**Problem 1 and 2:**

*Assumption: Communication can be ignored and problem size is fixed.*

Ignoring communication allows for to be dropped, and the removal of n comes when the problem size is fixed. The resulting equation describes the speedup as a relationship between the fraction of the program that is serial and the number of cores.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| p | 2 | 4 | 8 | 16 | ∞ |
| Speedup | 1.74 | 2.76 | 3.90 | 4.92 | 6.6 |
| Efficiency | .86 | .69 | .49 | .31 | 0 |

**Problem 3 and 4:**

Assumption: Work scales to maintain value of s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| p | 2 | 4 | 8 | 16 |
| Speedup | 1.85 | 3.55 | 6.95 | 13.75 |
| Efficiency | .925 | .8875 | .868 | .859 |

**Problem 5:**

Solve for f:

**Problem 5 (cont.):**

Solve for p:

**Problem 6:**

Program 1:

This program scales extremely well. This implies that communication is not a growing issue, and that there is an increasingly large workload to accommodate the growing number of cores. Additionally, such an incredibly small e value implies that the serial fraction of the parallel computation is very small. This means that not only can quite a bit be done in parallel, but that almost all of the time the program is executed is spent in the parallel execution. Because so little time is spent in the serial code, the program is able to make full use of the added cores.

Program 2:

This program has much worse speedup than program 1, but despite the fact that e is larger; it is still quite small. Much smaller than expected considering the speedup. In this case the poor speedup measurements can be most easily explained by work that is not scaling proportionally. With an increase in the quantity amount of work, the fact that almost the entire program is spent in parallel execution can be utilized in a much stronger way.