关于《Verifying Qthreads: Is Model Checking Viable for User Level Tasking Runtimes?》的阅读报告

一、相关背景

随着系统发展到极致，成本和技术限制决定了改进不是来自更快的处理器或指令级并行性，而是来自每个处理器增加内核和硬件线程的数量。这种从单核到许多小型并行内核的转变，使得利用这种并行性的有效编程模型的开发成为 HPC 系统研究的一大挑战。与顺序模型相比，并行编程模型更难推理——限制有能力或资源创建正确、可扩展和高效的大型并行程序的开发人员池。

因此，为了向最多的用户群提供并行系统，需要抽象或简化并行的编程模型。MPI 和 OpenMP 编程模型的成功，使共享内存和消息传递平行性通过标准界面提供给各种科学用户，证明了这种方法的可行性。

除了这两种模型之外，用户级别任务模型还允许程序员指定要执行的最小连续工作单元（任务）中的计算及其数据和控制依赖关系。这种自动分工将管理工作调度的责任推至运行时间系统，简化了用户的并行性视图。

目前，这些系统通常是以临时方式开发的，单位测试为正确性和行为保证提供了"足够好"的保证。当 Top500 类 HPC 系统探索具有轻松内存模型的架构时，不同运行时间组件之间的交互的复杂性使得临时测试站不住脚。正式验证和验证为这一问题提供了潜在的解决方案，是 HPC 研究的一个活跃主题。

HPC 运行时间正确性的理想方法是执行一个任务并行运行时间，该任务通过构建（即可证明实现数学上严格的规范）进行正确的任务。经过充分验证的运行时间实现可能会完全消除测试的需要——操作语义和细化证明既会指定任务平行运行时间，又可证明该规范的实施是正确的。然而，即使最近在证明工程方面取得了进展，但非平凡任务并行运行时间的操作语义和经过充分验证的实现，将是专家对一个人一年工作量顺序的重大努力。

另一方面，模型检查已经在 HPC 和其他科学应用中探索。模型检查运行时间可以提供具有完整代码覆盖的测试套件，并将提供一种“轻量级”方法，在可行的时间范围内验证系统。通过抽象实现详细信息并暴露运行时间的核心功能，生成的模型还可以为运行时间的操作语义提供基础，该语义可以用作样式中更实质

性的改进证明的一部分。

尽管这些模型比全面改进语义和证明更简单，但它们仍然是大量的开发工作。用户级别的任务运行时间很复杂，通常依赖于线程共享内存并发模型，而不是指定的邮件传递，因此由于同步与通信分离，因此在本质上更难推理。这种缺乏协调需要更多的计算努力和记忆来探索系统各组成部分之间可能存在的非决定性相互作用所产生的状态空间。

Qthreads 是桑迪亚国家实验室的跨平台、通用、平行运行时间。Qthreads 使用两个基本抽象实现任务并行运行时间，即安排在本地域上的轻量级线程和全/空位 （FEB） 同步原始。这些抽象的目标是模拟硬件线程架构，实现大规模轻量级多读和同步，如 Tera MTA / Cray XMT。单个计算线程通常是匿名的，具有单个资源分配，并明确开发本地。这种资源处理方法使 Qthread 与传统的线程模型有着根本的不同。Qthread 中的调度是合作的。当一个 Qthread 无法再取得进展时，无论是通过同步原始还是通过显式收益，则控制将传递给另一个等待的 Qthread。此上下文切换完全发生在用户空间内，通常是纳秒，而不是系统呼叫的微秒，因为 Qthreads 不需要保存信号处理器或全套系统寄存器。这种使用用户级别的上下文切换还允许 Qthreads 运行时间与数据访问交错计算。

作为工业中使用的相对成熟和强大的基于任务的模型，Qthreads 提供了一个有趣的验证目标。Qthreads 目前还面临着从验证中获益的挑战，特别是需要验证对旧架构的支持，并协助验证新的和未来的架构。

Qthreads 广泛的架构支持对于验证来说尤其成问题。Qthreads 支持 ARM32、PowerPC64、SparcV9 和 Tilera 平台，目前 Power9 和 ARMv8 架构的本地端口正在进行中。然而，由于每个架构的内存模型存在细微差异，因此难以维持对这些架构的支持。特别是，支持基于 TSO 的内存模型或更宽松的语义的性能权衡。将 Qthreads 端口移植到这些平台的尝试已导致以前隐藏的错误和性能回归暴露在最前沿，特别是 Sine 任务级别障碍实现。

二、研究问题

一个关键的未回答的研究问题是，如何验证一个现实模型，准确表示一个功能齐全的用户级任务运行时间的实现？

三、现有解决方案

1、在 HPC 验证中，检查传统消息传递运行时间的模型具有悠久的历史。MPI-SPIN 模型检查器允许对 MPI 程序、Promela 进行 MPI 应用程序的现场模型检查。

2、更普遍的是，CIVL 语言允许在 C 中编写 MPI 和 OpenMP 程序，并自动生成代码的模型检查版本。Modex 还允许用户从 C 程序生成 Promela 模型，并使用这些模型来理解 C 程序。

作者提到说，据他们所知，他们的方法是首次尝试对 HPC 用户级别的任务运行时间进行建模检查。

四、作者的核心思想与创新点

1、核心思想

为了探索模型检查用户级任务运行时间的可行性，作者团队编写了 Qthreads 用户级任务层的 Promela 模型，并在 SPIN 模型检查器中验证了该模型。主要工作分为以下四点：

a)用户级任务运行时间的第一个 Promela 模型。

b)描述实施用户级任务运行时间的现实模型所需的实施决策。

c)对该模型可扩展性进行实际比例的评估。

d)模型检查允许在 Qthreads 代码中发现预先存在的 bug 的实际示例。

2、创新点

与从现有代码库中衬入或生成模型的方法不同，作者团队的方法尝试提供从零开始的 Promela 模型，以用作一种机制，能够在一定的运行时间对新功能和内存模型进行规范和测试。

五、实验过程与评估

1、验证任务并行运行时间

历史上，像 SPIN 这样的模型检查器一直不适合这种探索，它们假设内存访问的顺序一致性，但是最近关于在 SPIN 中表示可选内存模型的研究开辟了使用 SPIN 模型探索可选内存顺序的可能性。

考虑到这些挑战，作者团队开发并评估了 SPIN 模型检查器中 Qthreads 运行时间核心的 Promela 模型。在 Promela 中模拟线程队列，以便以稳健的方式探索 Qthreads 的新功能和内存模型支持。

A. SPIN

SPIN 是一个"支持异步过程系统设计和验证的通用验证系统"。特别是，SPIN 提供了丰富的语义，用于使用共享内存或缓冲通道描述流程之间的相互作用。SPIN 可以使用原子语句和共享内存同步来模拟原子存储器订购指令。

SPIN 模型以模型描述语言 Promela 实施，并汇编成 C 验证器，这些验证器详尽地探索模型的所有状态，或者由 SPIN 本身进行解释。

B. 实施

传统上，用户级任务运行时被实现为在多个工作线程之上多路复用的协同程序。这些协同程序由用户级调度器安排，然后将每个协同程序映射到相应的过程或内核线程。然后，这些协同程序执行到完成或同步点，其中协同程序将处理器的控制权让给调度器。

虽然基于任务的运行时间共享此常见结构，但它们在记忆模型和并发原始器上有所不同。有些具有 CSP 风格的通道语义，或者有些具有更传统的共享内存，具有相互排斥的语义。

Qthreads 是作为具有明确同步的传统共享内存运行时间实现的。在基于 POSIX 的系统上，Qthreads 首先调用 qthread_initialize，每个都运行 qthread_master 功能。qthread_master 功能执行，然后根据其状态处理/重新安排线程。

1） 指针

由于 Promela 不提供指针表达式，因此在 Promela 中，指针建模是一个常见

且易于理解的问题。传统的方法是手动实现指针，将单个指针表示为外部定义数组的数组索引。

Qthreads 模型使用此方法实现其指针引用。Qthreads Promela 模型中的所有指针都被重新实现为 int 值（使用-1 表示空值），并指向目标数据结构的全局数组。

2） 函数

Promela 没有函数的概念。Promela 中的中心计算单元是 proctype，它要么在运行时用 active 关键字实例化，要么使用 run 命令从另一个 proctype 进程手动启动。

为了在 Promela 中实现一个抽象级别，Promela 提供了内联函数，允许用户生成有效的宏来封装代码块中的常见行为。内联语句的这种用法很复杂，因为这些内联语句的参数没有进行类型检查，只能将简单的值传递给内联定义。如果数组和 typedef 数据类型传递给内联语句并且在内联语句体中取消引用，则会触发编译时错误。

此外，内联语句没有控制流、传递值参数或返回值的概念。内联语句一直运行到完成，对内联语句的参数所做的任何修改都是破坏性的，并且会修改程序的全局状态。

为了在 Qthreads 中对 C 函数建模，作者团队向返回表示参数的值的函数添加了另一个参数。这种方法类似于 C 如何通过传递指针值来处理多个返回值。通过使用内联语句对函数进行建模，可以将 Qthreads 的 Promela 模型分解为与 Qthreads C 实现几乎相同的结构。

3） 函数指针

在模拟函数之后，仍然需要函数指针的概念。尽管 TLS 和指针在作者团队的模型中实现起来相对简单，但是函数指针（用于指定运行时实现和运行的实际任务）却不那么简单。

4） 同步

为了处理并发性，Qthreads 使用 Pthreads 实现中的各种函数。特别是，Qthreads 依赖于互斥量和条件变量来同步线程队列和公共数据结构。

Promela 不直接提供互斥或条件变量，但它们很容易用 Promela 的语义表示。为了对 Pthreads 样式的锁进行建模，使用共享的原子状态变量进行互斥，并使用通道表示条件变量信令。

5） 模型程序结构

一旦满足了这些实现细节，就能够将模型组合成一个（有点麻烦的）类似于原始 C 实现的结构。

Promela 中 C 特定语法的这种表示方式允许编写一个模型，在 C 函数与 Promela 内联线之间建立一对一的对应关系，这使得在很大程度上可以直接将原始 Qthreads 代码转到 Promela 模型。

2、评估

为了评估模型，作者团队使用两个标准，模型的可行性，例如，它的能力；探索其模型的状态在可牵引的时间和合理的规模以及有效性，例如，模型发现错误的能力。

评估是在桑迪亚国家实验室的伏尔特里诺高级建筑试验场集群的节点进行的。此节点承载 Intel(R)Haswell Xeon(R)CPU E5-2698 v3，每个核心有 32 个内核和 2 个超线程，价格为 2.30 GHz。此节点还包含 132GB 的 DDR4 主内存。

A. 可行性

任何核查项目的一个核心方面是它是否可行——它是否能在可处理的时间内产生有用的结果。

为了评估模型的可行性，作者团队测试了模型的可扩展性，以实际量的并发性，以及其资源使用在现代硬件上是否可控。

模型检查时遇到的经典可行性问题是状态爆炸。由于模型检查算法探索模型执行中的每一个可能状态转换，因此模型复杂性的任何增加都可能导致运行时间和状态搜索的指数级爆炸。Qthreads Promela 实现特别容易受到状态爆炸的影

响。简化的 Qthreads 模型目前为 1200 行 Promela 代码，比 SPIN 分布中的最大示例大 4 倍。

B. 有效性

在确定模型可行多达 16 个过程后，下一步是确定它是否为 Qthreads 调度器提供了任何有用的见解。

为了判断 Qthreads 模型的有效性，作者团队使用随机模拟（即运行模型的 SPIN 实例）运行模型，在 C 中编译 SPIN 验证器，并彻底查找与所需行为的反例。特别是，是否可以发现任何潜在的错误，在 Qthreads 模型中实现。

为了执行此验证，作者团队将几个 Qthreads 单元测试移植到模型中，并使用它们来评估 Qthreads 实施中以前潜伏的错误的可能性，从而有效地模拟检查单元测试。

采用这种方法的标准是，看看程序在功能上是否以与传统评估相当的方式运行（例如，在传统分布中运行进行检查），同时在作为泛实例进行详尽评估时，为 Qthreads 实现的正确性提供潜在的见解。

这种方法是成功的。模型成功地为多个调度器调度了一个 Qthread，为每个调度器安排了一个 helloworld 任务，并执行了正确的失效函数。然而，当在穷举验证模式下运行时，它捕捉到竞争条件，另一个没有工作的调度器可以从另一个 shepherd 的链表中“窃取”一个终止的进程，这打破了调度器提供的任务必须处于“New”、“Running”或“yield”状态的不变。

六、启发与感想

选择这篇文章来写读书报告一是因为网站上很多其他文章都完全看不懂，二是因为这篇文章标题是 Qthreads，而我们在课上已经学过了 Pthreads，没想到还有 Qthreads，所以让我眼前一亮。

QThreads 建立在 Pthreads 之上.它们提供了面向对象,使得使用线程更容易.除了 QThreads 是可移植的,它们可以在使用底层线程系统的任何系统上运行,而 Pthreads 是特定于 POSIX 系统的。

文章描述了使用 SPIN 模型检查器验证的 Promela 中 Qthreads 用户级任务运

行时间的初始模型。该模型允许快速验证和开发新的并发扩展到 Qthreads，并为探索当前和未来的放松记忆模型的 Qthreads 奠定了基础。它还可以形成 Qthread 或其他用户级任务运行时间的"深度规范"的基础。评估表明，模型检查既可在小尺度上进行，而且只需人工检查即可发现难以检测的潜在错误。

令我惊讶的是，我原来以为运行时间只是一个简单的小问题，而没想到这其中蕴含了这么多复杂的工作，需要建立模型才能准确计算得出。文章中也让我体会到了“封装”的重要性，就像 MPI 和 OpenMP 分别标准化了消息传递和共享内存并行编程模型一样，Qthreads Promela 模型还可以用来封装线程运行时的行为，继续改进，为任务分配模型的潜在标准化打下基础。文章也让我有一些疑问，比如面对“状态爆炸”应该如何有效抑制？为什么在穷举验证模式下运行时，模型能打破调度器提供的任务必须处于“New”、“Running”或“yield”状态的不变？

总而言之，对于刚刚接触并行计算的我来说，还有很多知识是要学习的。知识的海洋如此广阔，而我又是如此渺小。我必须像一块海绵一样不断汲取知识，才能不断收获成长。希望在未来我也能选上一些关于并行计算的课程，继续探索，继续成长！

七、原文链接

https://ieeexplore.ieee.org/document/8638407