The scalable commutativity rule:designing scalable software for multicore process

可扩展性的重大潜在机会在接口。可以在实现之前就对可扩展性做一些预测吗，只考虑接口的规范。本文介绍了下列规则：无论何时接口操作可交换，他们可被一种好的扩展性的方式实现。这使得开发者在设计软件时，考虑可扩展性时，从接口设计开始，一直到实现，测试和评估。 为帮助使用，开发了commuter.该工具支持高级接口模型，生成可交换的操作的测试。通过使用这些测试，该工具可以评估一个实现的可扩展性。并使用该工具测试了Linux和sv6的应用可扩展性。Sv6是一个新的研究的操作系统内核。

文章内容

当前评估多核软件的可扩展性的最好的方法：选择一个工作负载，测量描绘在不同核数时的性能。使用differential profiling的方法来确定可扩展性的瓶颈。但是不同的工作负载和核数会呈现出不同的瓶颈，不清楚哪个才是最主要的瓶颈。这一问题发现晚了，在开发后期就不太可能再去改动接口了。本文提出了新的方法，在软件接口确定应用的可扩展性的好坏。这可以使得应用开发者在硬件可用或实现前就可以对开发的程序进行扩展性的评估。可扩展性通常被认为是实现的属性，而不是接口的属性。然而，如果我们假设一个共享内存的多核处理器，通过一个MESI-like的协议带有cache.在这些处理器上，一个核可以扩展性地读和写数据，这些数据是其缓存在其自己核内。可以扩展性地读其缓存在共享模式下的数据。写一个缓存行，给行最后一次读写是被其他核完成的，是不可扩展的。

因此我们说一些操作是可扩展的。如果他们的实现有0冲突的内存访问--当一个核读写一个缓存行的时候，没有核对该行进行写。在内存访问0冲突的情况下，增加核数会线性提升应用的性能。

我们方法的核心是：可扩展交换规则---几个操作可以交换，使用接口无法确认他们的执行次序。这些操作的实现具备内存访问0冲突的属性。更准确的说法：无论何时接口可交换，他们可以以一种可扩展的方式实现。我们使用可交换性来判断软件的可扩展性。

正确性说明：操作的可交换性是直白的：操作的执行顺序对最终的结果没有影响。这些操作之间的通信是不必要的，消除这些通信能够产生无冲突的实现。 SIM commutaticvity:状态依赖的和基于接口的，单调的。当操作在一个特定的系统状态上下文，特定参数和特定的并发操作时是可交换的，可以证明存在一个0冲突的实现。 连接性规则使得可扩展软件设计有了更好的设计方法：首先分析接口的可交换性，然后再设计在可交换情景下的实现。 在此之前的方法：我们尝试决定这些实现是否是可扩展的，只能通过我们想到的所有的实现。这个过程很难，没有指导性，而且不能扩展到复杂的接口上。 作者将commuter应用到一个简化的POSIX系统调用模型。从这个模型，作者得到了13664个可交换的系统调用对。使用这些测试来指导新的OS--sv6.

相关工作

有相关的工作是只做内存访问的接口的；disjoint-access parallelism is specialized to the memory system

interface。本文的工作进行了接口的扩展。有一些后续的工作探讨了一些拥有不独立的进程的可扩展性。本文发现了在实际的硬件上，一个单独的修改的共享缓存行可以击败可扩展性。 The laws of order探索了为了正确执行，接口的强非交换性和其实现是否需要原子指令或hence之间的关系。秩序定律类似于交换性规则，但得出关于顺序性能的结论，而不是可伸缩性。缓存行的争用可能导致较大的可扩展性的下降。本文的可交换规则建立在MCS lock的基础上。

设计可扩展OS：行业参与者通常按照迭代开发来提高软件的可扩展性。设计，实现，测量和重复。Absent a method for reasoning about interface-level scalability, it is unclear which of the bottlenecks are inherent to its system call interface 还有的是针对多核处理器的多内核。

交换性：可交换可以增加系统的并发性。作者的工作证明了可交换的操作总是有无冲突的实现的。相比于之前的工作，本文的工作集中关注可扩展性，给定并行，可交换的操作，证明这些操作有可扩展的实现。 数据库社区一直使用逻辑readset，writeset,冲突和执行历史来推理交易是如何被插入的，同时保证序列化。

测试用例生成：concolic测试和符号执行方面的研究工作通过符号执行特定的实现来完成测试用例生成。我们的commuter工具使用符号和同调执行的组合，但是基于该实现接口的模型生成任意实现的测试用例。commuter的测试用例生成的目标是实现冲突覆盖，使用符号地址或索引来测试不同的访问模式。

可扩展交换规则

什么是可扩展交换规则？为什么这一规则是正确的？采用了形式化的证明，这一形式化的证明基于抽象的行动，历史和实现。 动作：将系统执行建模为一系列的动作，一个动作是调用或响应。在OS中，调用代表着带有参数的系统调用，响应代表着对应的返回结果。一个动作由以下组成：操作类型；操作参数或返回值；相关线程；独特性标记。一次系统执行被称作是历史。我们只考虑形成较好的历史，每个线程的动作形成了一系列的调用和响应对。 交换性：交换性的核心：动作的顺序没有关系，不会影响最终的结果。规范对此作出了明确的规定：一整套操作是可交换的，当规范与操作集合的执行顺序是相同的。这意味着对于交换集的一个次序有效的任何响应对于交换集的任意阶都是有效的，并且同样地，对于一个命令无效的任何响应对于任何顺序都无效。状态依赖性意味着SIM交换性捕捉到了可以交换的动作，因此可以扩展。SIM交换性同时也是基于接口的，其只使用规范评价执行熟悉怒的结果，这并不意味着每个重排在给定的实现上有相同的结果，需要每个重排是允许的，因为任何实现都可能有不必要的扩展性瓶颈。开发人员可以在软件设计阶段，接口设计阶段就使用可交换规则。 实现：为了探究实现可扩展性，我们需要建模实现足够多的细节来区分是否不同的线程的内存访问是无冲突的。我们把一个实现作为步骤函数：给定一个状态和调用，其产生新的状态和响应。特殊的continue操作使能并发的overlap操作和阻塞。定义了3个集合：S：实现状态的集合；I：有效调用的集合；R:有效响应的集合。给定一个老的状态和调用，实现会产生新的状态和响应（响应和调用是相同的线程）。一个continue响应表明了对于该线程的一个真的响应还没有准备好，允许实现有效地切换到另外一个线程。continue调用给了实现一个完成outstanding请求的机会。当对实现进行调用时，实现会产生历史。当实现产生的响应总是被规范允许的，实现m是正确的。注意，一个正确的实现不需要能够产生所有的可能正确的响应。只是其产生的每个响应是有效的。为了判断是否没有冲突，我们必须查看实现状态，确认读和写操作，检查访问冲突。2个实现步骤有访问冲突，当他们处于不同的线程，一个在写状态，另外一个是读或者写状态。一组实现步骤是没有冲突的当且仅当这个集合中的step pairs没有访问冲突。

规则：有了上面的定义之后，就可以进行形式化的工作了。在文中的3.5的部分集中进行了形式化的证明工作。交换性规则和证明构建将状态和历史依赖性推到了极致。证明构建是专门为单个可交换区域定制化的。

构建的重复应用可以构建一个实现，这个实现可以在一个历史中的多个可交换区域上进行扩展。交换性规则证明了SIM-可交换的区域是没有冲突的实现。可交换性对于无冲突访问的判断是充分的，但不是必要的。一些不可交换的接口也有可扩展的实现。

设计可交换的接口：规则使得接口和规范层面的扩展性推理变得容易，SIM连通性使得我们可以将规则用在复杂的接口上。主要的方法有：分解复合的操作，拥抱规范的不确定性；允许弱的顺序性；异步释放资源。

commuter:完全理解一个复杂接口的可交换性是很难的，当操作可交换时，实现一个避免共享的实现又增加了难度。通过利用可交换规则的可交换性，开发者可以自动化这一推理过程。作者开发了commuter的工具，分为3个组件:分析器，测试生成器，测试用例运行器(MTRACE) 首先：分析器将接口的符号化的模型作为输入，计算精确的条件，这些条件是接口操作可交换的条件。接着测试生成器根据这些条件生成具体的测试案例，这样的话，根据根据可交换规则应该会有一个无冲突的实现。最后，Mtrace对每个测试用例检查一个特定实现是否是无冲突的。开发者可以使用这些测试用例来理解他们应该考虑的可交换情况，迭代地找到代码中的可扩展性问题

文章中的第6部分探讨了：发现扩展性机会的问题。为了理解commuter是否对内核开发者有用，作者建模了几个POSIX文件系统以及虚拟内存调用，使用这些来评估Linux的可扩展性从而开发可扩展的文件和虚拟内存系统。对commuter进行了一次实际使用。 文章的实验性能评估部分主要回答了以下几个问题:不可交换的操作在真实的硬件上是否会限制性能；可交换操作的无冲突实现在真实的硬件上的可扩展性怎么样？对于扩展性的提升是否影响了sequential performance.

可扩展的交换性规则为软件开发者提供了新的方法来从软件接口层面理解和利用多核可扩展性。我们定义了SIM可交换性，允许开发者将规则用于有状态，复杂的接口上。作者还开发了commuter工具，帮助开发者分析程序的扩展性。最后，作者使用了SV6证明了实现一个广义上可扩展的POSIX实现是实际的。可交换性对于在硬件上实现可扩展性和性能是必要的。

结论

可扩展交换规则对于软件开发者来说提供了一个新的方法，从软件接口的角度来理解和利用多核可扩展性。定义了SIM交换律，允许开发者将该规则用到复杂有状态的接口。提出了工具commuter。