我们在一个四核系统上，对一个大小为 $n=2^{31}$ 的序列运行 psum- mutex，测量它的运行时间(以秒为单位)，作为线程数的函数，得到的结果难懂又令人奇怪：

	线程数				
版本	1	2	4	8	16
psum-mutex	68	432	719	552	599

程序单线程顺序运行时非常慢，几乎比多线程并行运行时慢了一个数量级。不仅如此，使用的核数越多，性能越差。造成性能差的原因是相对于内存更新操作的开销，同步操作(P 和 V)代价太大。这突显了并行编程的一项重要教训：同步开销巨大，要尽可能避免。如果无可避免，必须要用尽可能多的有用计算弥补这个开销。

在我们的例子中，一种避免同步的方法是让每个对等线程在一个私有变量中计算它自己的部分和，这个私有变量不与其他任何线程共享，如图 12-33 所示。主线程(图中未显示)定义一个全局数组 psum，每个对等线程 i 把它的部分和累积在 psum[i]中。因为小心地给了每个对等线程一个不同的内存位置来更新，所以不需要用互斥锁来保护这些更新。唯一需要同步的地方是主线程必须等待所有的子线程完成。在对等线程结束后，主线程把 psum 向量的元素加起来，得到最终的结果。

code/conc/psum-array.c

```
1   /* Thread routine for psum-array.c */
2   void *sum_array(void *vargp)
3   {
4       long myid = *((long *)vargp);          /* Extract the thread ID */
5       long start = myid * nelems_per_thread; /* Start element index */
6       long end = start + nelems_per_thread;  /* End element index */
7       long i;
8
9       for (i = start; i < end; i++) {
10          psum[myid] += i;
11      }
12      return NULL;
13  }
```

code/conc/psum-array.c

图 12-33　psum-array 的线程例程。每个对等线程把它的部分和累积在一个私有数组元素中，不与其他任何对等线程共享该元素

在四核系统上运行 psum-array 时，我们看到它比 psum-mutex 运行得快好几个数量级：

	线程数				
版本	1	2	4	8	16
psum-mutex	68.00	432.00	719.00	552.00	599.00
psum-array	7.26	3.64	1.91	1.85	1.84

在第 5 章中，我们学习到了如何使用局部变量来消除不必要的内存引用。图 12-34 展示了如何应用这项原则，让每个对等线程把它的部分和累积在一个局部变量而不是全局变量中。当在四核机器上运行 psum-local 时，得到一组新的递减的运行时间：