# Scaling Linear Solvers to Petascale Systems

# Analytical Modeling – Basics

- A number of performance measures are intuitive.

- Wall clock time – the time from the start of the first processor to the stopping time of the last processor in a parallel ensemble. But how does this scale when the number of processors or problem instance is changed, or the program is ported to another machine?

- How much faster is the parallel version? This begs the obvious followup question – what is the baseline serial version with which we compare? Can we use a suboptimal serial program to make our parallel program look

- Raw FLOP count – What good are FLOP counts when they dont solve a problem?

# Scaling Characteristics of Parallel Programs

- The efficiency $E$ of a parallel program can be written in terms of speedup $S$, serial time $T_S$ and parallel time $T_P$ as:

$$E = \frac{S}{p} = \frac{T_S}{pT_P}$$

or

$$E = \frac{1}{1 + \frac{T_o}{T_S}}. \tag{1}$$

- The total overhead function $T_o$ here is defined as $T_o = pT_P - T_S$.

- The total overhead function $T_o$ is an increasing function of $p$.

# Scaling Characteristics of Parallel Programs

- For a given problem size (i.e., the value of $T_S$ remains constant), as we increase the number of processing elements, $T_o$ increases.

- The overall efficiency of the parallel program goes down. This is the case for all parallel programs.

# Scaling Characteristics of Parallel Programs

- Total overhead function $T_o$ is a function of both problem size $T_S$ and the number of processing elements $p$.
- In many cases, $T_o$ grows sublinearly with respect to $T_S$.
- In such cases, the efficiency increases if the problem size is increased keeping the number of processing elements constant.
- For such systems, we can simultaneously increase the problem size and number of processors to keep efficiency constant.
- Such systems are called *scalable* parallel systems.

# Scaling Characteristics of Parallel Programs

- Recall that cost-optimal parallel systems have an efficiency of $\Theta(1)$.

- Scalability and cost-optimality are therefore related.

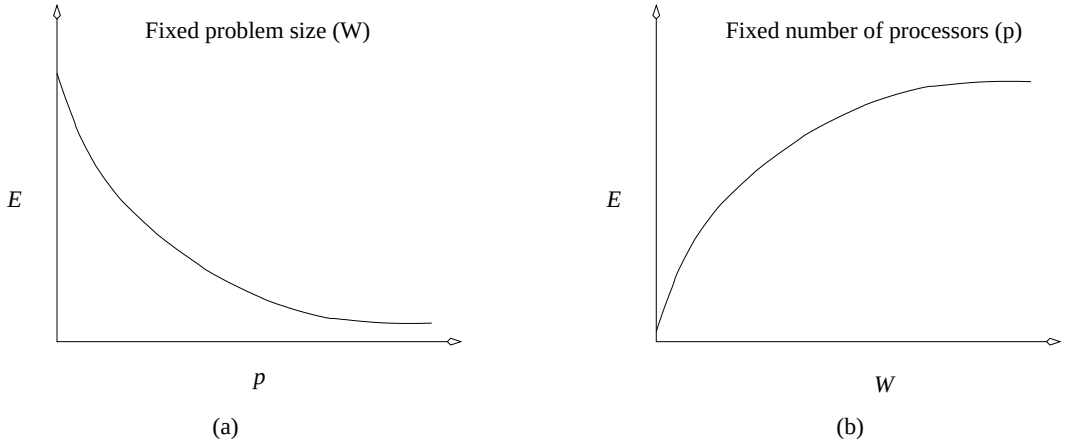
- A scalable parallel system can always be made cost-optimal if the number of processing elements and the size of the computation are chosen appropriately.

# Isoefficiency Metric of Scalability

- For a given problem size, as we increase the number of processing elements, the overall efficiency of the parallel system goes down for all systems.

- For some systems, the efficiency of a parallel system increases if the problem size is increased while keeping the number of processing elements constant.

# Isoefficiency Metric of Scalability




Variation of efficiency: (a) as the number of processing elements is in creased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

# Isoefficiency Metric of Scalability

- What is the rate at which the problem size must increase with respect to the number of processing elements to keep the efficiency fixed?

- This rate determines the scalability of the system. The slower this rate, the better.

# Isoefficiency Metric of Scalability

- We can write parallel runtime as:

$$T_P = \frac{W + T_o(W,p)}{p} \tag{2}$$

  Here, $W$ is the problem size, i.e., the number of basic steps in the algorithm.

- The resulting expression for speedup is

$$S = \frac{W}{T_P} = \frac{Wp}{W + T_o(W,p)}.$$

- Finally, we write the expression for efficiency as

$$E = \frac{S}{p} = \frac{W}{W + T_o(W,p)} = \frac{1}{1 + T_o(W,p)/W}. \tag{3}$$

# Isoefficiency Metric of Scalability

- For scalable parallel systems, efficiency can be maintained at a fixed value (between 0 and 1) if the ratio $T_o/W$ is maintained at a constant value.

- For a desired value $E$ of efficiency,

$$\begin{aligned} E &= \frac{1}{1+T_o(W,p)/W}, \\ \frac{T_o(W,p)}{W} &= \frac{1-E}{E}, \\ W &= \frac{E}{1-E}T_o(W,p). \end{aligned} \tag{4}$$

- If $K = E/(1-E)$ is a constant depending on the efficiency to be maintained, since $T_o$ is a function of $W$ and $p$, we have

$$W = KT_o(W,p). \tag{5}$$

# Isoefficiency Metric of Scalability

- The problem size $W$ can usually be obtained as a function of $p$ by algebraic manipulations to keep efficiency constant.

- This function is called the *isoefficiency function*.

- This function determines the ease with which a parallel system can maintain a constant efficiency and hence achieve speedups increasing in proportion to the number of processing elements.

# Pseudo-Analytical Approach to Performance Prediction

- Analytical approaches provide precise asymptotic estimates of scalability. They can be applied to algorithms and provide gross guidance to architecture design.

- Empirical approaches work on performance extrapolation. They are often inaccurate and work on realizations of programs. They provide little guidance to algorithm and architecture designers.

- Simulation-based approaches combine static analysis, dynamic profiling, and a discrete-event based simulation. They are often slow, and are only as accurate as the simulation.

- We adopt an analytical approach to performance modeling, but parametrize models using experimental data.

# Pseudo-Analytical Approach to Performance Prediction

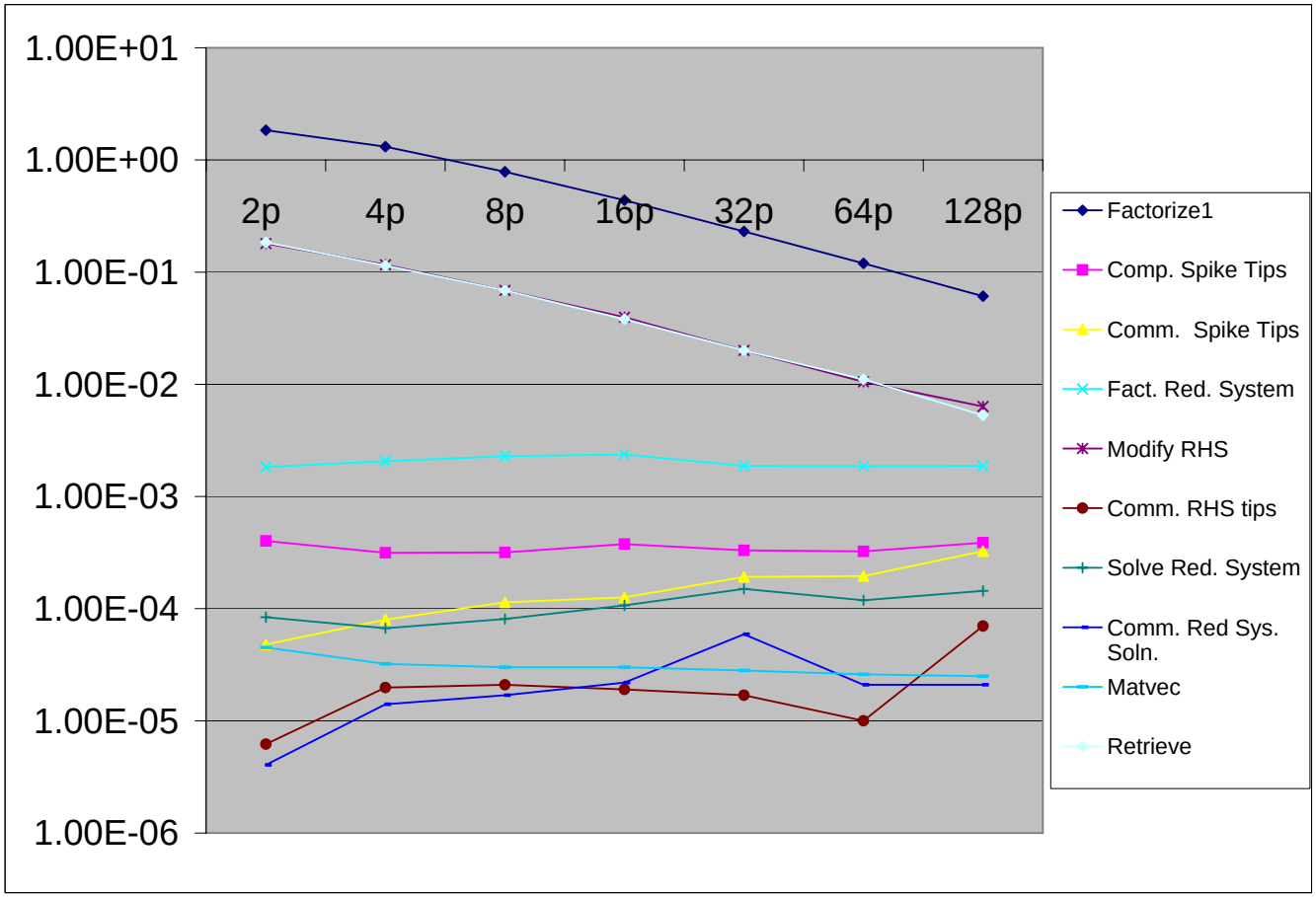
- Approach based on factoring out program behavior and machine characteristics.
- Characterize program behavior (computation requirements, communication overheads) using experimental data fitted on asymptotic estimates.
- Characterize platform using experimental data for communication overheads for underlying permutations.
- Match program and platform characterizations for accurate performance modeling.

# Scalability Case Study: Spike Solver Performance

Table 1: Spike_TA0 Algorithm Steps

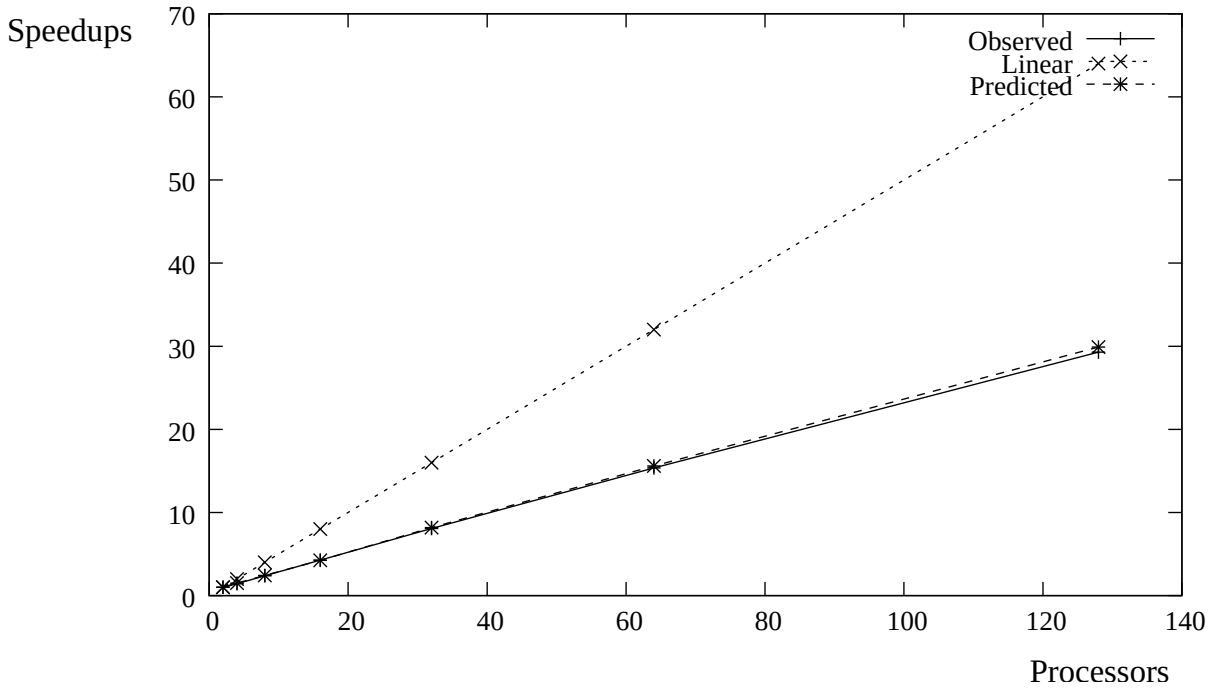
| Number | Description | Cost |
|---|---|---|
| 1 | Factorize the Diagonal Blocks | $O(n \times k^2)$ |
| 2 | Compute Tips of Spikes | $O(k^3)$ |
| 3 | Communicate Tips of Spikes | $O(k^2)$ |
| 4 | Factorize The Reduced System | $O(k^3)$ |
| 5 | Modify the Right Hand Side | $O(n \times k)$ |
| 6 | Communicate Tips of MRHS | $O(k)$ |
| 7 | Solve the Reduced System | $O(k^2)$ |
| 8 | Communicate Soln. of Reduced System | $O(k)$ |
| 9 | Matrix Vector Multiplication | $O(k^2)$ |
| 10 | Retrieve the Solution | $O(n \times k)$ |

# Scalability Case Study: Spike Solver Performance




Parametrizations of various steps for an IBM JS21.

# Scalability Case Study: Spike Solver Performance

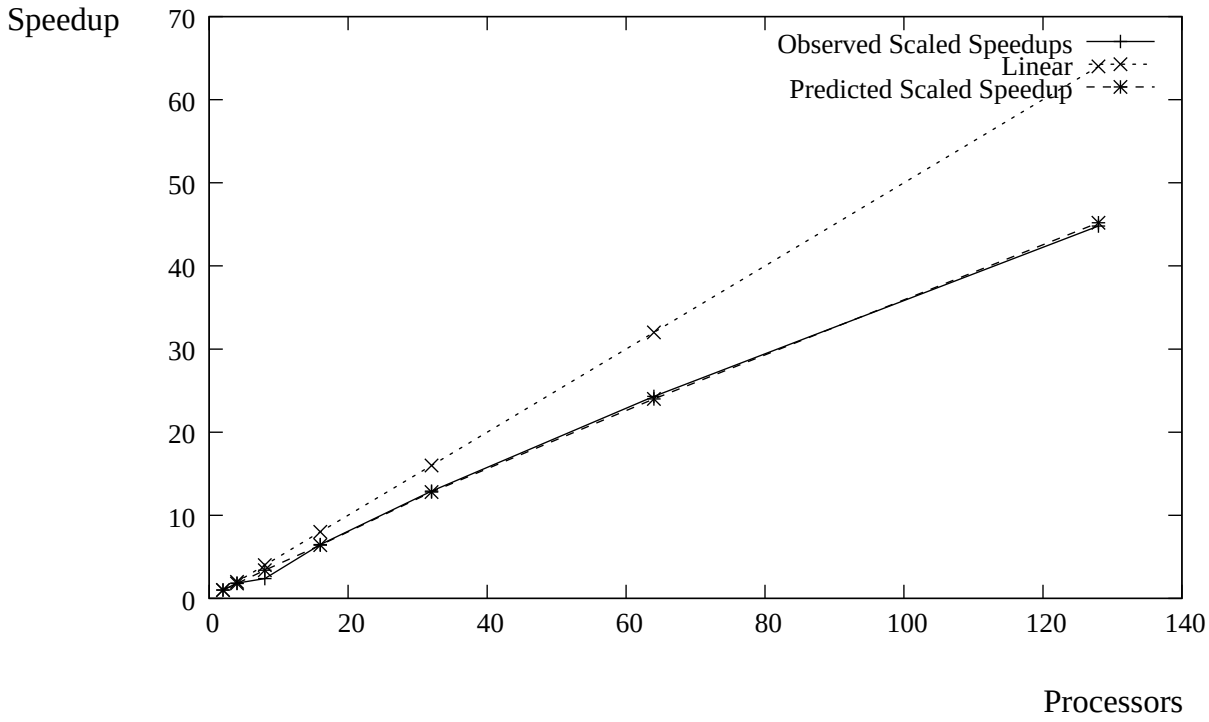



Observed and predicted speedups on the IBM JS21.

# Scalability Case Study: Spike Solver Performance

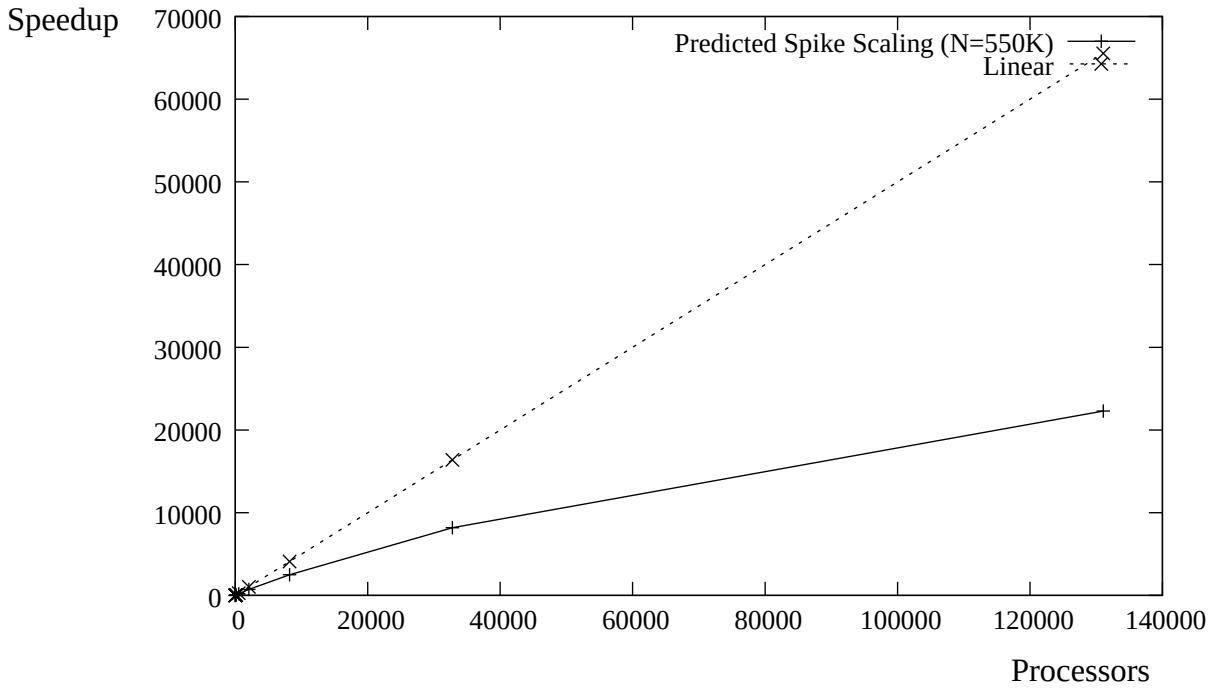
- Isoefficiency of the Spike solver is close to linear ($O(p)$).
- This implies that strong scaling should yield close to linear speedups.

# Scalability Case Study: Spike Solver Performance




Observed and predicted scaled speedups on the IBM JS21.

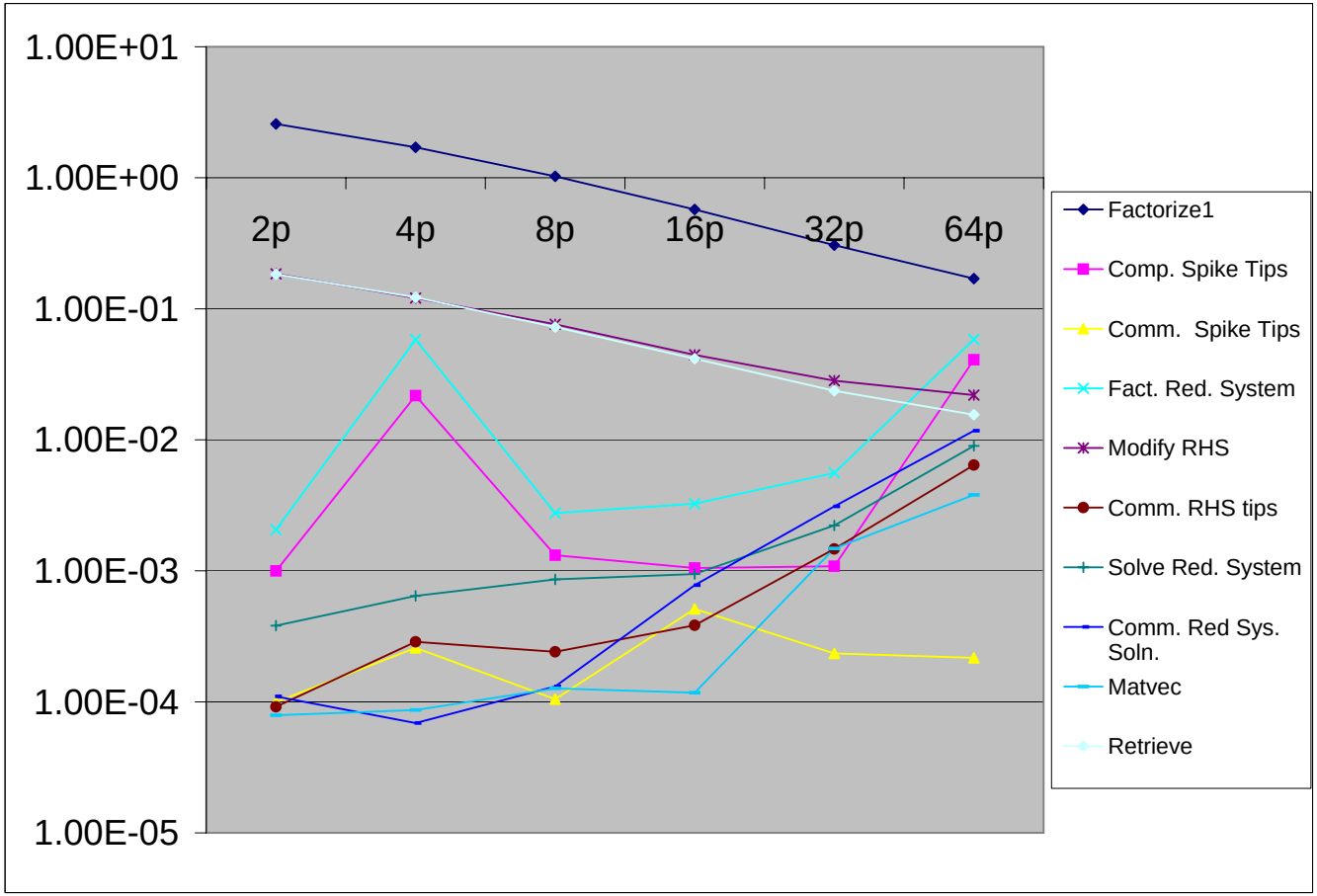
# Scalability Case Study: Spike Solver Performance




Predicted Spike Scaling to petaFLOPS onn the IBM JS21.

# Scalability Case Study: Spike Solver Performance

Shared address space nodes, though complicate life!




Parametrizations of various steps for an SGI Altix.

# Scalability Analysis: Conclusions

- Proposed and demonstrated a novel pseudoanalytical approach for scalability analysis.
- Technique factors application and architecture abstractions to enable portability studies.
- Application to Spike solver shows high degree of predictive accuracy.
- Demonstrated projected scaling of Spike to petascale.

# Scalability Analysis: Conclusions

- Complete study by analysing a broader class of platforms.
- Characterize other test-cases to fully validate scalability analysis methodology.

# Reference Slides

# Other Scalability Metrics

- A number of other metrics have been proposed, dictated by specific needs of applications.
- For real-time applications, the objective is to scale up a system to accomplish a task in a specified time bound.
- In memory constrained environments, metrics operate at the limit of memory and estimate performance under this problem growth rate.

# Other Scalability Metrics: Scaled Speedup

- Speedup obtained when the problem size is increased linearly with the number of processing elements.
- If scaled speedup is close to linear, the system is considered scalable.
- If the isoefficiency is near linear, scaled speedup curve is close to linear as well.
- If the aggregate memory grows linearly in $p$, scaled speedup increases problem size to fill memory.
- Alternately, the size of the problem is increased subject to an upper-bound on parallel execution time.

# Scaled Speedup: Example

- The serial runtime of multiplying a matrix of dimension $n \times n$ with a vector is $t_c n^2$.

- For a given parallel algorithm,

$$S = \frac{t_c n^2}{t_c \frac{n^2}{p} + t_s \log p + t_w n} \tag{6}$$

- Total memory requirement of this algorithm is $\Theta(n^2)$.

# Scaled Speedup: Example (continued)

Consider the case of memory-constrained scaling.

- We have $m = \Theta(n^2) = \Theta(p)$.
- Memory constrained scaled speedup is given by

$$S' = \frac{t_c c \times p}{t_c \frac{c \times p}{p} + t_s \log p + t_w \sqrt{c \times p}}$$

  or $S' = O(\sqrt{p})$.

- This is not a particularly scalable system.

# Scaled Speedup: Example (continued)

Consider the case of time-constrained scaling.

- We have $T_P = O(n^2/p)$.
- Since this is constrained to be constant, $n^2 = O(p)$.
- Note that in this case, time-constrained speedup is identical to memory constrained speedup.
- This is not surprising, since the memory and time complexity of the operation are identical.

# Scaled Speedup: Example

- The serial runtime of multiplying two matrices of dimension $n \times n$ is $t_c n^3$.
- The parallel runtime of a given algorithm is:

$$T_P = t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}$$

- The speedup $S$ is given by:

$$S = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}} \tag{7}$$

# Scaled Speedup: Example (continued)

Consider memory-constrained scaled speedup.

- We have memory complexity $m = \Theta(n^2) = \Theta(p)$, or $n^2 = c \times p$.
- At this growth rate, scaled speedup $S'$ is given by:

$$S' = \frac{t_c(c \times p)^{1.5}}{t_c\frac{(c \times p)^{1.5}}{p} + t_s \log p + 2t_w\frac{c \times p}{\sqrt{p}}} = O(p)$$

- Note that this is scalable.

# Scaled Speedup: Example (continued)

Consider time–constrained scaled speedup.

- We have $T_P = O(1) = O(n^3/p)$, or $n^3 = c \times p$.
- Time-constrained speedup $S''$ is given by:

$$S'' = \frac{t_c c \times p}{t_c \frac{c \times p}{p} + t_s \log p + 2t_w \frac{(c \times p)^{2/3}}{\sqrt{p}}} = O(p^{5/6})$$

- Memory constrained scaling yields better performance.

# Serial Fraction $f$

- If the serial runtime of a computation can be divided into a totally parallel and a totally serial component, we have:

$$W = T_{ser} + T_{par}.$$

- From this, we have,

$$T_P = T_{ser} + \frac{T_{par}}{p}.$$

$$T_P = T_{ser} + \frac{W - T_{ser}}{p} \tag{8}$$

# Serial Fraction $f$

- The serial fraction $f$ of a parallel program is defined as:

$$f = \frac{T_{ser}}{W}.$$

  Therefore, we have:

$$\begin{aligned} T_P &= f \times W + \frac{W - f \times W}{p} \\ \frac{T_P}{W} &= f + \frac{1 - f}{p} \end{aligned}$$

# Serial Fraction

- Since $S = W/T_P$, we have

$$\frac{1}{S} = f + \frac{1-f}{p}.$$

- From this, we have:

$$f = \frac{1/S - 1/p}{1 - 1/p}. \tag{9}$$

- If $f$ increases with the number of processors, this is an indicator of rising overhead, and thus an indicator of poor scalability.

# Serial Fraction: Example

Consider the problem of extimating the serial component of the matrix-vector product.

We have:

$$f = \frac{\frac{t_c \frac{n^2}{p} + t_s \log p + t_w n}{t_c n^2}}{1 - 1/p} \tag{10}$$

or

$$\begin{aligned} f &= \frac{t_s p \log p + t_w n p}{t_c n^2} \times \frac{1}{p-1} \\ f &\approx \frac{t_s \log p + t_w n}{t_c n^2} \end{aligned}$$

Here, the denominator is the serial runtime and the numerator is the overhead.