$$
\begin{aligned}
p_0 &= a_0 \\
p_i &= p_{i-1} + a_i, \quad 1 \leqslant i < n
\end{aligned}
\tag{5.1}
$$

```
1   /* Compute prefix sum of vector a */
2   void psum1(float a[], float p[], long n)
3   {
4       long i;
5       p[0] = a[0];
6       for (i = 1; i < n; i++)
7           p[i] = p[i-1] + a[i];
8   }
9
10  void psum2(float a[], float p[], long n)
11  {
12      long i;
13      p[0] = a[0];
14      for (i = 1; i < n-1; i+=2) {
15          float mid_val = p[i-1] + a[i];
16          p[i]    = mid_val;
17          p[i+1]  = mid_val + a[i+1];
18      }
19      /* For even n, finish remaining element */
20      if (i < n)
21          p[i] = p[i-1] + a[i];
22  }
```

图 5-1　前置和函数。这些函数提供了我们如何表示程序性能的示例

函数 psum1 每次迭代计算结果向量的一个元素。第二个函数使用*循环展开*(loop unrolling)的技术，每次迭代计算两个元素。本章后面我们会探讨循环展开的好处。(关于分析和优化前置和计算的内容请参见练习题 5.11、5.12 和家庭作业 5.19。)

这样一个过程所需要的时间可以用一个常数加上一个与被处理元素个数成正比的因子来描述。例如，图 5-2 是这两个函数需要的周期数关于 n 的取值范围图。使用*最小二乘拟*

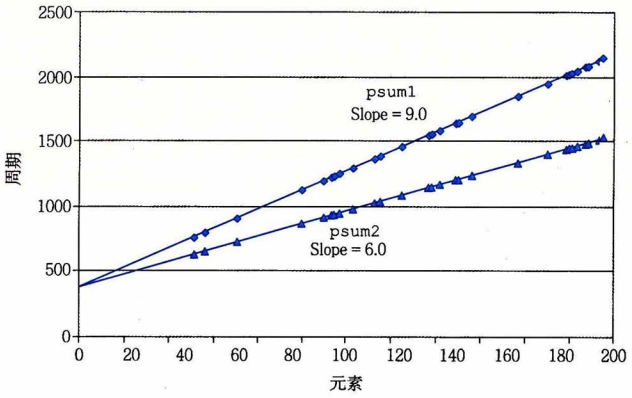

图 5-2　前置和函数的性能。两条线的斜率表明每元素的周期数(CPE)的值