有一条特别有趣的平坦的山脊线，对于步长 1 垂直于步长轴，此时读吞吐量相对保持不变，为 12GB/s，即使工作集超出了 L1 和 L2 的大小。这显然是由于 Core i7 存储器系统中的硬件预取(prefetching)机制，它会自动地识别顺序的、步长为 1 的引用模式，试图在一些块被访问之前，将它们取到高速缓存中。虽然文档里没有记录这种预取算法的细节，但是从存储器山可以明显地看到这个算法对小步长效果最好——这也是代码中要使用步长为 1 的顺序访问的另一个理由。

如果我们从这座山中取出一个片段，保持步长为常数，如图 6-42 所示，我们就能很清楚地看到高速缓存的大小和时间局部性对性能的影响了。大小最大为 32KB 的工作集完全能放进 L1 d-cache 中，因此，读都是由 L1 来服务的，吞吐量保持在峰值 12GB/s 处。大小最大为 256KB 的工作集完全能放进统一的 L2 高速缓存中，对于大小最大为 8 M，工作集完全能放进统一的 L3 高速缓存中。更大的工作集大小主要由主存来服务。

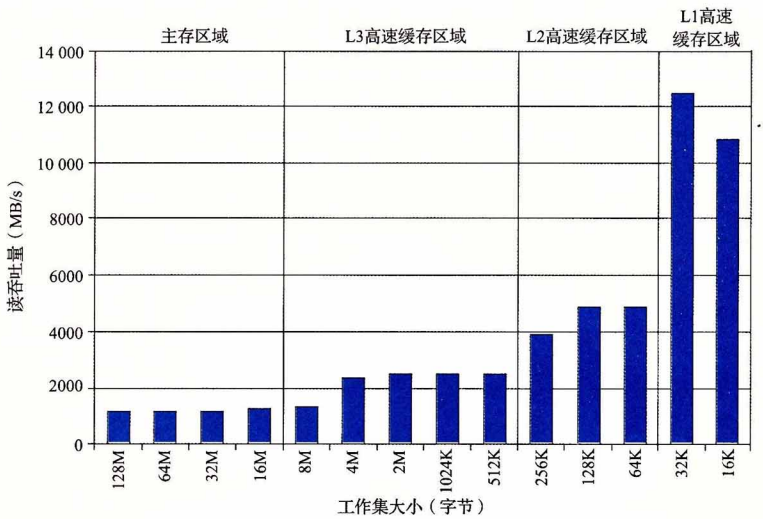

图 6-42　存储器山中时间局部性的山脊。这幅图展示了图 6-41 中 stride=8 时的一个片段

L2 和 L3 高速缓存区域最左边的边缘上读吞吐量的下降很有趣，此时工作集大小为 256KB 和 8MB，等于对应的高速缓存的大小。为什么会出现这样的下降，还不是完全清楚。要确认的唯一方法就是执行一个详细的高速缓存模拟，但是这些下降很有可能是与其他数据和代码行的冲突造成的。

以相反的方向横切这座山，保持工作集大小不变，我们从中能看到空间局部性对读吞吐量的影响。例如，图 6-43 展示了工作集大小固定为 4MB 时的片段。这个片段是沿着图 6-41 中的 L3 山脊切的，这里，工作集完全能够放到 L3 高速缓存中，但是对 L2 高速缓存来说太大了。

注意随着步长从 1 个字增长到 8 个字，读吞吐量是如何平稳地下降的。在山的这个区域中，L2 中的读不命中会导致一个块从 L3 传送到 L2。后面在 L2 中这个块上会有一定数量的命中，这是取决于步长的。随着步长的增加，L2 不命中与 L2 命中的比值也增加了。因为服务不命中要比命中更慢，所以读吞吐量也下降了。一旦步长达到了 8 个字，在这个系统上就等于块的大小 64 个字节了，每个读请求在 L2 中都会不命中，必须从 L3 服务。