

- **Execution time** is basic and most important parameter in HPC
 - other measures based on execution time
- T_n ... execution time of a program on n nodes
- **Speedup** measures parallel execution time relative to sequential execution time
- $speedup(n) := T_1 / T_n$
 - ideal $speedup(n) = n$
 - *superlinear speedup possible: cache usage, thread scheduling, ...*
 - also termed *strong* scalability
- **Efficiency**: speedup per node
- $efficiency(n) := speedup(n) / n$, ideally 1

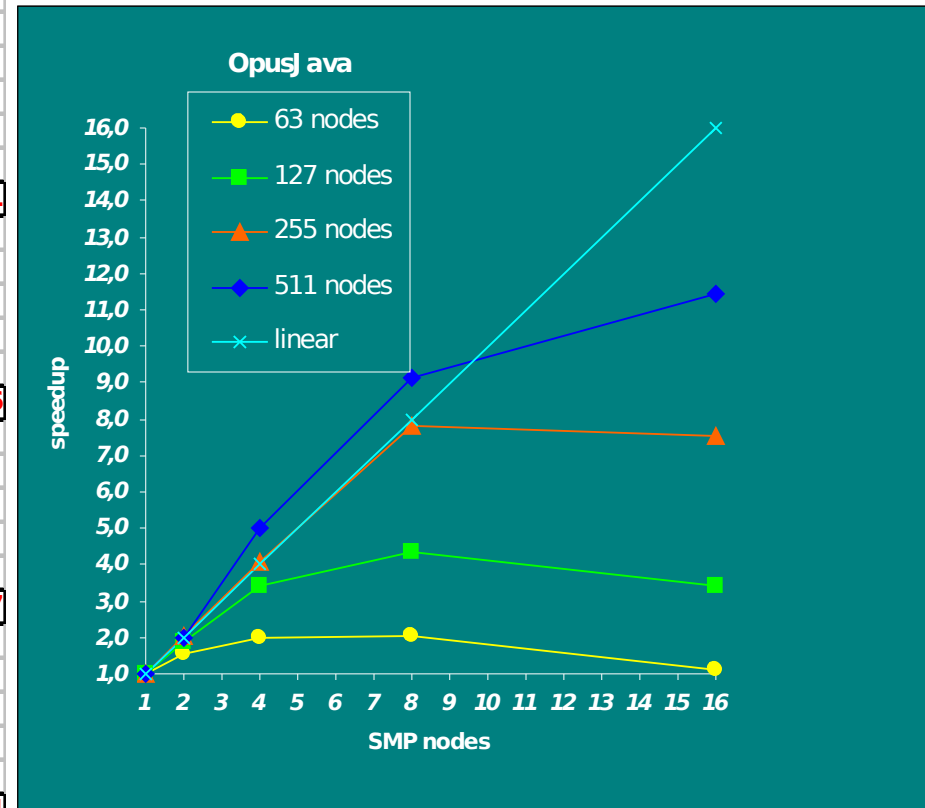
- Considering “problem size”
 - memory used, data size
 - more general: computational work, number of worker tasks
- T_n^N ... execution time for solving problem of size N on n nodes
- What is the effect of increasing problem size?
- ***growth_factor***(k) $:= T_n^{k*N} / T_n^N$
 - $n=1$: sequential growth
 - $n>1$: parallel growth

- What is the effect of simultaneously increasing (e.g., doubling) problem size N and number of nodes n ?
 - increase of n compensates for increase of N ?
- **scalability_factor**(k) := T_n^N / T_{k*n}^{k*N}
 - ideal *scalability_factor*(k) = 1
 - 0.8 is quite good
 - also termed *weak* scalability
- How does increase of problem size influence speedup ?
- **robustness**(n) := T_1^N / T_n^N
 - fixed relation $N/n=c$
 - e. g. ($c=64$), T_1^{128} / T_2^{128} , T_1^{256} / T_4^{256} , T_1^{512} / T_8^{512} , ...

Example: Stochastic Optimization



Tree Nodes	Compute Nodes	Execution Time	Speedup	Scalability
			Robustness	
63	1	742,00	1,00	1
63	2	476,00	1,56	
63	4	375,00	1,98	
63	8	359,00	2,07	
63	16	664,00	1,12	
127	1	2302,00	1,00	
127	2	1211,00	1,90	0,61271676
127	4	671,00	3,43	
127	8	532,00	4,33	
127	16	673,00	3,42	
255	1	6658,00	1,00	
255	2	3267,00	2,04	
255	4	1623,00	4,10	0,45717807
255	8	854,00	7,80	
255	16	885,00	7,52	
511	1	18099,00	1,00	
511	2	9182,00	1,97	
511	4	3596,00	5,03	
511	8	1979,00	9,15	0,37493684
511	16	1584,00	11,43	



- *Memory constrained* scaling
 - keep amount of memory used per processor constant
- Vs. **time constrained** scaling
 - keep total execution time constant, assuming perfect speedup
- $T_{k*n}^x = T_n^N$ (1)
 - how large is the scaled problem size x ?
 - requires time $T_n^N(N)$ to be known
 - e.g., $T_n^N = N^2$
- choose x s.t. $T_n^x = k * T_n^N$ (2)
 - e.g., $x^2 = k N^2$, $x = \text{sqrt}(k)*N$
- perfect speedup: $T_{k*n}^N = T_n^N / k$ (3)
 - $k * T_{k*n}^N \stackrel{(3)}{=} k * T_n^N / k \stackrel{(2)}{=} T_n^x / k \stackrel{(3)}{=} T_{k*n}^x = T_n^N$

- Measures not based on execution time may also be important in HPC
- Can provide insight into reasons for bad performance and potential performance losses
- **Work distribution**
- **Communication**
 - number of data transfers
 - amount of data transferred
- **Computation/communication ratio**
- **Cache misses**
- ...

- Scalability is (implicit) goal of performance optimization
- Minimize sequential code portions
 - cf. Amdahl's law: $speedup(n) = 1 / ((1-f) + f/n)$
 - f ... parallel code portion
 - e.g., $speedup(100)=80 \Rightarrow 1-f = 0.25\%$
 - e.g., $1-f = 20\% \Rightarrow maximal\ speedup = 5$
- Minimize communication
 - i.e., minimize communication/computation *ratio*
- Communication latency *hiding*
 - $comm_time = c_0 + c_1 * data_volume$

- Communication/computation *overlapping*
- Workload balancing: keep processors busy
- Granularity tradeoff
 - small granularity, high degree of parallelism, lot of communication
 - larger granularity, lower degree of parallelism, less communication
- Asynchronous communication
 - asynchronous algorithms
- Remove data dependencies
 - by transforming the sequential code
 - manually, (semi-)automatic

- For distributed memory systems, scalability is strongly determined by the degree of data locality
- *Data parallel programming*
 - partitioning of data domain
 - every processor executes the same program, but on different data („owner computes“-rule)
 - SPMD: Single Program Multiple Data
- *Data distribution* determines *work distribution*
 - can be derived by a parallelizing compiler
 - sequential program is annotated by *data distribution directives*

- Adaptation of HPC experiences to the TripCom scalability issue
- What can be re-used
- What must be modified
- ...