

**硕士学位论文开题**

**文献综述报告**

**报告题目 \_面向异构计算平台的结构化\_**

**\_\_\_\_\_并行编程设计方法研究\_\_\_\_**

学 号 **\_ 2120151009**

姓 名 **\_\_ 廖心怡\_ \_\_**

导 师 **\_\_ 计卫星\_\_ \_**

研究方向 **\_嵌入式高性能计算\_\_**

二级学科 **\_计算机科学与技术\_\_**

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**

**文献综述报告要求与打印格式说明**

1. 文献综述报告应符合硕士研究生所在学科培养方案的要求。
2. 文献综述报告的内容不再在开题报告中重复，需单独在必修环节中上传。
3. 文献综述报告必须对相关领域已取得之成果进行归纳总结，结合学位论文选题对相关领域未来的发展和研究提出自己的观点。
4. 打印用纸：A4；**装订后，学院归档留存备查。**
5. 页眉为“北京理工大学硕士学位论文开题文献综述报告”； 加黑宋体，小3号，居中。页码居右排版；
6. 页面设计：页眉2.5cm，页脚1.5cm，左边距3cm，右边距2.4cm，正文用宋体，小4号，行间距26磅。

# 研究背景概述

1.1异构计算系统

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。由于存在多个不同的异构计算单元，异构多核的出现使得同一个系统面临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。

1.2结构化并行编程

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有潜力来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等。这样的结构化并行编程有额外的好处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。

目前越来越多的并行程序设计语言提供了一些特殊的并行语言结构（Parallel Language Construct）来简化并行程序的设计，例如Cilk中的spawn/sync[5]，Habanero Java中的async/finish[5]，Li Lu等人在Ruby中实现的DoAll和CoBegin[6]等。这些并行程序设计语言结构是已有经典并行模式的总结和抽象[7]，不仅能够方便的表述任务并行和数据并行，且使用这些结构编写的并行程序具有良好的嵌套结构，因此简化了程序的推理和分析；另外，用户只负责用这些并行结构标识程序中“哪些是可以并行的”以及“如何并行”，但是不需要说明并行的具体过程。并行任务的创建、管理、调度和同步由语言的运行时系统来保证和完成，既减少了出错的可能性，也使程序具有较好的可扩展性和可移植性。更为重要的一点在于，近期研究表明，部分并行结构（async/finish, spawn/sync, DoAll和CoBegin）不仅具有确定性的语义[8]，且对于给定的输入，能够使用数据竞争检测算法通过程序的一次执行够精确地找到所有的数据竞争[6][9][10]，为程序的确定性验证以及确定性执行提供了有力的保证。

1.3脚本语言

脚本语言是目前最流行的编程语言之一，易于编程的特点使其广受编程人员的喜爱。然而，执行速度并不是在开发中选择编程语言的唯一标准。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，所以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。

# 2.国内外研究现状及发展趋势

2.1 异构计算

**2.1.1 OpenCL**

每一次超级计算机体系结构的变化，都会伴随着并行程序设计环境和语言的进化。为了高效地为分布式存储和共享存储的并行计算机编写并行程序，工业界在总结众多各类并行语言的优点的基础上，分别提出了目前已经被广泛接受和使用的并行程序开发环境MPI和OpenMP工业界标准。这两个标准并行程序设计环境和语言的提出和推广普及，极大地提升了大规模并行应用软件的开发效率，降低了开发成本，确保了软件在不同厂家机器之间的可移植性和高性能。随着当前国际上主流体系架构逐渐向异构混合并行的转变，如何为这一新体系架构设计高效的且被广为接受的工业界标准并行程序开发环境和语言，成为一个迫切需要解决的问题。

开放计算语言[11]（Open Computing Language, OpenCL）正是在这一背景下由一个非盈利性组织技术联盟Khronos Group进行管理的异构编程框架，它得到了众多硬件和软件厂商的支持，成为了工业界异构计算的标准语言。OpenCL支持多种层次的并行[12][13][14]，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。

**2.1.2 CUDA**

CUDA 表示 Compute Unified Device Architecture（统一计算设备架构）[15][16][17]，是一种新型的硬件和软件架构，用于将 GPU 上作为数据并行计算设备在 GPU 上进行计算的发放和管理，而无需将其映射到图像 API。它可用于 GeForce 8 系列、Tesla 解决方案和一些 Quadro 解决方案。操作系统的多任务机制负责管理多个并发运行的 CUDA 和图像应用程序对 GPU的访问。

CUDA 软件堆栈由几层组成，硬件驱动器，应用编程接口(API)及其 runtime 库,还有两个更高层的通用数学库 CUFFT 和 CUBLAS，这两个库在单独的文档中介绍。硬件已经设计为支持轻量级驱动和 runtime 层，以达到高性能。

CUDA 提供了一般的 DRAM 内存寻址以实现更多的编程灵活性，分散和收集内存操作。从编程角度看，这转换为就像在 CPU 上一样在 DRAM 的任何位置读取和写入数据的能力。

CUDA 提供了具有非常快速的一般读写访问的并行数据高速缓存或芯片共享内存，线程可以使用它来互相共享数据。应用程序可以利用它来最小化对 DRAM 的过度提取和巡回，从而降低对 DRAM 内存带宽的依赖程度。

CUDA采用C语言作为编程语言提供大量的高性能计算指令开发 能力，使开发者能够在GPU的强大计算能力的基础上建立起一种效率 更高的密集数据计算解决方案。CUDA是业界的首款并行运算语言，而 且其非常普及化，目前有高达8千万的PC用户可以支持该语言。

CUDA和OpenCL中，前者是配备完整工具包、针对单一供应商(NVIDIA)的成熟的开发平台，后者是一个开放的标准。

虽然两者抱着相同的目标：通用并行计算。但是CUDA仅仅能够在NVIDIA的GPU硬件上运行，而OpenCL的目标是面向任何一种大规模并行处理器，期望能够对不同种类的硬件给出一个相同的编程模型。由于这一根本区别，二者在很多方面都存在不同：

1）开发者友好程度。CUDA在这方面显然受更多开发者青睐。原因在于其统一的开发套件(CUDA Toolkit, NVIDIA GPU Computing SDK以及NSight等等)、非常丰富的库(cuFFT, cuBLAS, cuSPARSE, cuRAND, NPP, Thrust)以及NVCC(NVIDIA的CUDA编译器)所具备的PTX(一种SSA中间表示，为不同的NVIDIA GPU设备提供一套统一的静态ISA)代码生成、离线编译等更成熟的编译器特性。相比之下，使用OpenCL进行开发，只有AMD对OpenCL的驱动相对成熟。

2）跨平台性和通用性。这一点上OpenCL占有很大优势（这也是很多国家实验室使用OpenCL进行科学计算的最主要原因）。OpenCL支持包括ATI,NVIDIA,Intel,ARM在内的多类处理器，并能支持运行在CPU的并行代码，同时还独有任务并行执行模式，能够更好的支持异构计算。这一点是仅仅支持数据级并行并仅能在NVIDIA众核处理器上运行的CUDA无法做到的。

3）市场占有率。作为一个开放标准，缺少背后公司的推动，OpenCL显然没有占据通用并行计算的主流市场。NVIDIA则凭借CUDA在科学计算、生物、金融等领域的推广牢牢把握着主流市场。

由于目前独立显卡市场的萎缩、新一代处理器架构(AMD的Graphics Core Next (GCN)、Intel的Sandy Bridge以及Ivy Bridge)以及新的SIMD编程模型(Intel的ISPC等)的出现，未来的通用并行计算市场会有很多不确定因素，CUDA和OpenCL都不是终点，期待未来会有更好的并行编程模型的出现(当然也包括CUDA和OpenCL，如果它们能够持续发展下去)。

**2.1.3 其他异构计算模型**

在论文[18]中，作者提出了一种基于编译指令的节点内编程模型OpenMC。该模型实现了对异构节点，特别是天河超级计算机中的节点的易编程、高性能、可移植性等特点。虽然现有的模型更多的是通过加速器来加速计算，OpenMC旨在使用一个计算节点中的多个CPU和加速器来均匀和充分利用其性能。OpenMC通过提供一个统一的硬件资源的抽象来达到这样的目标，并且促进了异步任务并行性的开发。

2.2 结构化并行编程

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[19]、DMP[20]、CoreDet[21]和Grace[22]，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[23]。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[24]，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[5]中的spawn/sync，Habanero Java[19]和X10[24]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[25]。J. Mellor-Crummey针对split-merge程序提出基于Vector Clock的数据竞争检测算法[26]。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[27]，Habanero Java研究小组针对async/finish提出SPD3算法[5]，良好嵌套的并行Ruby程序所设计的TARDIS算法[6]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[27]和Future[28]等

在论文[7]中，作者介绍了算法框架的首次提出[29][30]和详细的说明[31]，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。

2.3脚本语言并行编程

PyOpenCL是OpenCL的Python编程环境，是OpenCL在Python上的实现。在论文[32]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，其主要操作原则为人类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。

在论文[33]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[34]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[35]编译生成工具，Python Remote Object和PyCSP[36]。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。

在论文[37]中，作者描述了为了适当的扩展并行脚本语言，一个确定性如何被获取且动态地执行或验证。特别地，作者介绍了确定性并行Ruby（DPR）的结构和一个运行时系统（TARDIS）。该系统可以验证确定性所需要的属性，包括减少和交换操作符的正确用法，每个并行任务之间的相互独立性（没有数据竞争）等。

# 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(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.

[6(20)] W Ji, L Lu, M Scott. TARDIS: Task-level Access Race Detection by Intersecting Sets, WoDet’2013.

[7(14)] McCool, Michael D. Structured parallel programming with deterministic patterns[J]. Dr Dobbs Journal, 2010:5-5.

[8(8)]Martin G. Overview of the MPSoC design challenge[J]. 2006:274-279.

[9]Raman R, Zhao J, Sarkar V, et al. Scalable and precise dynamic datarace detection for structured parallelism[J]. Acm Sigplan Notices, 2012, 47(6):531-542.

[10] Feng M, Leiserson C E. Efficient Detection of Determinacy Races in Cilk Programs[J]. Theory of Computing Systems, 1999, 32(3):301-326.

[11] BenedictR.Gaster, 贾斯特, 张云泉. OpenCL异构计算[M]. 清华大学出版社, 2012.

[12]John E. Stone, David Gohara, Guochun Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems[J]. Computing in Science & Engineering, 2010, 12(3):66-72.

[13]Munshi A, Gaster B, Mattson T G, et al. OpenCL programming guide =[M]. 科学出版社, 2012.

[14]Du P, Weber R, Luszczek P, et al. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming [J]. Parallel Computing, 2012, 38(8):391-407.

[15]Nickolls J, Buck I, Garland M, et al. Scalable parallel programming with CUDA[C]// Acm Siggraph. 2008:40-53.

[16]Nicholas Wilt, 威尔特, 苏统华,等. CUDA专家手册:GPU编程权威指南[M]. 机械工业出版社, 2014.

[17]吴焰斌. cuda编程模型[J]. 科技风, 2009(3):65-66.

[18]廖湘科, 杨灿群, 唐滔,等. OpenMC: Towards Simplifying Programming for TianHe Supercomputers[J]. Journal of Computer Science and Technology, 2014, 29(3):532-546.

[19] Olszewski M, Ansel J, Amarasinghe S. Kendo: efficient deterministic multithreading in software[J]. Acm Sigplan Notices, 2009, 44(3):97-108.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] J Shirako, V Cave, J Zhao and V Sarkar. Finish Accumulators: a Deterministic Reduction Construct for Dynamic Task Parallelism, WoDet’2013.

[28]. Jun Shirako, Vincent Cave, Jisheng Zhao and Vivek Sarkar. Finish Accumulators: a Deterministic Reduction Construct for Dynamic Task Parallelism, WoDet’2013.

[29] Cole M I. Algorithmic skeletons: a structured approach to the management of parallel computation[M]. 1988.

[30] Cole M. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming[J]. Parallel Computing, 2004, 30(3):389-406..

[31] Skillicorn D B, Talia D. Models and languages for parallel computation[J]. Acm Computing Surveys, 1970, 30(2):123-169.

[32] 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.

[33] 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.

[34] 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.

[35] Parr T. The Definitive ANTLR Reference: Building Domain-Specific Languages[M]. Pragmatic Bookshelf, 2007.

[36] Mcewan A A, Schneider S, Ifill W, et al. PyCSP- Communicating Sequential Processes for Python[J]. Communicating Process Architectures, 2007, 65:229-248

[37]Lu L, Ji W, Scott M L. Dynamic enforcement of determinism in a parallel scripting language[J]. Acm Sigplan Notices, 2014, 49(6):519-529.

硕士研究生文献综述报告评阅表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | | | |  |
| 文献综述报告成绩 | | | | | | □ 优 □ 良 □ 通过 □ 未通过 | | | | | |
| 导师评阅意见 | | | | | | | | | | | |
| 廖心怡同学的文献综述报告对当前异构计算中的各个并行编程方法进行了总结与分析，并针对结构化并行编程方法进行了详细的阐述。文献综述给出了课题相关的最新进展和研究现状，说明了课题研究的重要意义，为后续研究工作的展开奠定了良好的基础。  签字：  年 月 日 | | | | | | | | | | | |