每个程序员都应该了解的内存知识【第一部分】

[编辑的话: Ulrich Drepper最近问我们，是不是有兴趣发表一篇他写的内存方面的长文。我们不用看太多就已经知道，LWN的读者们会喜欢这篇文章的。内存的使用常常是软件性能的决定性因子，而如何避免内存瓶颈的好文章却不好找。这篇文章应该会有所帮助。

他的原文很长，超过100页。我们把它分成了7篇，每隔一到两周发表一篇。7篇发完后，Ulrich会把全文发出来。

对原文重新格式化是个很有挑战性的工作，但愿结果会不错吧。为了便于网上阅读，我们把Ulrich的脚注{放在了文章里}，而互相引用的超链接(和[参考书目])要等到全文出来才能提供。

非常感谢Ultich，感谢他让LWN发表这篇文章，期待大家在不久的将来都能写出内存优化很棒的软件。]

1 简介

早期计算机比现在更为简单。系统的各种组件例如CPU，内存，大容量存储器和网口，由于被共同开发因而有非常均衡的表现。例如，内存和网口并不比CPU在提供数据的时候更（特别的）快。

曾今计算机稳定的基本结构悄然改变，硬件开发人员开始致力于优化单个子系统。于是电脑一些组件的性能大大的落后因而成为了瓶颈。由于开销的原因，大容量存储器和内存子系统相对于其他组件来说改善得更为缓慢。

大容量存储的性能问题往往靠软件来改善: 操作系统将常用(且最有可能被用)的数据放在主存中，因为后者的速度要快上几个数量级。或者将缓存加入存储设备中，这样就可以在不修改操作系统的前提下提升性能。{然而，为了在使用缓存时保证数据的完整性，仍然要作出一些修改。}这些内容不在本文的谈论范围之内，就不作赘述了。

而解决内存的瓶颈更为困难，它与大容量存储不同，几乎每种方案都需要对硬件作出修改。目前，这些变更主要有以下这些方式:

- RAM的硬件设计(速度与并发度)
- 内存控制器的设计
- CPU缓存
- 设备的直接内存访问(DMA)

本文主要关心的是CPU缓存和内存控制器的设计。在讨论这些主题的过程中，我们还会研究DMA。不过，我们首先会从当今商用硬件的设计谈起。这有助于我们理解目前在使用内存子系统时可能遇到的问题和限制。我们还会详细介绍RAM的分类，说明为什么会存在这么多不同类型的内存。

本文不会包括所有内容，也不会包括最终性质的内容。我们的讨论范围仅止于商用硬件，而且只限于其中的一小部分。另外，本文中的许多论题，我们只会点到为止，以达到本文目标为标准。对于这些论题，大家可以阅读其它文档，获得更详细的说明。

当本文提到操作系统特定的细节和解决方案时，针对的都是Linux。无论何时都不会包含别的操作系统的任何信息，作者无意讨论其他操作系统的情况。如果读者认为他/她不得不使用别的操作系统，那么必须去要求供应商提供其操作系统类似于本文的文档。

在开始之前最后的一点说明，本文包含大量出现的术语“经常”和别的类似的限定词。这里讨论的技术在现实中存在于很多不同的实现，所以本文只阐述使用得最广泛最主流的版本。在阐述中很少有地方能用到绝对的限定词。

1.1文档结构

这个文档主要视为软件开发者而写的。本文不会涉及太多硬件细节，所以喜欢硬件的读者也许不会觉得有用。但是在我们讨论一些有用的细节之前，我们先要描述足够多的背景。

在这个基础上，本文的第二部分将描述RAM（随机寄存器）。懂得这个部分的内容很好，但是此部分的内容并不是懂得其后内容必须部分。我们会在之后引用不少之前的部分，所以心急的读者可以跳过任何章节来读他们认为有用的部分。

第三部分会谈到不少关于CPU缓存行为模式的内容。我们会列出一些图标，这样你们不至于觉得太枯燥。第三部分对于理解整个文章非常重要。第四部分将简短的描述虚拟内存是怎么被实现的。这也是你们需要理解全文其他部分的背景知识之一。

第五部分会提到许多关于Non Uniform Memory Access (NUMA)系统。

第六部分是本文的中心部分。在这个部分里面，我们将回顾其他许多部分中的信息，并且我们将给阅读本文的程序员许多在各种情况下的编程建议。如果你真的很心急，那么你可以直接阅读第六部分，并且我们建议你在必要的时候回到之前的章节回顾一下必要的背景知识。

本文的第七部分将介绍一些能够帮助程序员更好的完成任务的工具。即便在彻底理解了某一项技术的情况下，距离彻底理解在非测试环境下的程序还是很遥远的。我们需要借助一些工具。

第八部分，我们将展望一些在未来我们可能认为好用的科技。

1.2 反馈问题

作者会不定期更新本文档。这些更新既包括伴随技术进步而来的更新也包含更改错误。非常欢迎有志于反馈问题的读者发送电子邮件。

1.3 致谢

我首先需要感谢Johnray Fuller尤其是Jonathan Corbet，感谢他们将作者的英语转化成为更为规范的形式。Markus Armbruster提供大量本文中对于问题和缩写有价值的建议。

1.4 关于本文

本文题目对David Goldberg的经典文献《What Every Computer Scientist Should Know About Floating-Point Arithmetic》[goldberg]表示致敬。Goldberg的论文虽然不普及，但是对于任何有志于严格编程的人都会是一个先决条件。

2 商用硬件现状

鉴于目前专业硬件正在逐渐淡出，理解商用硬件的现状变得十分重要。现如今，人们更多的采用水平扩展，也就是说，用大量小型、互联的商用计算机代替巨大、超快(但超贵)的系统。原因在于，快速而廉价的网络硬件已经崛起。那些大型的专用系统仍然有一席之地，但已被商用硬件后来居上。2007年，Red Hat认为，未来构成数据中心的“积木”将会是拥有最多4个插槽的计算机，每个插槽插入一个四核CPU，这些CPU都是超线程的。{超线程使单个处理器核心能同时处理两个以上的任务，只需加入一点点额外硬件}。也就是说，这些数据中心中的标准系统拥有最多64个虚拟处理器。当然可以支持更大的系统，但人们认为4插槽、4核CPU是最佳配置，绝大多数的优化都针对这样的配置。

在不同商用计算机之间，也存在着巨大的差异。不过，我们关注在主要的差异上，可以涵盖到超过90%以上的硬件。需要注意的是，这些技术上的细节往往日新月异，变化极快，因此大家在阅读的时候也需要注意本文的写作时间。

这么多年来，个人计算机和小型服务器被标准化到了一个芯片组上，它由两部分组成: 北桥和南桥，见图2.1。

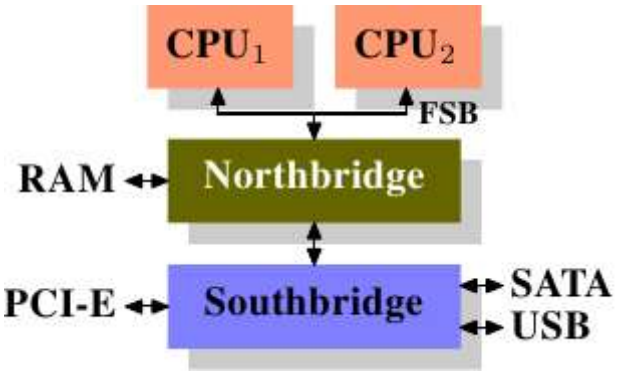


图2.1 北桥和南桥组成的结构

CPU通过一条通用总线(前端总线，FSB)连接到北桥。北桥主要包括内存控制器和其它一些组件，内存控制器决定了RAM芯片的类型。不同的类型，包括DRAM、Rambus和SDRAM等等，要求不同的内存控制器。

为了连通其它系统设备，北桥需要与南桥通信。南桥又叫I/O桥，通过多条不同总线与设备们通信。目前，比较重要的总线有PCI、PCI Express、SATA和USB总线，除此以外，南桥还支持PATA、IEEE 1394、串行口和并行口等。比较老的系统上有连接北桥的AGP槽。那是由于南北桥间缺乏高速连接而采取的措施。现在的PCI-E都是直接连到南桥的。

这种结构有一些需要注意的地方:

- 从某个CPU到另一个CPU的数据需要走它与北桥通信的同一条总线。
- 与RAM的通信需要经过北桥
- RAM只有一个端口。{本文不会介绍多端口RAM，因为商用硬件不采用这种内存，至少程序员无法访问到。这种内存一般在路由器等专用硬件中采用。}
- CPU与南桥设备间的通信需要经过北桥

在上面这种设计中，瓶颈马上出现了。第一个瓶颈与设备对RAM的访问有关。早期，所有设备之间的通信都需要经过CPU，结果严重影响了整个系统的性能。为了解决这个问题，有些设备加入了直接内存访问(DMA)的能力。DMA允许设备在北桥的帮助下，无需CPU的干涉，直接读写RAM。到了今天，所有高性能的设备都可以使用DMA。虽然DMA大大降低了CPU的负担，却占用了北桥的带宽，与CPU形成了争用。

第二个瓶颈来自北桥与RAM间的总线。总线的具体情况与内存的类型有关。在早期的系统上，只有一条总线，因此不能实现并行访问。近期的RAM需要两条独立总线(或者说通道，DDR2就是这么叫的，见图2.8)，可以实现带宽加倍。北桥将内存访问交错地分配到两个通道上。更新的内存技术(如FB-DRAM)甚至加入了更多的通道。

由于带宽有限，我们需要以一种使延迟最小化的方式来对内存访问进行调度。我们将会看到，处理器的速度比内存要快得多，需要等待内存。如果有多个超线程核心或CPU同时访问内存，等待时间则会更长。对于DMA也是同样。

除了并发以外，访问模式也会极大地影响内存子系统、特别是多通道内存子系统的性能。关于访问模式，可参见2.2节。

在一些比较昂贵的系统上，北桥自己不含内存控制器，而是连接到外部的多个内存控制器上(在下例中，共有4个)。

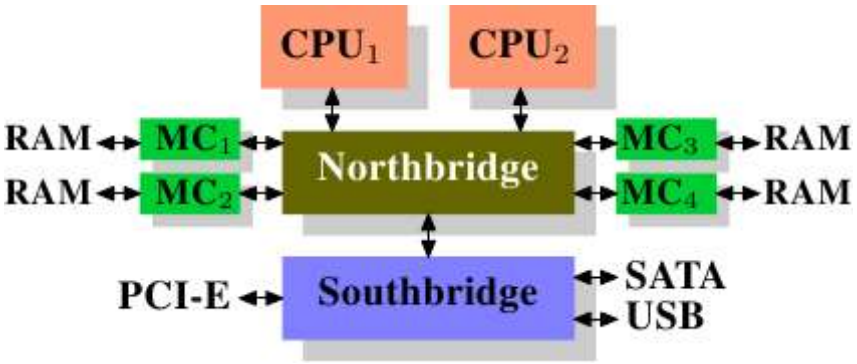


图2.2 拥有外部控制器的北桥

这种架构的好处在于，多条内存总线的存在，使得总带宽也随之增加了。而且也可以支持更多的内存。通过同时访问不同内存区，还可以降低延时。对于像图2.2中这种多处理器直连北桥的设计来说，尤其有效。而这种架构的局限在于北桥的内部带宽，非常巨大(来自Intel)。{出于完整性的考虑，还需要补充一下，这样的内存控制器布局还可以用于其它用途，比如说「内存RAID」，它可以与热插拔技术一起使用。}

使用外部内存控制器并不是唯一的办法，另一个最近比较流行的方法是将控制器集成到CPU内部，将内存直连到每个CPU。这种架构的走红归功于基于AMD Opteron处理器的SMP系统。图2.3展示了这种架构。Intel则会从Nehalem处理器开始支持通用系统接口(CSI)，基本上也是类似的思路——集成内存控制器，为每个处理器提供本地内存。

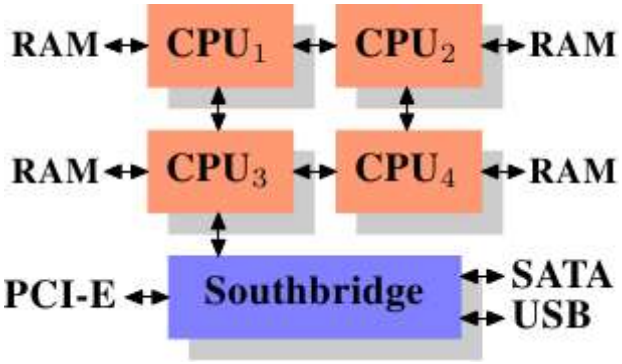


图2.3 集成的内存控制器

通过采用这样的架构，系统里有几个处理器，就可以有几个内存库(memory bank)。比如，在4CPU的计算机上，不需要一个拥有巨大带宽的复杂北桥，就可以实现4倍的内存带宽。另外，将内存控制器集成到CPU内部还有其它一些优点，这里就不赘述了。

同样也有缺点。首先，系统仍然要让所有内存能被所有处理器所访问，导致内存不再是统一的资源(NUMA即得名于此)。处理器能以正常的速度访问本地内存(连接到该处理器的内存)。但它访问其它处理器的内存时，却需要使用处理器之间的互联通道。比如说，CPU 1如果要访问CPU 2的内存，则需要使用它们之间的互联通道。如果它需要访问CPU 4的内存，那么需要跨越两条互联通道。

使用互联通道是有代价的。在讨论访问远端内存的代价时，我们用「NUMA因子」这个词。在图2.3中，每个CPU有两个层级: 相邻的CPU，以及两个互联通道外的CPU。在更加复杂的系统中，层级也更多。甚至有些机器有不止一种连接，比如说IBM的x445和SGI的Altix系列。CPU被归入节点，节点内的内存访问时间是一致的，或者只有很小的NUMA因子。而在节点之间的连接代价很大，而且有巨大的NUMA因子。

目前，已经有商用的NUMA计算机，而且它们在未来应该会扮演更加重要的角色。人们预计，从2008年底开始，每台SMP机器都会使用NUMA。每个在NUMA上运行的程序都应该认识到NUMA的代价。在第5节中，我们将讨论更多的架构，以及Linux内核为这些程序提供的一些技术。

除了本节中所介绍的技术之外，还有其它一些影响RAM性能的因素。它们无法被软件所左右，所以没有放在这里。如果大家有兴趣，可以在第2.1节中看一下。介绍这些技术，仅仅是因为它们能让我们绘制的RAM技术全图更为完整，或者是可能在大家购买计算机时能够提供一些帮助。

以下的两节主要介绍一些入门级的硬件知识，同时讨论内存控制器与DRAM芯片间的访问协议。这些知识解释了内存访问的原理，程序员可能会得到一些启发。不过，这部分并不是必读的，心急的读者可以直接跳到第2.2.5节。

2.1 RAM类型

这些年来，出现了许多不同类型的RAM，各有差异，有些甚至有非常巨大的不同。那些很古老的类型已经乏人问津，我们就不仔细研究了。我们主要专注于几类现代RAM，剖开它们的表面，研究一下内核和应用开发人员们可以看到的一些细节。

第一个有趣的细节是，为什么在同一台机器中有不同的RAM？或者说得更详细一点，为什么既有静态RAM(SRAM {SRAM还可以表示「同步内存」。})，又有动态RAM(DRAM)。功能相同，前者更快。那么，为什么不全部使用SRAM？答案是，代价。无论在生产还是在使用上，SRAM都比DRAM要贵得多。生产和使用，这两个代价因子都很重要，后者则是越来越重要。为了理解这一点，我们分别看一下SRAM和DRAM一个位的存储的实现过程。

在本节的余下部分，我们将讨论RAM实现的底层细节。我们将尽量控制细节的层面，比如，在「逻辑的层面」讨论信号，而不是硬件设计师那种层面，因为那毫无必要。

2.1.1 静态RAM

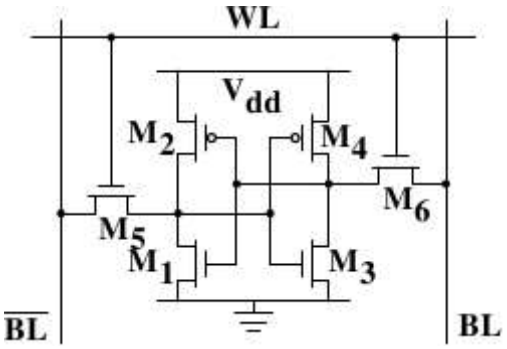


图2.6 6-T静态RAM

图2.4展示了6晶体管SRAM的一个单元。核心是4个晶体管M1-M4，它们组成两个交叉耦合的反相器。它们有两个稳定的状态，分别代表0和1。只要保持Vdd有电，状态就是稳定的。

当需要访问单元的状态时，升起字访问线WL。BL和BL上就可以读取状态。如果需要覆盖状态，先将BL和BL设置为期望的值，然后升起WL。由于外部的驱动强于内部的4个晶体管，所以旧状态会被覆盖。

更多详情，可以参考[sramwiki]。为了下文的讨论，需要注意以下问题:

一个单元需要6个晶体管。也有采用4个晶体管的SRAM，但有缺陷。

维持状态需要恒定的电源。

升起WL后立即可以读取状态。信号与其它晶体管控制的信号一样，是直角的(快速在两个状态间变化)。

状态稳定，不需要刷新循环。

SRAM也有其它形式，不那么费电，但比较慢。由于我们需要的是快速RAM，因此不在关注范围内。这些较慢的SRAM的主要优点在于接口简单，比动态RAM更容易使用。

2.1.2 动态RAM

动态RAM比静态RAM要简单得多。图2.5展示了一种普通DRAM的结构。它只含有一个晶体管和一个电容器。显然，这种复杂性上的巨大差异意味着功能上的迥异。

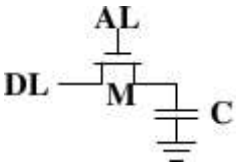


图2.5 1-T动态RAM

动态RAM的状态是保持在电容器C中。晶体管M用来控制访问。如果要读取状态，升起访问线AL，这时，可能会有电流流到数据线DL上，也可能没有，取决于电容器是否有电。如果要写入状态，先设置DL，然后升起AL一段时间，直到电容器充电或放电完毕。

动态RAM的设计有几个复杂的地方。由于读取状态时需要对电容器放电，所以这一过程不能无限重复，不得不在某个点上对它重新充电。

更糟糕的是，为了容纳大量单元(现在一般在单个芯片上容纳10的9次方以上的RAM单元)，电容器的容量必须很小(0.000000000000001法拉以下)。这样，完整充电后大约持有几万个电子。即使电容器的电阻很大(若干兆欧姆)，仍然只需很短的时间就会耗光电荷，称为「泄漏」。

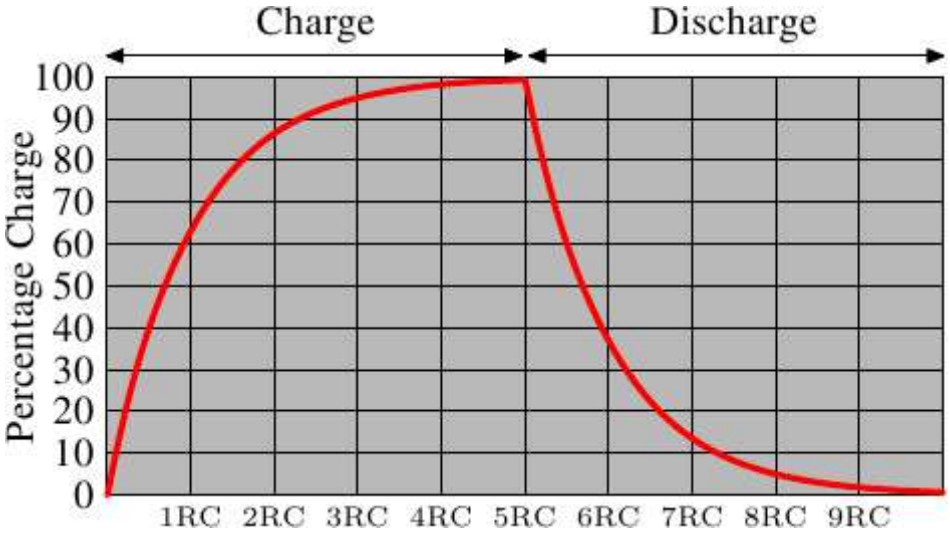
这种泄露就是现在的大部分DRAM芯片每隔64ms就必须进行一次刷新的原因。在刷新期间，对于该芯片的访问是不可能的，这甚至会造成半数任务的延宕。（相关内容请察看【highperfdram】一章）

这个问题的另一个后果就是无法直接读取芯片单元中的信息，而必须通过信号放大器将0和1两种信号间的电势差增大。

最后一个问题在于电容器的冲放电是需要时间的，这就导致了信号放大器读取的信号并不是典型的矩形信号。所以当放大器输出信号的时候就需要一个小小的延宕，相关公式如下

$$
\begin{aligned}
Q_{\text{Charge}}(t) &= Q_0(1 - e^{-\frac{t}{RC}}) \\
Q_{\text{Discharge}}(t) &= Q_0 e^{-\frac{t}{RC}}
\end{aligned}
$$

这就意味着需要一些时间（时间长短取决于电容C和电阻R）来对电容进行冲放电。另一个负面作用是，信号放大器的输出电流不能立即就作为信号载体使用。图2.6显示了冲放电的曲线，x轴表示的是单位时间下的R*C



与静态RAM可以即刻读取数据不同的是，当要读取动态RAM的时候，必须花一点时间来等待电容的冲放电完全。这一点点的时间最终限制了DRAM的速度。

当然了，这种读取方式也是有好处的。最大的好处在于缩小了规模。一个动态RAM的尺寸是小于静态RAM的。这种规模的减小不单单建立在动态RAM的简单结构之上，也是由于减少了静态RAM的各个单元独立的供电部分。以上也同时导致了动态RAM模具的简单化。

综上所述，由于不可思议的成本差异，除了一些特殊的硬件（包括路由器什么的）之外，我们的硬件大多是使用DRAM的。这一点深深的影响了咱们这些程序员，后文将会对此进行讨论。在此之前，我们还是先了解下DRAM的更多细节。

2.1.3 DRAM 访问

一个程序选择了一个内存位置使用到了一个虚拟地址。处理器转换这个到物理地址最后将内存控制选择RAM芯片匹配了那个地址。在RAM芯片去选择单个内存单元，部分的物理地址以许多地址行的形式被传递。

它单独地去处理来自于内存控制器的内存位置将完全不切实际：4G的RAM将需要 2^{32} 地址行。地址传递DRAM芯片的这种方式首先必须被路由器解析。一个路由器的N多地址行将有2^N 输出行。这些输出行能被使用到选择内存单元。使用这个直接方法对于小容量芯片不再是个大问题

但如果许多的单元生成这种方法不在适合。一个1G的芯片容量（我反感那些SI前缀，对于我一个*giga-bit将总是2^{30} 而不是10^9*字节）将需要30地址行和2^{30} 选项行。一个路由器的大小及许多的输入行以指数方式递增当速度不被牺牲时。一个30地址行路由器需要一大堆芯片的真实身份另外路由器也就复杂起来了。更重要的是，传递30脉冲在地址行同步要比仅仅传递15脉冲困难的多。较少列能精确布局相同长度或恰当的时机（现代DRAM类型像DDR3能自动调整时序但这个限制能让他什么都能忍受）

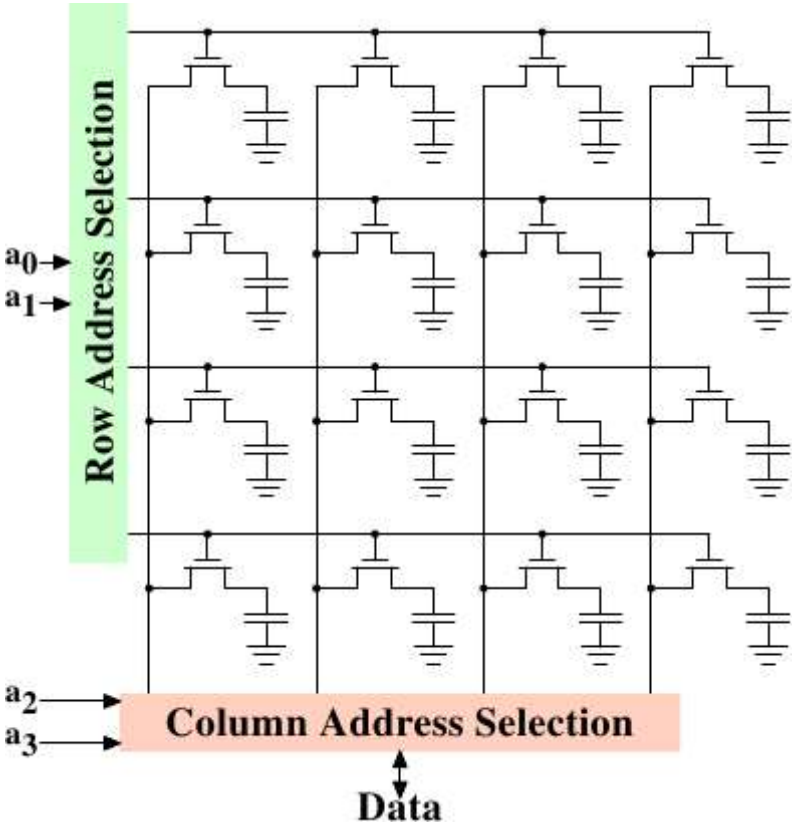


图2.7展示了一个很高级别的一个DRAM芯片，DRAM被组织在行和列里。他们能在一行中对奇但DRAM芯片需要一个大的路由器。通过阵列方法设计能被一个路由器和一个半的multiplexer获得{多路复用器（multiplexer）和路由器是一样的，这的multiplexer需要以路由器身份工作当写数据时候。那么从现在开始我们开始讨论其区别.}这在所有方面会是一个大的存储。例如地址linesa$_{0和}$ a$_1$通过行地址选择路由器来选择整个行的芯片的地址列，当读的时候，所有的芯片目录能使其纵列选择路由器可用，依据地址linesa$_{2和}$a$_3$一个纵列的目录用于数据DRAM芯片的接口类型。这发生了许多次在许多DRAM芯片产生一个总记录数的字节匹配给一个宽范围的数据总线。

对于写操作，内存单元的数据新值被放到了数据总线，当使用RAS和CAS方式选中内存单元时，数据是存放在内存单元内的。这是一个相当直观的设计，在现实中——很显然——会复杂得多，对于读，需要规范从发出信号到数据在数据总线上变得可读的时延。电容不会像前面章节里面描述的那样立刻自动放电，从内存单元发出的信号是如此这微弱以至于它需要被放大。对于写，必须规范从数据RAS和CAS操作完成后到数据成功的被写入内存单元的时延（当然，电容不会立刻自动充电和放电）。这些时间常量对于DRAM芯片的性能是至关重要的，我们将在下章讨论它。

另一个关于伸缩性的问题是，用30根地址线连接到每一个RAM芯片是行不通的。芯片的针脚是非常珍贵的资源，以至数据必须能并行传输就并行传输（比如：64位为一组）。内存控制器必须有能力解析每一个RAM模块（RAM芯片集合）。如果因为性能的原因要求并发行访问多个RAM模块并且每个RAM模块需要自己独占的30或多个地址线，那么对于8个RAM模块，仅仅是解析地址，内存控制器就需要240+之多的针脚。

在很长一段时间里，地址线被复用以解决DRAM芯片的这些次要的可扩展性问题。这意味着地址被转换成两部分。第一部分由地址位a0和a1选择行（如图2.7）。这个选择保持有效直到撤销。然后是第二部分，地址位a2和a3选择列。关键差别在于：只需要两根外部地址线。需要一些很少的线指明RAS和CAS信号有效，但是把地址线的数目减半所付出的代价更小。可是地址复用也带来自身的一些问题。我们将在2.2章中提到。

2.1.4 总结

如果这章节的内容有些难以应付，不用担心。纵观这章节的重点，有：

- 为什么不是所有的存储器都是SRAM的原因
- 存储单元需要单独选择来使用
- 地址线数目直接负责存储控制器，主板，DRAM模块和DRAM芯片的成本
- 在读或写操作结果之前需要占用一段时间是可行的

接下来的章节会涉及更多的有关访问DRAM存储器的实际操作的细节。我们不会提到更多有关访问SRAM的具体内容，它通常是直接寻址。这里是由于速度和有限的SRAM存储器的尺寸。SRAM现在应用在CPU的高速缓存和芯片，它们的连接件很小而且完全能在CPU设计师的掌控之下。我们以后会讨论到CPU高速缓存这个主题，但我们所需要知道的是SRAM存储单元是有确定的最大速度，这取决于花在SRAM上的艰难的尝试。这速度与CPU核心相比略慢一到两个数量级。

2.2 DRAM访问细节

在上文介绍DRAM的时候，我们已经看到DRAM芯片为了节约资源，对地址进行了复用。而且，访问DRAM单元是需要一些时间的，因为电容器的放电并不是瞬时的。此外，我们还看到，DRAM需要不停地刷新。在这一节里，我们将把这些因素拼合起来，看看它们是如何决定DRAM的访问过程。

我们将主要关注在当前的科技上，不会再去讨论异步DRAM以及它的各种变体。如果对它感兴趣，可以去参考[highperfdram]及[arstechtwo]。我们也不会讨论Rambus DRAM(RDRAM)，虽然它并不过时，但在系统内存领域应用不广。我们将主要介绍同步DRAM(SDRAM)及其后继者双倍速DRAM(DDR)。

同步DRAM，顾名思义，是参照一个时间源工作的。由内存控制器提供一个时钟，时钟的频率决定了前端总线(FSB)的速度。FSB是内存控制器提供给DRAM芯片的接口。在我写作本文的时候，FSB已经达到800MHz、1066MHz，甚至1333MHz，并且下一代的1600MHz也已经宣布。但这并不表示时钟频率有这么高。实际上，目前的总线都是双倍或四倍传输的，每个周期传输2次或4次数据。报的越高，卖的越好，所以这些厂商们喜欢把四倍传输的200MHz总线宣传为“有效的”800MHz总线。

以今天的SDRAM为例，每次数据传输包含64位，即8字节。所以FSB的传输速率应该是有效总线频率乘于8字节(对于4倍传输200MHz总线而言，传输速率为6.4GB/s)。听起来很高，但要知道这只是峰值速率，实际上无法达到的最高速率。我们将会看到，与RAM模块交流的协议有大量时间是处于非工作状态，不进行数据传输。我们必须对这些非工作时间有所了解，并尽量缩短它们，才能获得最佳的性能。

2.2.1 读访问协议

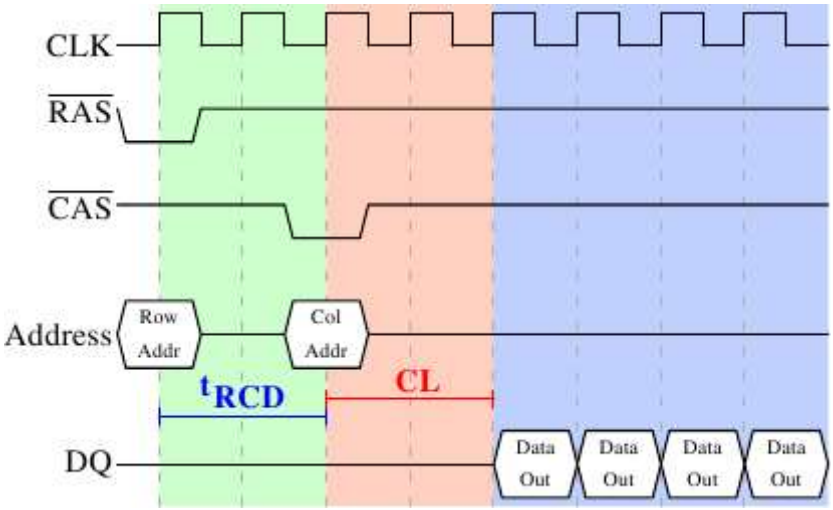


图2.8: SDRAM读访问的时序

图2.8展示了某个DRAM模块一些连接器上的活动，可分为三个阶段，图上以不同颜色表示。按惯例，时间为从左向右流逝。这里忽略了许多细节，我们只关注时钟频率、RAS与CAS信号、地址总线和数据总线。首先，内存控制器将行地址放在地址总线上，并降低RAS信号，读周期开始。所有信号都在时钟(CLK)的上升沿读取，因此，只要信号在读取的时间点上保持稳定，就算不是标准的方波也没有关系。设置行地址会促使RAM芯片锁住指定的行。

CAS信号在tRCD(RAS到CAS时延)个时钟周期后发出。内存控制器将列地址放在地址总线上，降低CAS线。这里我们可以看到，地址的两个组成部分是怎么通过同一条总线传输的。

至此，寻址结束，是时候传输数据了。但RAM芯片任然需要一些准备时间，这个时间称为CAS时延(CL)。在图2.8中CL为2。这个值可大可小，它取决于内存控制器、主板和DRAM模块的质量。CL还可能是半周期。假设CL为2.5，那么数据将在蓝色区域内的第一个下降沿准备就绪。

既然数据的传输需要这么多的准备工作，仅仅传输一个字显然是太浪费了。因此，DRAM模块允许内存控制指定本次传输多少数据。可以是2、4或8个字。这样，就可以一次填满高速缓存的整条线，而不需要额外的RAS/CAS序列。另外，内存控制器还可以在不重置行选择的前提下发送新的CAS信号。这样，读取或写入连续的地址就可以变得非常快，因为不需要发送RAS信号，也不需要把行置为非激活状态(见下文)。是否要将行保持为“打开”状态是内存控制器判断的事情。让它一直保持打开的话，对真正的应用会有不好的影响(参见[highperfdram])。CAS信号的发送仅与RAM模块的命令速率(Command Rate)有关(常常记为T*x*，其中*x*为1或2，高性能的DRAM模块一般为1，表示在每个周期都可以接收新命令)。

在上图中，SDRAM的每个周期输出一个字的数据。这是第一代的SDRAM。而DDR可以在一个周期中输出两个字。这种做法可以减少传输时间，但无法降低时延。DDR2尽管看上去不同，但在本质上也是相同的做法。对于DDR2，不需要再深入介绍了，我们只需要知道DDR2更快、更便宜、更可靠、更节能(参见[ddrtwo])就足够了。

2.2.2 预充电与激活

图2.8并不完整，它只画出了访问DRAM的完整循环的一部分。在发送RAS信号之前，必须先把当前锁住的行置为非激活状态，并对新行进行预充电。在这里，我们主要讨论由于显式发送指令而触发以上行为的情况。协议本身作了一些改进，在某些情况下是可以省略这个步骤的，但预充电带来的时延还是会影响整个操作。

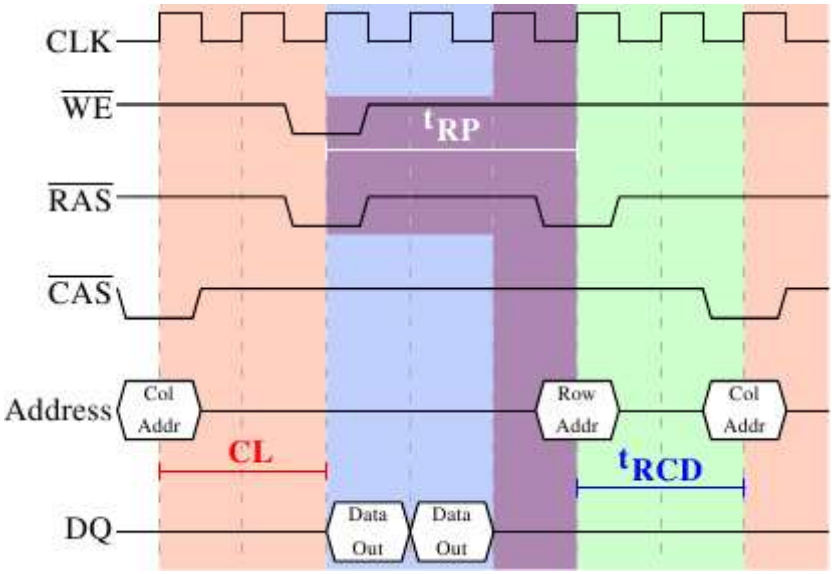


图2.9: SDRAM的预充电与激活

图2.9显示的是两次CAS信号的时序图。第一次的数据在CL周期后准备就绪。图中的例子里，是在SDRAM上，用两个周期传输了两个字的数据。如果换成DDR的话，则可以传输4个字。

即使是在一个命令速率为1的DRAM模块上，也无法立即发出预充电命令，而要等数据传输完成。在上图中，即为两个周期。刚好与CL相同，但只是巧合而已。预充电信号并没有专用线，某些实现是用同时降低写使能(WE)线和RAS线的方式来触发。这一组合方式本身没有特殊的意义(参见[micronddr])。

发出预充电信命令后，还需等待t_{RP}(行预充电时间)个周期之后才能使行被选中。在图2.9中，这个时间(紫色部分)大部分与内存传输的时间(淡蓝色部分)重合。不错。但t_{RP}大于传输时间，因此下一个RAS信号只能等待一个周期。

如果我们补充完整上图中的时间线，最后会发现下一次数据传输发生在前一次的5个周期之后。这意味着，数据总线的7个周期中只有2个周期才是真正在用的。再用它乘于FSB速度，结果就是，800MHz总线的理论速率6.4GB/s降到了1.8GB/s。真是太糟了。第6节将介绍一些技术，可以帮助我们提高总线有效速率。程序员们也需要尽自己的努力。

SDRAM还有一些定时值，我们并没有谈到。在图2.9中，预充电命令仅受制于数据传输时间。除此之外，SDRAM模块在RAS信号之后，需要经过一段时间，才能进行预充电(记为t_{RAS})。它的值很大，一般达到t_{RP}的2到3倍。如果在某个RAS信号之后，只有一个CAS信号，而且数据只传输很少几个周期，那么就有问题了。假设在图2.9中，第一个CAS信号是直接跟在一个RAS信号后免的，而t_{RAS}为8个周期。那么预充电命令还需要被推迟一个周期，因为t_{RCD}、CL和t_{RP}加起来才7个周期。

DDR模块往往用w-z-y-z-T来表示。例如，2-3-2-8-T1，意思是：

w 2 CAS时延(CL)
x 3 RAS-to-CAS时延(t_{RCD})
y 2 RAS预充电时间(t_{RP})
z 8 激活到预充电时间(t_{RAS})
T T1 命令速率

当然，除以上的参数外，还有许多其它参数影响命令的发送与处理。但以上5个参数已经足以确定模块的性能。

在解读计算机性能参数时，这些信息可能会派上用场。而在购买计算机时，这些信息就更有用了，因为它们与FSB/SDRAM速度一起，都是决定计算机速度的关键因素。

喜欢冒险的读者们还可以利用它们来调优系统。有些计算机的BIOS可以让你修改这些参数。SDRAM模块有一些可编程寄存器，可供设置参数。BIOS一般会挑选最佳值。如果RAM模块的质量足够好，我们可以在保持系统稳定的前提下将减小以上某个时延参数。互联网上有大量超频网站提供了相关的文档。不过，这是有风险的，需要大家自己承担，可别怪我没有事先提醒哟。

2.2.3 重充电

谈到DRAM的访问时，重充电是常常被忽略的一个主题。在2.1.2中曾经介绍，DRAM必须保持刷新。……行在充电时是无法访问的。[highperfdram]的研究发现，“令人吃惊，DRAM刷新对性能有着巨大的影响”。

根据JEDEC规范，DRAM单元必须保持每64ms刷新一次。对于8192行的DRAM，这意味着内存控制器平均每7.8125μs就需要发出一个刷新命令(在实际情况下，由于刷新命令可以纳入队列，因此这个时间间隔可以更大一些)。刷新命令的调度由内存控制器负责。DRAM模块会记录上一次刷新行的地址，然后在下次刷新请求时自动对这个地址进行递增。

对于刷新及发出刷新命令的时间点，程序员无法施加影响。但我们在解读性能参数时有必要知道，它也是DRAM生命周期的一个部分。如果系统需要读取某个重要的字，而刚好它所在的行正在刷新，那么处理器将会被延迟很长一段时间。刷新的具体耗时取决于DRAM模块本身。

2.2.4 内存类型

我们有必要花一些时间来了解一下目前流行的内存，以及那些即将流行的内存。首先从SDR(单倍速)SDRAM开始，因为它们是DDR(双倍速)SDRAM的基础。SDR非常简单，内存单元和数据传输率是相等的。

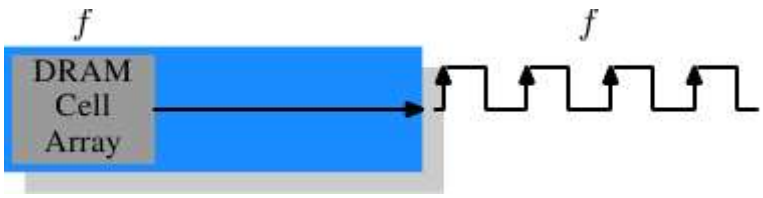


图2.10: SDR SDRAM的操作

在图2.10中，DRAM单元阵列能以等同于内存总线的速率输出内容。假设DRAM单元阵列工作在100MHz上，那么总线的数据传输率可以达到100Mb/s。所有组件的频率*f*保持相同。由于提高频率会导致耗电量增加，所以提高吞吐量需要付出很高的的代价。如果是很大规模的内存阵列，代价会非常巨大。{*功率 = 动态电容 x 电压*^{2} *x 频率*}。而且，提高频率还需要在保持系统稳定的情况下提高电压，这更是一个问题。因此，就有了DDR SDRAM(现在叫DDR1)，它可以在不提高频率的前提下提高吞吐量。

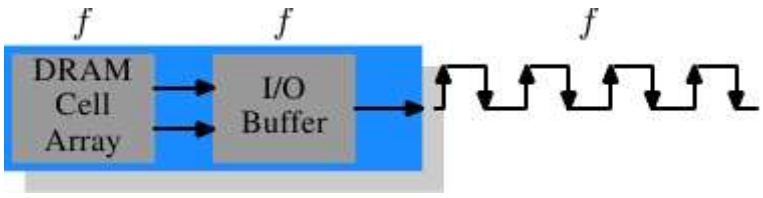


图2.11 DDR1 SDRAM的操作

我们从图2.11上可以看出DDR1与SDR的不同之处，也可以从DDR1的名字里猜到那么几分，DDR1的每个周期可以传输两倍的数据，它的上升沿和下降沿都传输数据。有时又被称为“双泵(double-pumped)”总线。为了在不提升频率的前提下实现双倍传输，DDR引入了一个缓冲区。缓冲区的每条数据线都持有两位。它要求内存单元阵列的数据总线包含两条线。实现的方式很简单，用同一个列地址同时访问两个DRAM单元。对单元阵列的修改也很小。

SDR DRAM是以频率来命名的(例如，对应于100MHz的称为PC100)。为了让DDR1听上去更好听，营销人员们不得不想了一种新的命名方案。这种新方案中含有DDR模块可支持的传输速率(DDR拥有64位总线):

100MHz x 64位 x 2 = 1600MB/s

于是，100MHz频率的DDR模块就被称为PC1600。由于1600 > 100，营销方面的需求得到了满足，听起来非常棒，但实际上仅仅只是提升了两倍而已。{我接受两倍这个事实，但不喜欢类似的数字膨胀戏法。}

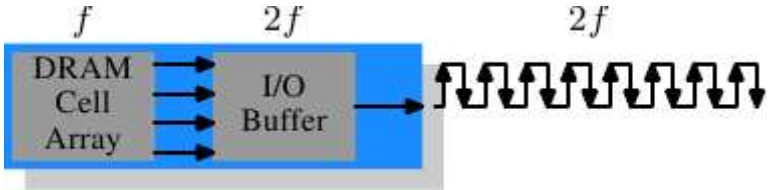


图2.12: DDR2 SDRAM的操作

为了更进一步，DDR2有了更多的创新。在图2.12中，最明显的变化是，总线的频率加倍了。频率的加倍意味着带宽的加倍。如果对单元阵列的频率加倍，显然是不经济的，因此DDR2要求I/O缓冲区在每个时钟周期读取4位。也就是说，DDR2的变化仅在于使I/O缓冲区运行在更高的速度上。这是可行的，而且耗电也不会显著增加。DDR2的命名与DDR1相仿，只是将因子2替换成4(四泵总线)。图2.13显示了目前常用的一些模块的名称。

阵列频率	总线频率	数据率	名称(速率)	名称 (FSB)
133MHz	266MHz	4,256MB/s	PC2-4200	DDR2-533
166MHz	333MHz	5,312MB/s	PC2-5300	DDR2-667
200MHz	400MHz	6,400MB/s	PC2-6400	DDR2-800
250MHz	500MHz	8,000MB/s	PC2-8000	DDR2-1000
266MHz	533MHz	8,512MB/s	PC2-8500	DDR2-1066

图2.13: DDR2模块名

在命名方面还有一个拧巴的地方。FSB速度是用有效频率来标记的，即把上升、下降沿均传输数据的因素考虑进去，因此数字被撑大了。所以，拥有266MHz总线的133MHz模块有着533MHz的FSB“频率”。

DDR3要求更多的改变(这里指真正的DDR3，而不是图形卡中假冒的GDDR3)。电压从1.8V下降到1.5V。由于耗电是与电压的平方成正比，因此可以节约30%的电力。加上管芯(die)的缩小和电气方面的其它进展，DDR3可以在保持相同频率的情况下，降低一半的电力消耗。或者，在保持相同耗电的情况下，达到更高的频率。又或者，在保持相同热量排放的情况下，实现容量的翻番。

DDR3模块的单元阵列将运行在内部总线的四分之一速度上，DDR3的I/O缓冲区从DDR2的4位提升到8位。见图2.14。

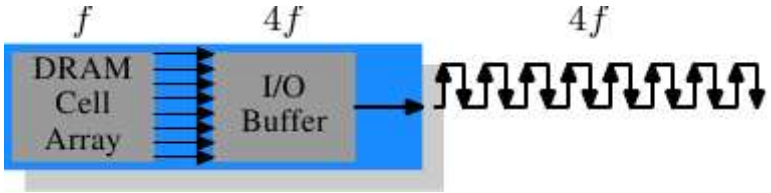


图2.14: DDR3 SDRAM的操作

一开始，DDR3可能会有较高的CAS时延，因为DDR2的技术相比之下更为成熟。由于这个原因，DDR3可能只会用于DDR2无法达到的高频率下，而且带宽比时延更重要的场景。此前，已经有讨论指出，1.3V的DDR3可以达到与DDR2相同的CAS时延。无论如何，更高速度带来的价值都会超过时延增加带来的影响。

DDR3可能会有一个问题，即在1600Mb/s或更高速率下，每个通道的模块数可能会限制为1。在早期版本中，这一要求是针对所有频率的。我们希望这个要求可以提高一些，否则系统容量将会受到严重的限制。

图2.15显示了我们预计中各DDR3模块的名称。JEDEC目前同意了前四种。由于Intel的45nm处理器是1600Mb/s的FSB，1866Mb/s可以用于超频市场。随着DDR3的发展，可能会有更多类型加入。

阵列频率	总线频率	数据速率	名称(速率)	名称 (FSB)
100MHz	400MHz	6,400MB/s	PC3-6400	DDR3-800
133MHz	533MHz	8,512MB/s	PC3-8500	DDR3-1066

166MHz	667MHz	10,667MB/s	PC3-10667	DDR3-1333
200MHz	800MHz	12,800MB/s	PC3-12800	DDR3-1600
233MHz	933MHz	14,933MB/s	PC3-14900	DDR3-1866

图2.15: DDR3模块名

所有的DDR内存都有一个问题：不断增加的频率使得建立并行数据总线变得十分困难。一个DDR2模块有240根引脚。所有到地址和数据引脚的连线必须被布置得差不多一样长。更大的问题是，如果多于一个DDR模块通过菊花链连接在同一个总线上，每个模块所接收到的信号随着模块的增加会变得越来越扭曲。DDR2规范允许每条总线（又称通道）连接最多两个模块，DDR3在高频率下只允许每个通道连接一个模块。每条总线多达240根引脚使得单个北桥无法以合理的方式驱动两个通道。替代方案是增加外部内存控制器（如图2.2），但这会提高成本。

这意味着商品主板所搭载的DDR2或DDR3模块数将被限制在最多四条，这严重限制了系统的最大内存容量。即使是老旧的32位IA-32处理器也可以使用64GB内存。即使是家庭对内存的需求也在不断增长，所以，某些事必须开始做了。

一种解法是，在处理器中加入内存控制器，我们在第2节中曾经介绍过。AMD的Opteron系列和Intel的CSI技术就是采用这种方法。只要我们能把处理器要求的内存连接到处理器上，这种解法就是有效的。如果不能，按照这种思路就会引入NUMA架构，当然同时也会引入它的缺点。而在有些情况下，我们需要其它解法。

Intel针对大型服务器方面的解法(至少在未来几年)，是被称为全缓冲DRAM(FB-DRAM)的技术。FB-DRAM采用与DDR2相同的器件，因此造价低廉。不同之处在于它们与内存控制器的连接方式。FB-DRAM没有用并行总线，而用了串行总线(Rambus DRAM had this back when, too, 而SATA是PATA的继任者，就像PCI Express是PCI/AGP的继承人一样)。串行总线可以达到更高的频率，串行化的负面影响，甚至可以增加带宽。使用串行总线后

1. 每个通道可以使用更多的模块。
2. 每个北桥/内存控制器可以使用更多的通道。
3. 串行总线是全双工的(两条线)。

FB-DRAM只有69个脚。通过菊花链方式连接多个FB-DRAM也很简单。FB-DRAM规范允许每个通道连接最多8个模块。

在对比下双通道北桥的连接性，采用FB-DRAM后，北桥可以驱动6个通道，而且脚数更少——6x69对比2x240。每个通道的布线也更为简单，有助于降低主板的成本。

全双工的并行总线过于昂贵。而换成串行线后，这不再是一个问题，因此串行总线按全双工来设计的，这也意味着，在某些情况下，仅靠这一特性，总线的理论带宽已经翻了一倍。还不止于此。由于FB-DRAM控制器可同时连接6个通道，因此可以利用它来增加某些小内存系统的带宽。对于一个双通道、4模块的DDR2系统，我们可以用一个普通FB-DRAM控制器，用4通道来实现相同的容量。串行总线的实际带宽取决于在FB-DRAM模块中所使用的DDR2(或DDR3)芯片的类型。

我们可以像这样总结这些优势：

DDR2 FB-DRAM

	DDR2	FB-DRAM
脚	240	69
通道	2	6
每通道DIMM数	2	8

最大内存	16GB	192GB
吞吐量	~10GB/s	~40GB/s

如果在单个通道上使用多个DIMM，会有一些问题。信号在每个DIMM上都会有延迟(尽管很小)，也就是说，延迟是递增的。不过，如果在相同频率和相同容量上进行比较，FB-DRAM总是能快过DDR2及DDR3，因为FB-DRAM只需要在每个通道上使用一个DIMM即可。而如果说到大型内存系统，那么DDR更是没有商用组件的解决方案。

2.2.5 结论

通过本节，大家应该了解到访问DRAM的过程并不是一个快速的过程。至少与处理器的速度相比，或与处理器访问寄存器及缓存的速度相比，DRAM的访问不算快。大家还需要记住CPU和内存的频率是不同的。Intel Core 2处理器运行在2.933GHz，而1.066GHz FSB有11:1的时钟比率(注: 1.066GHz的总线为四泵总线)。那么，内存总线上延迟一个周期意味着处理器延迟11个周期。绝大多数机器使用的DRAM更慢，因此延迟更大。在后续的章节中，我们需要讨论延迟这个问题时，请把以上的数字记在心里。

前文中读命令的时序图表明，DRAM模块可以支持高速数据传输。每个完整行可以被毫无延迟地传输。数据总线可以100%被占。对DDR而言，意味着每个周期传输2个64位字。对于DDR2-800模块和双通道而言，意味着12.8GB/s的速率。

但是，除非是特殊设计，DRAM的访问并不总是串行的。访问不连续的内存区意味着需要预充电和RAS信号。于是，各种速度开始慢下来，DRAM模块急需帮助。预充电的时间越短，数据传输所受的惩罚越小。

硬件和软件的预取(参见第6.3节)可以在时序中制造更多的重叠区，降低延迟。预取还可以转移内存操作的时间，从而减少争用。我们常常遇到的问题是，在这一轮中生成的数据需要被存储，而下一轮的数据需要被读出来。通过转移读取的时间，读和写就不需要同时发出了。

2.3 主存的其它用户

除了CPU外，系统中还有其它一些组件也可以访问主存。高性能网卡或大规模存储控制器是无法承受通过CPU来传输数据的，它们一般直接对内存进行读写(直接内存访问，DMA)。在图2.1中可以看到，它们可以通过南桥和北桥直接访问内存。另外，其它总线，比如USB等也需要FSB带宽，即使它们并不使用DMA，但南桥仍要通过FSB连接到北桥。

DMA当然有很大的优点，但也意味着FSB带宽会有更多的竞争。在有大量DMA流量的情况下，CPU在访问内存时必然会有更大的延迟。我们可以用一些硬件来解决这个问题。例如，通过图2.3中的架构，我们可以挑选不受DMA影响的节点，让它们的内存为我们的计算服务。还可以在每个节点上连接一个南桥，将FSB的负荷均匀地分担到每个节点上。除此以外，还有许多其它方法。我们将在第6节中介绍一些技术和编程接口，它们能够帮助我们通过软件的方式改善这个问题。

最后，还需要提一下某些廉价系统，它们的图形系统没有专用的显存，而是采用主存的一部分作为显存。由于对显存的访问非常频繁(例如，对于1024x768、16bpp、60Hz的显示设置来说，需要95MB/s的数据速率)，而主存并不像显卡上的显存，并没有两个端口，因此这种配置会对系统性能、尤其是时延造成一定的影响。如果大家对系统性能要求比较高，最好不要采用这种配置。这种系统带来的问题超过了本身的价值。人们在购买它们时已经做好了性能不佳的心理准备。

继续阅读：

- 第2节: CPU的高速缓存
- 第3节: 虚拟内存
- 第4节: NUMA系统

- 第5节: 程序员可以做什么 - 高速缓存的优化
- 第6节: 程序员可以做什么 - 多线程的优化
- 第7节: 内存性能工具
- 第8节: 未来的技术
- 第9节: 附录与参考书目

本文地址：https://www.oschina.net/translate/what-every-programmer-should-know-about-memory-part1
原文地址：http://lwn.net/Articles/250967/
