

$$S_p = \frac{T_1}{T_p}$$

这里  $p$  是处理器核的数量,  $T_k$  是在  $k$  个核上的运行时间。这个公式有时被称为强扩展 (strong scaling)。当  $T_1$  是程序顺序执行版本的执行时间时,  $S_p$  称为绝对加速比 (absolute speedup)。当  $T_1$  是程序并行版本在一个核上的执行时间时,  $S_p$  称为相对加速比 (relative speedup)。绝对加速比比相对加速比能更真实地衡量并行的好处。即使是当并行程序在一个处理器上运行时, 也常常会受到同步开销的影响, 而这些开销会人为地增加相对加速比的数值, 因为它们增加了分子的大小。另一方面, 绝对加速比比相对加速比更难以测量, 因为测量绝对加速比需要程序的两种不同的版本。对于复杂的并行代码, 创建一个独立的顺序版本可能不太实际, 或者因为代码太复杂, 或者因为源代码不可得。

一种相关的测量量称为效率 (efficiency), 定义为

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$$

通常表示为范围在  $(0, 100]$  之间的百分比。效率是对由于并行化造成的开销的衡量。具有高效率的程序比效率低的程序在有用的工作上花费更多的时间, 在同步和通信上花费更少的时间。

图 12-36 给出了我们并行求和示例程序的各个加速比和效率测量值。像这样超过 90% 的效率是非常好的, 但是不要被欺骗了。能取得这么高的效率是因为我们的问题非常容易并行化。在实际中, 很少会这样。数十年来,


线程 ( $t$ )	1	2	4	8	16
核 ( $p$ )	1	2	4	4	4
运行时间 ( $T_p$ )	1.06	0.54	0.28	0.29	0.30
加速比 ( $S_p$ )	1	1.9	3.8	3.7	3.5
效率 ( $E_p$ )	100%	98%	95%	91%	88%

图 12-36 图 12-35 中执行时间的加速比和并行效率

并行编程一直是一个很活跃的研究领域。随着商用多核机器的出现, 这些机器的核数每几年就翻一番, 并行编程会继续是一个深入、困难而活跃的研究领域。

加速比还有另外一面, 称为弱扩展 (weak scaling), 在增加处理器数量的同时, 增加问题的规模, 这样随着处理器数量的增加, 每个处理器执行的工作量保持不变。在这种描述中, 加速比和效率被表达为单位时间完成的工作总量。例如, 如果将处理器数量翻倍, 同时每个小时也做了两倍的工作量, 那么我们就有线性的加速比和 100% 的效率。

弱扩展常常是比强扩展更真实的衡量值, 因为它更准确地反映了我们用更大的机器做更多的工作的愿望。对于科学计算程序来说尤其如此, 科学计算问题的规模很容易增加, 更大的问题规模直接就意味着更好地预测。不过, 还是有一些应用的规模不那么容易增加, 对于这样的应用, 强扩展是更合适的。例如, 实时信号处理应用所执行的工作量常常是由产生信号的物理传感器的属性决定的。改变工作总量需要用不同的物理传感器, 这不太实际或者不太必要。对于这类应用, 我们通常想要用并行来尽可能快地完成定量的工作。

 **练习题 12.11** 对于下表中的并行程序, 填写空白处。假设使用强扩展。

线程 ( $t$ )	1	2	4
核 ( $p$ )	1	2	4
运行时间 ( $T_p$ )	12	8	6
加速比 ( $S_p$ )		1.5	
效率 ( $E_p$ )	100%		50%