24]和详细的说明[25]，并且定义了并行模式为特定的重复配置的计算和数据访问。随着技术的发展，研究者们发现结构化编程导致了大多数程序中的goto语句的淘汰[4]，同理可知，结构化并行模式可以淘汰显式化线程和同步的需求。作者们详细介绍了一系列有用的结构化串行和并行编程模式，强调了为了支持系统的发展确定性模式自动避免了不安全的数据竞争和死锁。  在论文[26]中，作者提出一种并行程序模型，其结合了目前两种知名执行模型，即单指令多数据流（SIMD）和单程序多数据流（SPMD）。该模型能够在全局地址空间中支持SIMD类型的数据并行，同时在局部地址空间支持SPMD类型的任务并行。这样由于在一个并行程序中同时集成了SIMD和SPMD的支持，数据并行和任务并行的应用都可以在应用于Python。这个模型提高了DistNumPy的并行度和伸展性，通过运行时的层次并行，使用OpenMP实现多线程，MPI实现节点间通信。然而，由于模型是在SIMD和SPMD的混合模式下，DistNumPy的自动通信延迟隐藏变得不可用，并且很难解决由于引用全局地址空间带来的伸缩性限制。作者以后会将关注点放在解决这些问题上。  在论文[27]中，作者提出了一种针对并行计算的python框架叫做Hydra。该框架证实了使用一种python的并发框架来开发一个CSP到python翻译器的可行性。整个框架使用了ANTLR[28]编译生成工具，Python Remote Object和PyCSP[29]。作者通过用CSP定义一个算法，然后将其转换成Python，最后使用多个python解释器实例来执行程序的方法验证了Hydra原型的可行性。然而，想要使用该框架，程序员必须熟悉CSP，否则对程序员而言需要花时间学习使用CSP，这又将成为另一种负担。其次，若是想要调试程序，将变得无比艰难，因为无法将问题归结于是运行时出现还是编写的CSP算法上。由于该框架使用创建多个进程的方法来解决并行问题，在创建进程的同时系统也为每个进程创建了对应的数据结构，资源管理器等等，带来了相应的空间开销和时间开销，同时也因为需要协调各个进程整个系统变得更难控制。 参考文献 [1] 马俊峰. 基于OpenCL的多GPU并行计算的研究与应用[D]. 哈尔滨理工大学, 2014.  [2] 富弘毅. OpenMP并行程序容错技术研究[D]. 国防科学技术大学, 2010.  [3] Dijkstra E W. Go To Statement Considered Harmful[C]// Springer-Verlag New York, Inc. 1968:147-148.  [4] 袁良. 局部性感知的并行计算模型与并行编程模型研究[D]. 中国科学院大学, 2012.  [5] 王蕾, 崔慧敏, 陈莉,等. 任务并行编程模型研究与进展[J]. 软件学报, 2013, 24(1):77-90.  [6] 赵世韬. 任务并行编程模型研究与进展[J]. 电子技术与软件工程, 2016(12):165-165.  [7] BenedictR.Gaster, 贾斯特, 张云泉. OpenCL异构计算[M]. 清华大学出版社, 2012.  [8] Klöckner A, Pinto N, Lee Y, et al. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation[J]. Parallel Computing, 2011, 38(3):157-174.  [9] Olszewski M, Ansel J, Amarasinghe S. Kendo: efficient deterministic multithreading in software[J]. Acm Sigplan Notices, 2009, 44(3):97-108.  [10] Deviettip J, Lucia B, Ceze L, et al. DMP: Deterministic Shared Memory Multiprocessing[C]// International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2009, Washington, Dc, Usa, March. 2009:85-96.  [11] Bergan T, Anderson O, Devietti J, et al. CoreDet: a compiler and runtime system for deterministic multithreaded execution[J]. Acm Sigarch Computer Architecture News, 2010, 38(1):53-64.  [12] Berger E D, Yang T, Liu T, et al. Grace: safe multithreaded programming for C/C++[J]. Acm Sigplan Notices, 2009, 44(10):81-96.  [13] Bocchino R L, Adve V S, Dig D, et al. A type and effect system for deterministic parallel Java[J]. Acm Sigplan Notices, 2009, 44(10):97-116.  [14] McCool, Michael D. Structured parallel programming with deterministic patterns[J]. Dr Dobbs Journal, 2010:5-5.  [15] Frigo M, Leiserson C E, Randall K H. The implementation of the Cilk-5 multithreaded language[J]. Acm Sigplan Notices, 1998, 33(5):212-223.  [16] Charles P, Grothoff C, Saraswat V, et al. X10: An object-oriented approach to Non-Uniform Cluster Computing[C]// Companion To the, ACM Sigplan Conference on Object-Oriented Programming, Systems, Languages, and Applications. 2005:519-538.  [17] Vechev M, Yahav E, Raman R, et al. Automatic Verification of Determinism for Structured Parallel Programs[C]// International Conference on Static Analysis. Springer-Verlag, 2010:455-471.  [18] Mellorcrummey J. On-the-fly detection of data races for programs with nested fork-join parallelism[C]// Supercomputing, 1991. Supercomputing '91. Proceedings of the 1991 ACM/IEEE Conference on. IEEE, 1991:24-33.  [19] Feng M, Leiserson C E. Efficient Detection of Determinacy Races in Cilk Programs[J]. Theory of Computing Systems, 1999, 32(3):301-326.  [20] W Ji, L Lu, M Scott. TARDIS: Task-level Access Race Detection by Intersecting Sets, WoDet’2013.  [21] J Shirako, V Cave, J Zhao and V Sarkar. Finish Accumulators: a Deterministic Reduction Construct for Dynamic Task Parallelism, WoDet’2013.  [22]. Jun Shirako, Vincent Cave, Jisheng Zhao and Vivek Sarkar. Finish Accumulators: a Deterministic Reduction Construct for Dynamic Task Parallelism, WoDet’2013.  [23] Cole M I. Algorithmic skeletons: a structured approach to the management of parallel computation[M]. 1988.  [24] Cole M. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming[J]. Parallel Computing, 2004, 30(3):389-406..  [25] Skillicorn D B, Talia D. Models and languages for parallel computation[J]. Acm Computing Surveys, 1970, 30(2):123-169.  [26] Kristensen M R B, Zheng Y, Vinter B. PGAS for Distributed Numerical Python Targeting Multi-core Clusters[J]. I P D P S Proceedings, 2012, 19:680-690.  [27] Tristram W B, Bradshaw K L, Tristram W B, et al. Hydra: A Python Framework for Parallel Computing.[C]// Communicating Process Architectures Conference, Cpa 2009, Organised under the Auspices of Wotug, Eindhoven, the Netherlands, 1-6 November. 2009:311-324.  [28] Parr T. The Definitive ANTLR Reference: Building Domain-Specific Languages[M]. Pragmatic Bookshelf, 2007.  [29] Mcewan A A, Schneider S, Ifill W, et al. PyCSP- Communicating Sequential Processes for Python[J]. Communicating Process Architectures, 2007, 65:229-248. |

**三 研究内容**

|  |
| --- |
| 1. 研究目标   基于脚本语言，面向异构平台设计并实现一种结构化并行编程方法，在不影响运行效率的前提下，能够提高异构计算平台的编程效率。   1. 研究内容   为了实现本课题的研究目标，需要研究的内容如下：  （1）异构计算平台的硬件设备建模：由于目前异构系统中拥有各种型号、计算能力各不相同的硬件设备。为了在系统中区分这些硬件并且访问使用它们，需要对这些硬件设备进行建模。  （2）软件建模：由于结构化并行程序中并行任务之间具有固定的组合模式，因此能够极大的简化并行程序的推理和分析。我们需要研究如何将这些软件任务进行建模。  （3）映射部分：当硬件建模和软件建模都结束以后，我们需要研究如何使用脚本语言将软件映射到对应的硬件设备上然后执行。  （4）任务调度：由于存在着多个任务同时进行，为了提高编程效率，最大化的利用异构计算系统的硬件设备，我们需要对任务进行调度。 |

**四 研究方案**

|  |
| --- |
| **4.1研究方法**  **4.1.1 方法框架**  异构计算平台下结构化并行编程方法的框架设计如图1所示：  Platform  Device  Platform  Device  Platform  Device  Device  Platform  Device  Device  Task  Task  Task  Task  Task  Task  Task  Task  Task  Software  Hardware  Mapping  图1 异构计算平台下结构化并行编程方法框架  该方法框架分为三部分，硬件建模部分，软件建模部分以及映射部分，我们将在下面三个小节中详细说明。  **4.1.2 硬件建模**  在多核异构系统中，存在着多种多个的计算设备。在我们的建模中，我们将所有的硬件设备构造出一棵设备树。如图2所示，设备树上的每个设备拥有唯一的名字，用户可以通过名字指定设备来执行任务。  System  System\_Platform1  System\_Platform2  System\_Platform1\_device1  System\_Platform1\_device2  System\_Platform2\_device1  图2 一棵异构计算平台下的硬件设备树  此外，我们还应考虑每个设备的计算能力，并在建模中体现出来，该参数在后续结构化并行编程和多任务调度中具有重大作用。  **4.1.3 软件建模**  Python软件实现一般分为两种：基于库的实现和基于语言的实现。基于库的实现一般是指程序员自行编写了一些函数库，然后在其他文件中调用该库中的函数。而基于语言的实现则是指通过修改Python虚拟机，从而实现目标软件。Python与C/C++的混合编程包括在Python和调用C/C++的API或者在C++中调用Python的脚本。Python调用C库比较简单，将C语言代码不经过任何封装打包成so，再调用Python的ctypes就可以了，而如果Python要调用C++(类)动态链接库也只能通过需要extern "C"来辅助，也就是说还是只能调用C函数，不能直接调用方法，但是能解析C++方法。而C/C++调用Python就比较简单了，只需要把Python代码当做文本形式的动态链接库即可。  PyOpenCL是OpenCL的Python封装，其开发者就是采用编写一个函数库的方式，实现了在Python中OpenCL的各个接口的调用。我们的软件建模的实现就将建立在PyOpenCL架构基础之上。  常见的结构化并行编程模式有以下四种：  Co-begin：在并行编程中，co-begin函数是指多个任务同时执行，但是需要这些任务全部执行结束后才能继续执行其他的任务。  Para-for：Parallel-for算法重复地以并行方式执行相同的任务。 其中每个任务都由迭代值进行参数化。 当一个循环体不在该循环的各个迭代之间共享资源时，此算法很有用。  Future：Future是指主任务在运行时创建一个子任务用于并行计算一些数据值，该子任务计算完毕后等待主任务访问使用其计算结果。  Pipeline：流水线技术是目前广泛应用于微处理芯片(CPU)中的一项关键技术。在有一系列数据的输入下，我们可以使用流水线方法来处理这些数据。每一个阶段处理一种任务，每个阶段接受上一阶段的结果作为输入，并且将其输出结果传到下一阶段。需要注意的是，在流水线中的每个执行周期里，阶段之间都是并行实执行的，在该执行周期结束之后，则需要等所有的阶段执行结束后才继续执行下一个执行周期。流水线技术通常适用于流式数据的分析和处理。在我们的建模中，pipeline函数是指多个任务以流水线式存在，第一个任务接受输入处理后将输入传递给第二个任务，第二个任务再传递给第三个，以此类推。  **4.1.4 映射**  在映射部分，我们使用脚本语言完成整个映射过程。脚本语言是一种编程语言，其支持脚本和编写程序在一个特殊的运行时环境上，该环境能够自动化的执行任务，或者可以有操作员一条一条顺序执行。脚本语言是通常是解释执行而不是编译。脚本语言拥有快速开发、容易部署、易于已有技术集成、易学易用、实时性等特点，它可以作为一个极好的黏合剂来黏合系统中软件部分与硬件部分，即实现软件到硬件的映射。  整个映射的过程如下：  程序员首先确定需要执行的任务，然后通过脚本语言，指定这些任务的并行执行顺序，接着需要指定哪一个（一些）任务在那一块（一组）设备上运行，执行脚本语言程序，这样就实现了不同的任务在不同类型的设备上运行的映射。  我们设计了四种不同的映射方式来完成整个映射，如图3所示：  Task  Device1  Device2  MapOtoN(task,devices)  Device1  Device2  MapNtoN(tasks,devices)  Task  Device  MapOtoO(task,device)  Task1  Device  MapNtoO(tasks,device)  Task2  Task1  Task2  图3 四种软件到硬件的映射方式  MapOtoO(task,device):将单个任务映射单个设备上。  MapOtoN(task,devices):将单个任务映射到多个设备上。  MapNtoO(tasks,device):将多个任务映射到单个设备上。  MapNtoN(tasks,devices):将多个任务映射到多个设备上。  值得注意的是，在MapNtoN这种映射方式中，多个任务被映射到多个设备上，该方法则需要确定每个设备的计算能力以调度各个任务的执行顺序，获得最好的性能。  在该方法中，我们需要实现每种并行编程模式到硬件设备的映射。由于每种模式都是多任务的，在多任务多设备的条件下，我们需要找到合适的调度方法来得到系统最好的编程效率。  **4.2. 方法评估**  在计算机体系结构领域，我们常使用加速比（speedup）来衡量并行系统或程序并行化的性能和效果。加速比指是同一个任务在单处理器系统和并行处理器系统中运行消耗的时间的比率，其计算公式如下：  其中，T1是单处理器下的运行时间，Tp是在有P个处理器并行系统中的运行时间。  在我们的方法中，由于存在着多任务和结构化并行编程，因此我们采用以下公式来计算方法的加速比：  其中，T1,T2,…Tn是每个任务在异构计算平台下的运行时间，Tp是使用我们的方法后所有任务在异构计算平台下的运行时间。  我们也可以使用编程效率来评估我们的方法。简单的评估指标可以有代码行数、单个程序编写时间等等。  **4.3. 可行性分析**  目前异构计算技术已经有了一定的发展，结构化并行编程技术也在稳步向前，前人的研究成果有着坚实的基础。对已有文献的方法进行比较学习，提出异构计算平台下的结构化并行编程方法切实可行。  此外，本实验是在北京理工大学计算机学院嵌入式高性能计算实验室中进行，为本课题的研究工作开展提供了所需要的软硬件平台以及良好的工作环境和条件。实验室具有良好的实验基础和实验环境，为课题的编码和运行环境提供保障。 |

**五 研究工作进度安排**

|  |
| --- |
| 本课题研究大致要经历以下几个阶段：  2016年4月-2016年8月，选题，包括确定研究内容、意义和方案等；  2016年9月-2016年12月，相关文献收集调研学习，确定算法方向；  2016年12月-2017年3月，提出算法思路并探索其可行性；  2017年4月-2017年6月，进行编码实现；  2017年7月-2017年9月，测试算法的性能；  2017年10月-2017年12月，毕业论文撰写；  按照以上的进度要求，到2017年12月前能够完成本课题的全部研究内容。 |

**六 预期研究成果**

|  |
| --- |
| 本科题的预期成果主要有以下几个方面：  1.设计实现一种异构计算平台下结构化并行编程方法。  2. 在国内外重要期刊或学术会议上发表1篇论文。 |

**七 本课题创新之处**

|  |
| --- |
| 本课题的创新点总结如下：   1. 利用结构化并行程序设计方法实现了任务并行； 2. 由于脚本语言的易于编程、较好的移植性、粘合性等优点，使用脚本语言进行并行编程设计，在不影响运行效率的前提下，有效的提高编程效率。 |

**八 研究基础**

|  |
| --- |
| 1.与本项目有关的研究工作积累和已取得的研究工作成绩。  作者本人已学习Python，OpenCL等语言，熟练操作异构计算平台下的编程，阅读了大量的相关文献，对在该领域进行研究的世界范围内的课题组合研究进展有了一定的跟踪和认识。本人具有一定的编程能力，在算法实现方面有足够的能力。  2.已具备的实验条件，尚缺少的实验条件和解决的途径（包括利用国家重点实验室和部门开放实验室的计划与落实情况）。  本课题将在北京理工大学计算机学院嵌入式高性能计算实验室进行，该实验室拥有多台异构多核系统的计算机和嵌入式设备，为本课题的研究工作开展提供了所需的软硬件平台和良好的工作环境和条件。  3.研究经费预算计划和落实情况。  实验室研究条件能够满足本课题研究需要，无需另外购置其他设备和软件。 |



**硕士学位论文开题报告——导师意见**

|  |  |
| --- | --- |
| 学籍状态 | ■𞉀注册 □ 悬置 |
| 学位论文是否已被批准为涉密论文 | □ 是 ■否 |
| 导师对开题报告的审阅意见：  随着计算机技术的不断发展，异构计算技术得到广泛的应用，传统的并行编程模式已经不能满足异构计算应用日益增长的需求。廖心怡同学提出的面向异构计算平台使用结构化并行编程设计方法，该课题具有较好的应用前景和应用价值。  在开题报告中，廖心怡同学对课题相关的研究工作进行了全面和深入的调研，在此基础之上，给出了一种面向异构计算平台的结构化并行程序设计方法的研究方案，并且对该方案的技术路线和实验方法进行了详细的阐述，为后续研究工作的顺利开展奠定了坚实的基础。  开题报告阐述清楚，方案切实可行，计划明确，同意廖心怡同学就该课题开展进一步深入研究。  签字：计卫星  2016年12月8日 | |

说明：本页全部由指导教师填写。



**硕士学位论文开题报告评审表**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学号 | | 2120151009 | | | 姓名 | | 廖心怡 | | 导师姓名 | 计卫星 | |
| 所在学院 | | | | 计算机学院 | | | 学科、专业 | | 计算机科学与技术 | | |
| 课程学习情况 | | | | 已修课程学分 | | | 31 | | 待修课程学分 | | 0 |
| 选题名称 | | | | 面向异构计算平台的结构化并行程序设计方法研究 | | | | | | | |
| 课题经费来源 | | | | 国家自然科学基金青年基金 | | | | | | | |
| 开题报告时间 | | | | 2016年12月8日 | | | | | | | |
| 评审组成员 |  | | 姓 名 | | | 职 称 | | 工作单位 | | | 签 字 |
| 组长 | | 石峰 | | | 教授 | | 北京理工大学计算机学院 | | |  |
| 组员 | | 刘辉 | | | 教授 | | 北京理工大学计算机学院 | | |  |
| 计卫星 | | | 副教授 | | 北京理工大学计算机学院 | | |  |
| 王一拙 | | | 讲师 | | 北京理工大学计算机学院 | | |  |
| 高玉金 | | | 讲师 | | 北京理工大学计算机学院 | | |  |
| 评审组意见：    廖心怡同学的开题报告选题目的明确，实用性强，具有较高的研究价值。报告中阐述了使用结构化并行程序设计方法对异构计算平台中的编程的重要作用，和设计实现一种面向异构计算平台的结构化并行程序设计方法的意义。根据要实现的内容，给出了解决方案和研究途径。该方案合理，切实可行，在现有的实验条件下可以实现。  该同学在开题过程中表达清晰，逻辑性强，较好的回答评审组的问题，具备开题条件，同意进入课题研究工作。  组长签字： 2016 年 12 月 8 日 | | | | | | | | | | | |