高带宽远程内存结构中的预取研究

许建卫，陈明宇，包云岗
（中科院计算技术研究所国家智能计算机研究中心，北京，100080）
（xjw,cmy,byg@ncic.ac.cn）

摘要

高速电路和光互联技术的发展极大地提高了网络的速度与带宽。因而，突破高性能计算机 CPU 与内存紧耦合的传统结构成为可能，CPU 与内存的耦合不再受距离的限制，这必将引起体系结构的变革。文献[1]提出 DSAG 结构——CPU 与内存在空间上分离，每个 CPU 节点上仅留少量内存，将海量内存放在远程统一管理作为内存服务器，CPU 节点和内存服务器之间通过高速网络互连。这种新的体系结构带来了更好的共享性和可扩展性，但同时也对我们解决 CPU 和内存之间的不平衡性问题带来了挑战。

为了降低 DSAG 这种远程内存结构增加的访存时延，我们考虑到 CPU 正常访存没有充分利用网络的高带宽，因此可以利用剩余的网络带宽来进行远程内存数据的预取。本论文在应用程序执行时记录本地（相对于远程内存）不命中的地址信息，以页对齐分析其中存在的页框流*（Page Frame Stream）的统计特征，并提出可基于页框流的预取机制可降低访存延迟、提升系统性能的观点。最后我们采用模拟的方法验证了此观点的可行性与正确性，进一步提出了三种预取策略，比较并分析影响预取效果的因素。

The Study on Prefetching of Remote Memory Architecture

Xu Jian-wei, CHEN Ming-yu, Bao Yun-gang
(National Research Center for Intelligent Computing Systems, Institute of Computing Technology,Chinese Academy of Sciences, Beijing 100080)

Abstract

High speed electrical and optical interconnection technique brings us high-speed and high-bandwidth network. Thus, we can break through the traditional computer architecture by decoupling memory from CPU. Distance between CPU and memory is no longer restricted, and this will consequentially cause innovation in high performance computer architecture. In paper[1] the authors present DSAG architecture—each CPU node is only attached with a small quantity of memory, while massive memory served as memory server is located away, and they are connected by high-speed network. This architecture provides better shareability and more scalability, but it also challenges us to reduce the gap between processor and memory.

To reduce the delay of remote memory access, with abundant network bandwidth, we can use the spare network bandwidth while CPU runs to prefetch data from the remote memory. In this paper, we record and analyze the address missed in local memory access while program runs, and analyze the statistical characteristic of the page frame stream. We propose a prefetching approach based on page frame stream to reduce remote memory access delay and improve the system performance. Finally, we use simulation technique to verify the feasibility and correctness of the prefetching approach, and propose three prefetching policies as well as the factors that affect the prefetching.

* 基址间隔呈等差分布的一系列物理页框（本文中流以页框为单位，访存地址是页对齐的）

1．引言

应用程序对内存的需求差别很大。在目前 CPU 与内存紧耦合的高性能计算机上，对于需求内存很少的应用，大量内存被闲置，造成内存资源的浪费；而对于内存需求很大的应用，可使用的内存又显得极为匮乏，不得不借助磁盘交换来弥补内存的不足，但这又牺牲了程序的性能。为了解决这个问题，人们提出了 NUMA 结构的远程内存体系结构[11]来实现内存共享，但受限于 CPU 和内存的紧耦合状况和有限的网络带宽，实现开销较大。

同时，高速电路和光互联技术正带给我们一个高速度高带宽的网络，文献[1]基于此提出了新的远程内存服务器体系结构,它实现了内存的按需分配，而且更方便 CPU 节点间共享内存。相比于传统体系结构，它还有如下优点：

- 高可扩展性：内存部件可以不受 CPU 硬件的限制而动态地增加或拆减。
- 高可靠性：内存节点的失效不影响 CPU 节点。
- 智能性：一些内存间操作（如内存间数据拷贝）可以在内存服务器完成[8]。

但 CPU 与内存速度的不匹配已是现代计算机的性能瓶颈，而远程内存结构则进一步加大了这一不平衡性，大大影响了系统的整体性能。因此，我们考虑采用预取机制来弥补新体系结构的不足，提升系统的整体性能。

许多预取机制已经被提出以解决 CPU 和本地内存之间的速度不匹配问题[12]。但是受到处理器结构和带宽的限制，以前的预取机制主要集中在 cache 一级：预取的数据量很小；预取单位从一个字[3]到一个 cache 块[4][9]；预取到的数据被放在 cache 中或是 cache 相关的缓冲区中 [4][9]。与他们不同，我们利用高带宽网络来对远程内存进行数据预取。在这种情况下，传统结构下的处理器内存紧耦合限制、带宽限制都不存在了，所以我们可以预取较多的数据，预取的单位也可以变大——在我们的研究中预取的单位即为页框。

CACHE 和本地内存已经充分挖掘了访存地址的时间和空间局部性，在此基础上，我们进一步研究远程访存页框基址的规则性。图一是快速排序算法在输入数据集大小为 3000 情况下的远程访存页框基址分布，从中我们可以看到页框基址分布具有很好的规则性，而这正是我们基于页框流进行预取研究的基础。

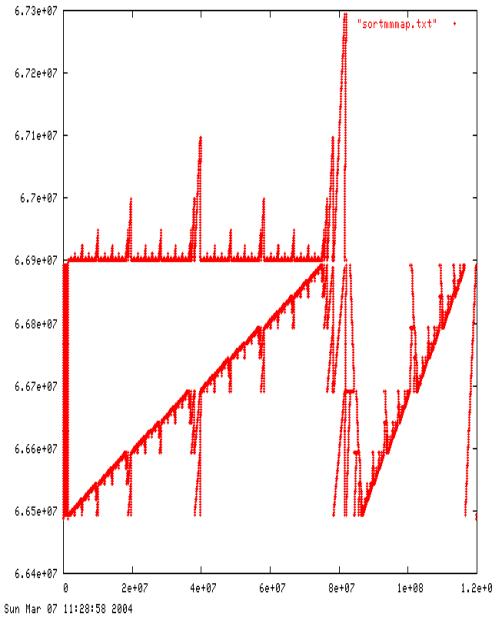
本论文的其余章节组织如下：第二节描述了实验背景，包括研究分析使用的结构模型以及几个假设。第三节中我们给出了页框流的统计特征，分析了预取的可行性，并介绍了页框流的检测和预取策略。在第四节我们给出模拟数据，验证分析采用预取机制带来的性能提升。第五节和第六节介绍了为相关工作和我们的下一步工作，最后一节给出总结。

2．实验背景

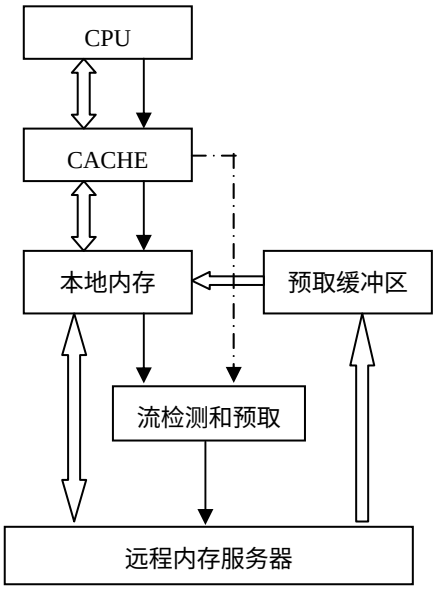
2.1 结构模型

文[1]中提出了一种基于光互连的新型计算机体系结构。它打破传统计算机中 CPU 和内存紧耦合的布局，CPU、内存、硬盘等部件被分离，相同类别的部件组织在一起，形成一个对外提供特定功能的“超级部件”，各“超级部件”间通过光纤网络互联。各 CPU 节点仅带有少量的内存，它用来做远程内存的高速缓冲区，海量的内存作为内存服务器放置在远端。内存服务器具有智能性，它自主实现内存资源的增加、删除、分配、回收等管理功能，对外则提供统一的功能接口。远程内存服务器不但带来了高共享性，而且由于 CPU 和内存间的

依赖性大为降低，整个系统的可扩展性、可靠性和兼容性也都大为提高。



图一 快速排序算法页框基址分布



图二 结构模型

在这种结构中，如图二所示，我们在本地和远程内存间增加页框流检测预取部件和预取缓冲区。页框流检测预取部件记录本地内存不命中的页框基址信息，检测本次不命中页框基址是否存在于已检测出的页框流中，如果是则直接从预取缓冲区取出页框数据到本地内存，同时发出远程内存访问请求预取页框流的后续页框数据，并用新取来的页框数据更新预取缓冲区。否则从远程内存取回页框数据放入本地内存。

2.2 几个假设

在调研了当前技术发展的趋势、并分析应用程序访存特性的基础上，我们提出了以下几个假设作为我们研究与实验的前提：

- 本地节点中仅有 CACHE 和少量的内存，但远程内存服务器的容量无限。
- 网络带宽无限。
- 传输延时主要由本地节点和远程内存服务器之间的距离决定，我们假定它不超过 300 米。
- 页框流的检测和预取由特殊的硬件或协处理器来实现，处理开销接近于零。
- 检测和预取算法的复杂性不考虑。
- 仅用一个小的预取缓冲区就可以容纳所有流中预取的数据。

前五个假设我们认为可以在不久的将来实现，最后一个假设我们将在在后面章节给出实验结果以进一步验证。

3．页框流的检测和预取

3.1 页框流的检测

我们以页框为单位分析应用程序的访存行为，并定义：

页框流：程序运行中访存出现的基址间隔呈等差分布的一系列物理页框。

我们的页框流检测算法源于文[2]中的流检测算法，但我们在三个方面进行了修改以满足我们的需要：

- 原算法所分析的流地址是程序运行中访存指令的所访问的所有地址，我们仅对本地不命中地址分析，分析的地址是页框对齐的，是每个页框的基址。
- 原算法中分析所有的访存地址信息，我们仅仅分析本地内存不命中的页框基址信息，这缩减了地址信息的存储开销和页框流检测算法需要分析的数据量。
- 我们将“伪流元素”剔除出去。在原始的算法中，流的检测仅仅由地址步长（流中任意两个相邻元素之间的地址差）决定，但时钟步长（流中任意两个相邻元素被访问的时间差，以内部时钟为单位）被忽略，我们则把时钟步长考虑进去。对于距页框流中上一个元素的基址差等于页框流的地址步长的元素，如果它距上一个元素的时钟步长超过某个值，我们称它为“伪页框流元素”，认为它不属于这个页框流。在我们的实验中，我们把这个值定为 5 倍平均时钟步长（页框流的平均时钟步长在程序运行过程中动态调整）。

3.2 页框流的规则性研究

为了考察基于页框流的预取机制的可行性，我们主要对页框流的以下四方面特征进行统计分析：

- 平均时钟步长 $\overline{\lambda_i}$：我们定义 t_{in} 为页框流 i 中第 n 个元素被访问的时间，它与页框流 i 中上一个被访问元素（第 n-1 个元素）的时钟间隔为 $T_{in}=t_{in}-t_{in-1}$。则流 i 的平均时钟步长定义为：

$$\overline{\lambda_i}=\frac{1}{N-1}\sum_{n=2}^{N}T_{in}$$

其中 N 为页框流 i 中的元素个数（下同）。$\overline{\lambda_i}$ 给出了页框流中任意两个相邻元素之间的时钟间隔的平均值，平均时钟步长与远程访存延迟决定了页框数据能否及时预取到预取缓冲区。

- 时钟步长的波动系数 $\overline{\gamma_i}$：我们定义页框流 i 的时钟步长波动系数为：

$$\gamma_i=\frac{\frac{1}{N-1}\sqrt{\sum_{n=2}^{N}\left(T_{in}-\overline{\lambda_i}\right)^2}}{\overline{\lambda_i}}$$

它反映了页框流时钟步长的波动情况，给出了预取发出时间的准确性。页框流的时钟步长与指令步长（页框流中连续两个元素间隔的指令条数）不是严格的正比关系，在一个循环中，同一访存指令对同一数组元素的访问的指令步长是相同的，但由于 CACHE 或 TLB 不命中等因素，页框流中的该数组元素的时钟步长可能发生变化。

- 同时活跃的页框流个数β：它影响预取缓冲区的大小。如果β很大，则需要很大的

缓冲区来容纳所有预取到的页框数据，否则预取到的页框数据可能在使用之前就已经被替换出缓冲区。

- 页框流的覆盖率φ：我们定义一个应用中元素个数为 m 的页框流的个数α_m，则元素个数为 k 的页框流的覆盖率φ定义为：

$$\varphi_k = \frac{k \times \alpha_k}{\sum_{m=3}^{\infty} m \times \alpha_m}$$

它反映了预取的有效性。页框流预取的有效性是指一个页框流误取元素个数与预取总个数的关系。对于每一个页框流而言，至少有一个误预取的数据（最后一个）。因此，如果页框流覆盖率分布在较小的区域，那么大部分页框流的元素个数都很少，预取将带来较多的浪费，降低预取的有效性。

3.3 页框流的检测和预取流程

页框流检测预取部件维护一个页框流表和记录池。页框流表中记录已经检测出的页框流的信息。记录池中记录尚未形成页框流的本地内存不命中页框基址信息，按先进先出的替换策略进行替换。具体的检测和分析页框流程如下：

a).得到本地内存不命中的页框基址信息。

b).根据该页框基址查页框流表，如果在某个页框流中，则执行 e)，否则执行 c)。

c).将数据从远程内存取到本地内存。

将当前不命中的页框基址信息放入记录池中。

对记录池中的所有地址信息进行分析，看能否形成一个新的页框流。如果不能，则执行 a)，否则执行 d)。

d).将新形成的页框流中的基址信息从记录池中删除。

把新形成的页框流加入到页框流表中，并采用预取策略来从远程内存中预取页框流中的数据到预取缓冲区中。执行 a)。

e).从预取缓冲区中将数据移入本地内存。

从远程内存中预取后续的页框数据到预取缓冲区中。执行 a)。

3.4 页框流预取策略

在我们的实验中，一个页框流被定义为程序运行中访存出现的基址间隔呈等差分布的一系列物理页框。一个页框流中数据元素个数的下限为 3，上限无穷大。当一个页框流中的前三个数据元素出现时，根据页框流的地址步长和时钟步长，开始预取页框流中的后续元素（页框数据）到预取缓冲区中。

假定存在一种流预取策略，它可以将所有预取的数据及时送入预取缓冲区中，也就是说所有预取的数据在它们实际被使用时都已经被预取到了预取缓冲区中。实际中，这种情况仅发生在页框流中的数据元素的时钟步长大于本地内存和远程内存间的数据传输时延，并且预取缓冲区中的数据在使用前没有被替换掉。它给出了基于页框流预取机制所能带来的性能提高的上限，我们称它为理想预取策略。

考虑到页框流的时钟步长可能小于本地内存和远程内存间的数据传输时延，预取的页框数据不能在使用之前到达预取缓冲区。因此预取命令发出的时间和每次预取的页框数量将影响预取策略的性能。根据页框流被检测到时首次预取的页框个数的多少，我们将实际使用的

预取策略分为贪心预取策略和保守预取策略两种。

贪心预取策略是指当一个页框流形成时，它首次预取一个或多个的页框数据到预取缓冲区中，预取的页框数量由页框流的时钟步长和本地远程内存间数据的传输时延这两个因素共同决定。贪心预取策略第一次预取足够多的页框数据到预取缓冲区中，使得首次预取的最后一个页框在使用之前已经到达预取缓冲区。之后每次预取命中（预取的页框数据被使用）都顺次预取页框流中的下一个元素替换预取缓冲区。

保守预取策略与贪心预取策略相似，区别在于当一个流形成时，保守预取策略第一次只预取一个页框数据到预取缓冲区中。

贪心预取策略第一次预取足够多的页框数据保证以后预取的页框数据能在使用时已经到达预取缓冲区。这种策略极大隐藏本地和远程内存之间的数据传输时延，但是对于每一个页框流而言，预取缓冲区中有多个预取的页框数据。因此当一个页框流中止时，预取缓冲区中误预取的元素（预取但未使用）也较多，这降低了预取缓冲区的使用效率和预取机制的准确性。

与贪心预取策略不同，保守预取策略首次只预取一个页框数据。当一个页框流中止时，预取缓冲区中仅有一个误预取的页框数据（最后预取的页框数据）。保守预取策略有效的利用了预取缓冲区而且预取准确性较高。但是当页框流的时钟步长小于本地和远程内存间的数据传输时延时，将会出现预取的页框数据在需要使用时仍未达预取缓冲区的情况。在这种情况下，保守预取策略比贪心预取策略带来更多的 CPU 等待时间。

对于一个应用来说，如果大部分页框流的时钟步长大于或等于本地和远程内存间的数据传输时延，理想预取，贪心预取和保守预取对系统性能提升相近；否则，由于后两种策略会出现需要使用时页框数据尚未到达预取缓冲区的情况，性能也将低于理想预取策略。

如果大部分页框流的时钟步长远小于本地和远程内存间的数据传输时延，或者大部分页框流的元素个数较多，贪心预取策略将比保守预取策略隐藏更多的传输延时时间，同时误预取的数据所占的比例也相对较小，在这种情况下贪心预取策略优于保守预取策略；否则结果相反。

我们的实验中对三种预取策略分别进行测试，4.3 节给出测试结果。

4．测试和结果

4.1 测试环境

我们的工作基于 Sand 模拟器和 Sand 操作系统，它们是我们在 VMIPS 模拟器[13]和 Topsy[14]操作系统的基础上实现的。Sand 模拟器是一个全系统的时钟级别的 MIPS 结构模拟器，它具有系统结构参数可调的特点。Sand 操作系统是运行在 Sand 模拟器上的小型操作系统，LINUX/MIPS 结构的可执行程序不加修改就可以在其上运行。

我们根据图二所示的结构修改 Sand 模拟器。不同于传统的本地内存，我们的结构中本地内存作为 CACHE 来使用，并且 CACHE 行很大。我们用 LMbench[15]来测试我们实际使用的计算机中 CACHE 和内存的延时参数，作为我们模拟器的时延输入参数。与内存相关的部件的配置如下：

- 一级 CACHE：16KB（I），16KB（D），每个 CACHE 行 32 字节，4 路组相联，随机替换。存取时延为 3ns。
- 二级 CACHE：256KB（U），每个 CACHE 行 64 字节，8 路组相联，随机替换。存

取时延为 7ns。

- 本地内存：1MB，共 256 项，每项大小为页框大小（4KB），全相联，LRU 替换策略。存取时延为 105ns。

我们模拟的计算机主频为 1GHz。

在我们的实验中，我们选择 LINPACK 和 SPEC2000 测试程序作为我们的测试应用。LINPACK 输入数据集的大小为 3000，SPEC2000 的输入测试集为 test 数据集。因为 SPEC2000 整数测试集中不命中的数据量较少，我们选取不命中数据最多的 MCF 为代表。根据每一个测试应用实际用到的物理内存的大小，我们选择几个有代表性的 SPEC2000 浮点测试程序：ART，EQUAKE，SIXTRACK，SWIM，APSI。

目前我们的系统仅模拟一个节点。考虑到网络带宽足够宽，所以如果 CPU 节点数在一定的规模内，增加 CPU 的数目不会引起网络的拥塞而导致性能下降，因此我们的单节点模拟结果适应于一定数目的多节点情况。文[10]的研究发现，大部分程序的访存模式随着输入数据集的大小保持恒定或随输入数据集的大小规则地变化，因此我们实验中从小输入数据集得出的实验结果对于大输入数据集也有参考意义。

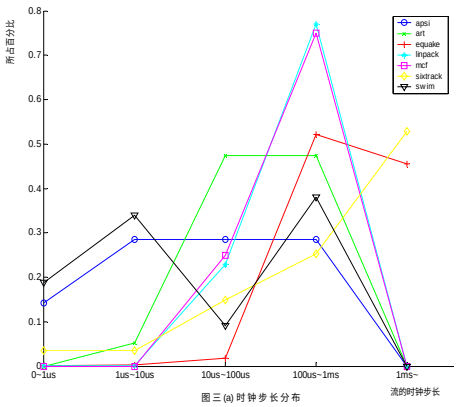
4.2 页框流特征分析

图三（a）给出了各测试应用的平均时钟步长分布，从中可以看出 APSI，SWIM 测试应用中页框流的平均时钟步长相对较小，主要分布在 1us~10us，其余五个测试应用中页框流的平均时钟步长基本都大于 10us，这为从远程内存中预取页框数据提供了足够的时间。

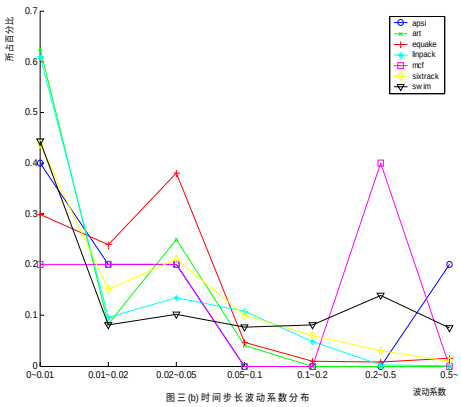
图三（b）给出了时钟步长波动系数分布，可以看出除了 MCF，其余均低于 5%。因此，我们可以基于时钟步长估计预取命令最恰当的发送时间，避免由于过早发送而导致预取页框数据在使用前就被替换出缓冲区，或者由于过晚发送而导致需要使用该页框数据时却还没有返回等情况。

图三（c）给出了同时活跃的页框流的个数分布，所有的都低于 20 个，因此使用 LRU 替换策略，我们可以用一个小的预取缓冲区来容纳所有当前活跃的页框流中预取到的页框数据，这验证了 2.2 节中的最后一个假设。

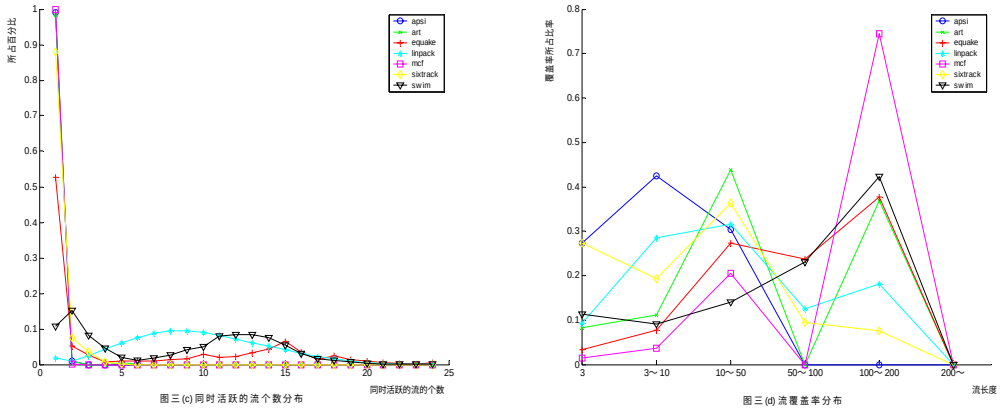
图三（d）显示出除了 APSI 和 LINPACK 的页框流覆盖率集中分布在 10 到 50 之间外，其他几个应用程序在 100 到 200 这两个区间页框流的覆盖率较大，所以误预取的数据所占的比例不高。



图三 (a) 时钟步长分布



图三 (b) 时间步长波动系数分布



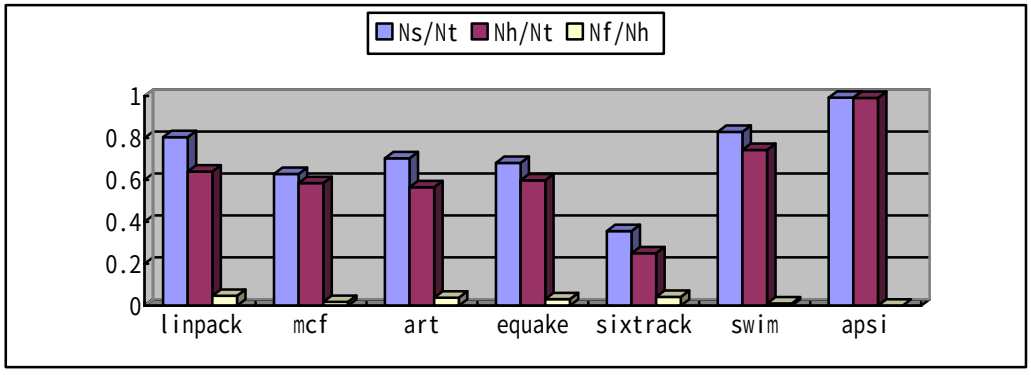
图三 页框流的特征

4.3 预取结果

在 2.2 节的假设下，当本地节点和远程内存服务器间的距离不多于 300 米时，考虑光互联网络的数据传输速度为 3×10^8 米/秒，单向数据传输时间近似为 1us。考虑到每次数据传输需要发送请求和送回数据的双向过程，所以总的数据传输时间近似为 2us。再加上初始化开销和发送接收时延，我们估算一次远程数据存取需 4us。

为了研究测试应用中不命中页框基址的规则性和预取的效率，我们主要对四个方面进行分析：本地内存不命中的页框总数 N_t（包括第一次引用不命中），可以被页框流捕捉到的页框总数 N_s，预取命中的页框总数 N_h 和贪心预取策略误预取的页框总数 N_f。

图四显示了上述数据的比例关系，从中可以看出，除 SIXTRACK 外，其余测试应用的 N_s/N_t 都大于 60%，这说明 60%以上不命中的页框都在页框流中。尤其是在 LINPACK，SWIM，APSI 中，这一比例接近或超过 80%。并且这些测试应用中的本地内存不命中页框基址信息具有很好的规则性，预取的可行性可以得到证明。



图四 流元素和预取命中、失误元素的覆盖率

在我们的实验中，当检测到页框流时才开始预取，因此除了页框流的前三个元素，后面的元素都将被预取到。从图四中我们看到，所有测试应用中 N_h/N_t 都大于 25%，而 N_f/N_h 都小于 5%，即 25%以上的本地内存不命中的页框都能被预取，所有预取的页框中误预取率在 5%以下。在 APSI 应用中，几乎所有的数据都可以被预取。

为了进一步研究远程内存结构的性能和预取带来的性能提高，我们模拟测试了不同情况

下各测试应用的执行时间，表一显示了采用不同的预取策略的程序执行时间比。第一行给出了每个应用实际使用的物理内存的大小，下面几行分别给出了不同情况下远程内存结构相比于传统内存结构下的执行时间比。

从表一的第二行可以看出，仅用 1M 的本地内存，在无任何预取措施的情况下，ART，MCF，SIXTRACK 和 APSI 的性能基本不受影响，只有 SWIM 的性能急剧下降。这说明对于数据密集型的应用，本地内存不命中的比例占的不多，远程内存结构系统性能很好。而对于在远程内存结构下性能明显降低的应用，则进一步采取预取措施以提升性能。

表一后三行给出了采用不同预取策略各应用性能提高的比较。从中可以看出对于在远程内存结构下性能下降的应用而言，预取可以有效的提高性能，验证了我们的分析。比如采用理想预取策略，对于 LINPACK 和 SWIM 性能提高分别为 13.7%和 39.9%。

	linpack	mcf	art	equake	sixtrack	swim	apsi
实际使用的内存大小	24M	1.4M	2.1M	10.4M	27M	59M	200M
无预取	1.273	1.001	1	1.038	1.002	2.159	1.006
理想预取策略	1.098	1	1	1.015	1.001	1.298	1
贪心预取策略	1.098	1	1	1.015	1.001	1.309	1
保守预取策略	1.098	1	1	1.015	1.001	1.848	1

表一 预取结果

从表一可知，除了 SWIM 之外，其余应用的性能提升在不同的预取策略下区别不大。而对于 SWIM 是个特例，原因在于——如图三(a)所示，SWIM 中的时钟步长主要分布在 4us 以下。根据 3.3 的分析，页框流的时钟步长小于本地和远程内存间的数据传输时延，则三种预取策略性能差别很大。对于其余的应用，时钟步长大于远程存取所需的数据传输时间，所以三种预取策略性能提升相近。

5．相关工作

为了实现不同节点间的物理内存共享，在传统的体系结构下一般采用软件实现的 DSM 技术[3][11]。它将物理上独立分布的内存节点统一组织起来形成一个大的虚拟地址空间，这个虚拟地址空间由所有的处理器节点共享，整体上是一种 NUMA 的结构。它的缺点是这个统一的虚拟地址空间是完全由软件来实现的，因此实现的开销较大；同时受限于低带宽的总线传输，传输的数据量较小，通信开销较大。而在我们的结构中，所有内存物理上被平等的共享，实现开销较小。在高带宽的网络下，一次可以传输大量的数据块（如我们实验中的页框）而没有明显增加通信开销。

为了减小通信开销，在 EM－X 体系结构中，使用多线程技术来在不同的处理器节点间传递小的数据包，由于采用多线程技术，通信和计算可以同时进行，因此可以隐藏远程内存访问带来的传输时延。但它仍没有改变传统的处理器和内存的紧耦合状况，因此还是受到低通信带宽的限制，而且软件实现的开销仍然很大。

文[3]中提出了一种硬件预取机制，其实现主要使用了两个数据结构：引用预测表(RPT)和前向 PC（LA-PC）。引用预测表中记录了访存指令的 PC 值、上一次访问的内存地址、地址步长、稳定状态等信息，前向 PC 是一个虚拟 PC，比 PC 超前运行。当前向 PC 和 RPT 表中的 PC 值匹配时，如果相应项的稳定信息表明它处于稳定状态，则对下一个数据进行预取。

RPT 表中的信息是基于同一条指令的，因此它不能在全局范围内分析访存模式的规则性。在我们的实验中我们对所有的不命中页框基址信息进行分析，相对于上述针对单条访存指令记录的地址信息分析范围增大，因此预取的机率增大。如果采用专门的硬件支持，我们实现中增多的分析不会带来明显的开销。

文[4]中使用一个先进先出缓冲区来存放预取的 CACHE 行。当缓冲区首部的 CACHE 行被访问时，它被移入到 CACHE 中，缓冲区中后面的数据前移，并从内存中预取新的 CACHE 行放到缓冲区队列尾部。文[9]中使用流缓冲区来替代二级 CACHE，它使用 LRU 替换策略。当一级 CACHE 不命中时，K 个相关的 CACHE 行被取到二级 CACHE 中。上述两种方法都仅利用了访存模式中的空间局部性，而没有访存模式中存在的规则性。当访存地址能够形成一个流但访问地址间隔大于 CACHE 行时，虽然采取了预取，CACHE 访问仍然不命中。而我们的流预取方法则很好的解决了上述问题。

6．下一步工作

目前我们的模拟是单节点的，我们正在开发多节点模拟器来研究多节点情况下远程内存结构的性能，分析处理器节点数增多对性能带来的影响。

目前的流分析是以页框为单位的，但以字节为单位的流可能已经很早出现在页框内部，但我们只有在该流持续了 3 个页框时才发现，因此如果我们分析的粒度变的更小，比如对单个地址信息进行分析，就可以把流就可以被更早的检测出来，因此预取操作就可以被提前。这样对于页框流的元素个数比较少的应用，如 SWIM，预取将带来更大的性能提高。但它要求检测/预取部件从更靠近 CPU 的地址相关部件获取页框基址信息，比如 CACHE，如图二中虚线所示。相应的，页框流检测算法必须足够简单以便及时地处理大量的地址信息。因此，高效率的页框流检测算法也是我们下一步研究的对象。

我们目前的工作中没有区分读操作和写操作，如果将它们分开处理，页框数据的规则性能否进一步提高也是我们关心的方面。目前的预取是基于页框流的，既然我们的结构中有大量富足的网络带宽，我们考虑进一步开发非流的规则性，根据它们来预取更多的数据来提升性能。

7．总结

远程内存服务器结构可以带来高共享性和高可扩展性，但同时也加重了 CPU 和内存之间的速度不匹配问题。本论文采用模拟的方法来分析远程内存结构下典型应用的性能，验证了一些应用在远程内存结构下性能将明显下降的观点。同时我们也发现，由于本地内存不命中地址信息中以页对齐得到的页框基址有较好的规则性，所以我们考虑采用页框流预取机制来隐藏应用程序访问远程内存的数据传输时延，通过对不命中页框基址信息中的流的特征进行分析，发现采用页框流预取机制可以很好提高远程内存结构下程序的性能。最后我们利用模拟器进行实验，结果表明，通过使用页框流预取机制，在远程内存模式下应用程序的性能得到了不同幅度地提升。

参考文献

[1]樊建平、陈明宇，“网格化的动态自组织体系结构 DSAG”,计算机研究与发展，2003.12
[2] Tushar Mohan, Bronis R. de Supinski, Sally A. McKee, Frank Mueller, Andy Yoo, Martin Schulz, “Identifying and Exploiting Spatial Regularity in Data Memory References”, Supercomputing Conference 2003, November 2003.
[3] Jean - Loup Baer, Teen.-Fu Chen, “An Effective On-chip Preloading Scheme to Reduce Data Access Penalty”, Supercomputing Conference'91, November 1991.
[4] Norman P. Jouppi, “Improving Direct-mapped Cache Performance by the Addition of a Small Fully associative Cache and Prefetch Buffers,” Proc. 17th International Symposium on Computer Architecture, May 1990.
[5] Neil Savage, “Linking with light”, IEEE Spectrum, Volumn 39, August 2002
[6] Weiwu Hu, Weisong Shi, Zhimin Tang, “JIAJIA: A software DSM system based on a new cache coherence protocol”, Proc. HPCN Europe 1999, 1999.
[7] Y. Kodama, H. Sakane, M. Sato, S. Sakai, Y. Yamaguchi, “Message-based efficient remote memory access on a highly parallel computer EM-X”, International Symposium on Parallel Architectures, Algorithms and Networks`94, 1994.
[8] J. Renau. “Memory Hierarchies in Intelligent Memories: Energy/Performance Design”, Master Thesis, Institution, January 2000.
[9] Palacharla, S. and R.E. Kessler, “Evaluating Stream Buffers as a Secondary Cache Replacement,” Proc. 21st International Symposium on Computer Architecture, April 1994.
[10] Yutao Zhong, Cheng Ding and Ken Kennedy, “Reuse Distance Analysis for Scientific Programs”, SC2003，2003.
[11] Earl C. Joseph II, Ph.D., Christopher G. Willard, Ph.D., and Debra Goldfarb, “NUMAflex Modular Computing: New Model for Scalable Computing Infrastructure”, report of IDC, 2000
[12] Steven P. VanderWiel David J. Lilja, “Data Prefetch Mechanisms”, ACM Computing Survey, June 2000.
[13] http://www.dgate.org/vmips/.
[14] http://www.tik.ee.ethz.ch/~topsy/.
[15] http://www.bitmover.com/lmbench/.