

# Chapter 11

# Metrics

## 2 Limits to parallel

How do we quantify the improvements achieved

By moving from single to distributed  
Moving from one form of distributed to another.

If a problem takes **T** seconds on a single processor.  
and **t** seconds on N processors

The speed up is **T/t** which we will call **S**  
If **T/t** equals N then we have ideal speed up.

Note prior to Amazon Cloud.  
If you had a job which took 5000 hours to run on one machine. Even with ideal speed up, it would cost you a good deal more to run it in 1 hour.

With EC2 – 1 machine for 1000 hours costs the same as 1000 machines for 1 hour.

### 3 Limits to parallel

.  
Ideal speed up implies that the number of instructions executed is the same in both scenarios (and that communication overheads are negligible).

One might assume that there is no limit to the value which **S** can take. This is not true, but worse.

Even if **S** increases without limit.  
**S/N** which is some sort of measure of efficiency may decrease.

Both money and power efficiency

**What limits speed up ?**

**SCALEABILITY**

**How can we measure it?**

.  
How much faster can N cores solve a problem compared with one?  
*How does the run time scale with core numbers*

How much more work can be done with N workers rather than one?

Consider a programme which has a “size”: finite element with n grid points if the size of the problem is doubled what happens to the runtime?

*How does the run time scale with problem size.*

Going from 100 cores to 200 cores may not halve runtime. (Speed up is rarely ideal.) Going from 1000000 to 2000000 grid points will increase the runtime by a factor of two. Doing both simultaneously and the runtime is unchanged (ideal) - it may well be closer to ideal than doubling the number of cores.

What is the effect of communications?

How efficiently are resource being used?

Limit on number of cores which lead to “good” speed up may depend on problem size.

.  
Simply model says  
 $\text{Problem Size} = \text{Parallel Fraction} + \text{serial Fraction}$

What are limits on parallelisability

*Algorithmic*: mutual dependencies - leading to an order of operations

*Bottlenecks*: shared resources. Shared memory paths in multicore chips; computer room wiring; wide area bandwidth. Access to database.

*Startup*: time to get all processes running.

*Communication*: slow and implies that both ends of the communication link must be present. Can write and then continue, but can only read when data is present. Except in special circumstances impossible, to ensure all reads precede writes.

Fixed size problem to be solved on **N** workers

Compare to single worker runtime

$$T_f^s = s + p$$

on N this becomes

$$T_f^p = s + p/N$$

This is called **Strong Scaling** amount of work is independent of number of processors.

Note subscript to distinguish between **f**(ixed) size and **v**(ariable)sized problems

We can scale the problem size with some power of N so size is  $s + pN^\alpha$  –  $\alpha$  is positive

Serial runtime  $T_v^s = s + pN^\alpha$   $\alpha$

and thus

Parallel runtime  $T_f^p = s + pN^{\alpha-1}$   $\beta$

This is referred to as **Weak Scaling**, although sometimes this is reserved for the special case where  $\alpha=1$

.  
**Application speedup** – quotient of parallel/serial execution time.

Serial performance  $P_f^s = (s + p) / T_f^s = 1$

Parallel

performance  $P_f^p = (s + p) / T_f^p(N) = 1 / (s + (1-s)/N)$

Serial speedup  $S_f = P_f^p / P_f^s = 1 / (s + (1-s)/N)$

**Amdahl's Law** Gene Amdahl 1967 – (fixed size)

We might reasonably ask what is the speed up for the parallel part of the calculation

Serial performance  $P_f^{sp} = p / T_f^s = p$

Parallel performance is

$$P_f^p = p / T_f^p(N) = (1-s) / (s + (1-s)/N)$$

Application speedup

$$S_f^p = P_f^{pp} / P_f^{sp} = 1 / (s + (1-s)/N)$$

as it must be.

Note as N goes to infinity  $S_f$  tends to  $1/s$ . Minimum runtime.

Performance is a factor p down  
If only a fraction f is parallelisable – only that can be affected

## 8 Simple scalability

Parallel performance  $P_f^p$  is now not the same as application speed up  $S_f^p$

So we have two different concepts we can play with in order to look at more complex ideas

We ask a question not from the '70s  
how much faster can I run my programme

We ask one from the 90's  
how much more accurate can I make my  
programme and have it finish in the same time?

Weather forecasting is an obvious place this is important.

How much more work can my program do in given time T when I put a larger problem on N cpus.

Serial performance  $P_v^s = (s + p) / T_f^s = 1$

$P_v^p = (\text{serial run time}) / (\text{parallel run time})$   
 $P_v^p = (s + pN^\alpha) / T_v^p(N) = (s + (1-s) N^\alpha) / (s + (1-s)N^{\alpha-1}) = S_v$

So parallel performance is identical to application speed up. In this case  $\alpha=0$  (string scaling) we recover Amdahl's Law.  
 $0 < \alpha < 1$  and for large N

$$S_v = (s + (1-s) N^\alpha) / s = 1 + pN^\alpha / s$$

In this case S increase without limit (although slowly) if p is small

For  $\alpha=1$ , the ideal case

$$S_v = s + (1-s)N$$

**Gustafson's Law**

We get a speed up which is linear in N – but if s is large then 1-s is small and the speed up increases only slowly with N. Efficiency falls

Now using subscript v for variable size problem

On slide 6  $\alpha, \beta$

Apparently beating the Amdahl limit

For a long time Amdahl's law was considered to be the final word on parallelisability.

It apparently set an absolute upper limit on parallel performance.

But it depends on the question you ask  
Amdahl was positing a fixed problem size, taking the execution time on a single processor as the baseline and then asking how much improvement was possible.

*A reasonable point of view in 1967*

By 1988 Gustafson pointed out that in many instances the size of problem was set by the solution time. That as the resources available become larger that problems considered soluble grow.

If we have a solution where the number of calculations grows as you add processors, as long as this growth is not too fast you can win.  
Amdahl says more performance means less time to do the same thing. Gustafson emphasises the ability to do more things in the same time.

11 Efficiency

So speed up can increase without limit, if we measure it in a particular (reasonable) way.

What about efficiency? In other words how much of the money we spend on electricity is useful.

Efficiency  $\varepsilon = \frac{\text{Performance on N CPUs}}{(N \times \text{performance on 1 CPU})}$

= speed up / N      speed up compared to ideal

Consider weak scaling

$$\varepsilon = \frac{S_v}{N} = \frac{sN^{-\alpha} + (1-s)}{sN^{1-\alpha} + (1-s)}$$

For  $\alpha = 0$        $\varepsilon = 1 / (sN + (1-s))$   
and as expected  $\varepsilon$  tends to zero for large N

For  $\alpha = 1$        $\varepsilon = (1-s) = p$  for large N. We are using some of the CPUs even when N is large, but wasted time grows linearly with N

But if we use the definition of work =  $pN^\alpha$

$$\varepsilon_p = \frac{S_v^p}{N} = \frac{N^{\alpha-1}}{s + (1-s) N^{\alpha-1}} = 1$$

If floating point operations only occur in the parallel part of the programme then MFLOPS / CPU will scale

Again look at only the parallel fraction

$$P_v^{sp} = p$$
$$P_v^{pp} = pN^\alpha / T_v^p(N) = (1-s)N^\alpha / (s + (1-s)N^{\alpha-1})$$

$$S_v^p = P_v^{pp} / P_v^{sp} = N^\alpha / (s + (1-s)N^{\alpha-1})$$

Once again speed up and performance are not identical.

For  $\alpha = 1$  application speed up becomes linear in  $N$  – slope = 1.

So it looks as if all is OK

if your metric is based on a number which is only relevant to parallel part of the calculation.

“Number of lattice sites updated”

**But** what you are paying for is CPU cycles, and this may be very misleading if it leads you to believe a calculation is “value for money”.

Need to look at efficiency.

Unsurprising

Metrics

### 13 More complex models

It is always attractive to add complexity to a model which is known not to accurately describe all parts of the system.

A more complex model has more factors which may interact, it is harder to tell what is affecting the final answer.

More complex models are harder to validate.

Models are (normally) used to predict behaviour away from measured values – or to try to improve behaviour in a measured regime.

Improvements normally provide their own validation. If the model suggests a course of action and following that course of action is successful you are happy.

Predictions are often used to guide a course of action where the investment (both financial and in time) is made before the prediction can be validated.

In this case validating the model is important – don't over complicate.

Add some communication overheads  
Assume communications and calculations cannot be overlapped.

$$T_v^{pc} = s + pN^{\alpha-1} + c_\alpha(N)$$

$$S_v^c = s + \frac{pN^\alpha}{T_v^{pc}(N)} = \frac{s + (1-s) N^\alpha}{s + (1-s) N^{\alpha-1} + c_\alpha(N)}$$

The problem then is to find an expression for  $c_\alpha(N)$

- might not be possible to write it in closed form. (See Lecture on RGMA)
- The same for all nodes?
- Consists of latency  $\lambda$  plus transfer time  $\kappa = n/B$ . With  $n$  the message size and  $B$  the bandwidth

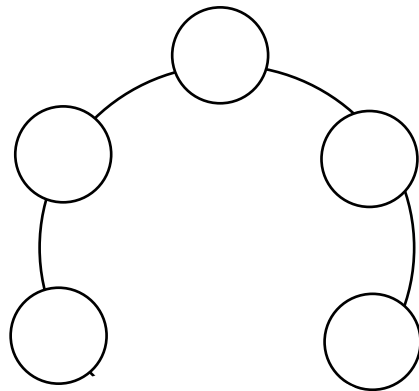
## 15 Examples (i)

•  $\alpha = 0$ : *blocking network*. eg bus – only 1 message at a time. Communication overhead is independent of  $N$

$$\begin{aligned}c_{\alpha}(N) &= (\kappa + \lambda)/N \\ S_v^c &= 1 / (s + (1-s)/N + (\kappa + \lambda)N) \\ &= 1 / (\kappa + \lambda)N \quad \text{for large } N\end{aligned}$$

Performance is dominated by communications – not surprising with only one channel.

### **BUS**

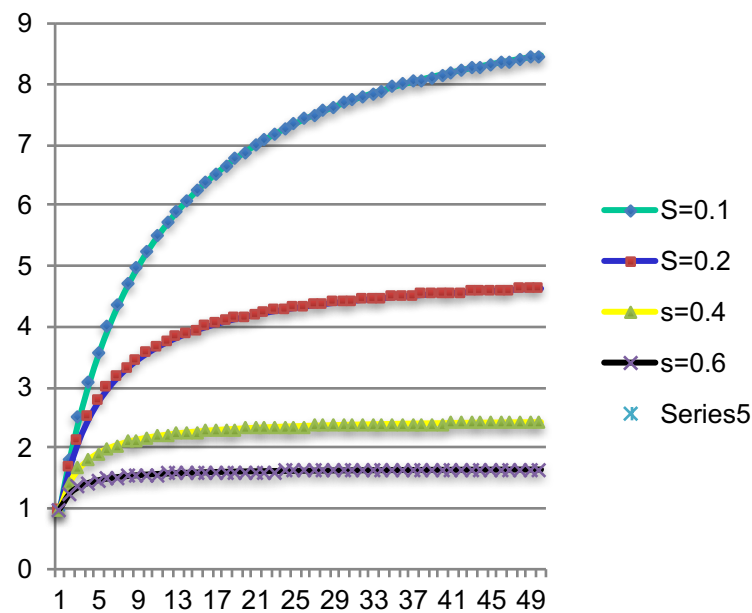


Single data path - shared

• $\alpha = 0$ : *non-blocking network, constant cost* (so no distance cost). Assuming infrastructure allows  $N/2$  messages with no collisions.  $c_\alpha(N) = (\kappa + \lambda)$

$$S_v^c = \frac{1}{s + (1-s)/N + (\kappa + \lambda)}$$
$$= \frac{1}{s + \kappa + \lambda} \quad \text{for large } N$$

Looks like Amdahl – so saturates - but at a lower number of nodes.



Improvement stops at finite N

•  $\alpha = 0$ : *non-blocking network domain decomposition with ghost layer*. Overhead decreases with  $N$ .  $c_\alpha(N) = \kappa N^{-\beta} + \lambda$ . For  $\beta > 0$  at large  $N$ , performance depends on  $s$  and  $\lambda$ .

$$S_v^c = 1 / (s + (1-s)/N + \kappa N^{-\beta} + \lambda)$$

$$= 1 / (s + \lambda) \quad \text{for large } N$$

Again looks like Amdahl's Law – but now it is just the latency that causes the improvement to saturate earlier.

The question is now if we see  $1/n$  behaviour what is it coming from  $s$ ,  $\kappa$  or  $\lambda$

If we try to measure  $S_v^c$  for a number of values of  $N$ , we can easily fit for the total  $1/n$  behaviour, but distributing it between the sources is much harder. Measurements will have uncertainties on them. Inhomogeneity in the system may cause any of these to vary with  $N$

We are looking for a system: problem, algorithm, architecture that scales.

Suppose we have a problem – where should we look?

One place is load imbalance  
nodes with more work finish late. Other processors are left idle. Underutilisation. May come from:

1. Inefficient algorithm
2. Bad optimisation.
3. Distribution algorithm, may not be compatible with solution algorithm
4. Work required “per chunk” unknown at compile time
5. Insufficient parallel tasks.
6. Interference from the OS

Operating systems need to perform “house-keeping” chores to keep the system running. This takes time away from user tasks and may create a task which lags behind the others.

Small number of cores infrequent and of little consequence.

Large number of cores, may happen every time to one core. Waste can be significant.

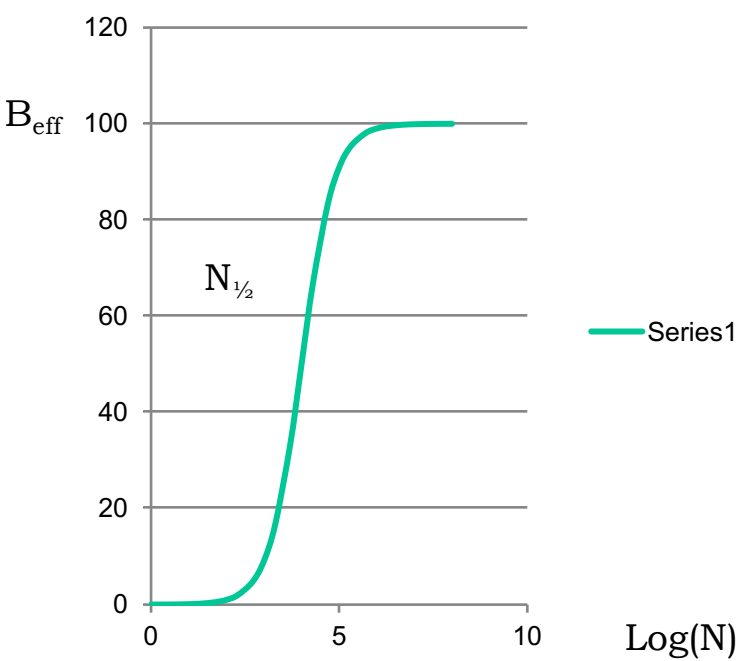
Research on, for instance synchronising house-keeping across a number of cores so that the pause happens to all cores simultaneously and loss is negligible. Not available as standard.

Reducing frequency of synchronisation points so most of the cores see one OS interruption per synchronisation. All arrive late and no waiting.

Bandwidth is the number of bits/s which can be transferred.

Transfer time is latency  $\lambda$  plus transfer time  $\kappa = n/B$

So the effective bandwidth =  $n/(\lambda + n/B)$



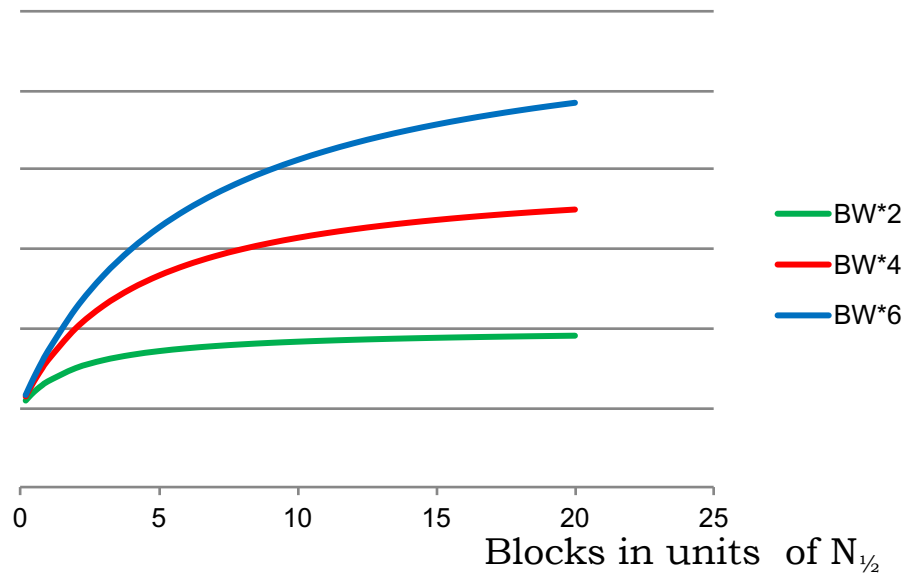
$\lambda = 100\mu\text{s}$   $\kappa = 120 \text{ Mbytes/s}$   
Note max is not quite 120

$N_{1/2}$  is blocksize  
when  $B_{\text{eff}} = \kappa/2$

Note logarithmic scale

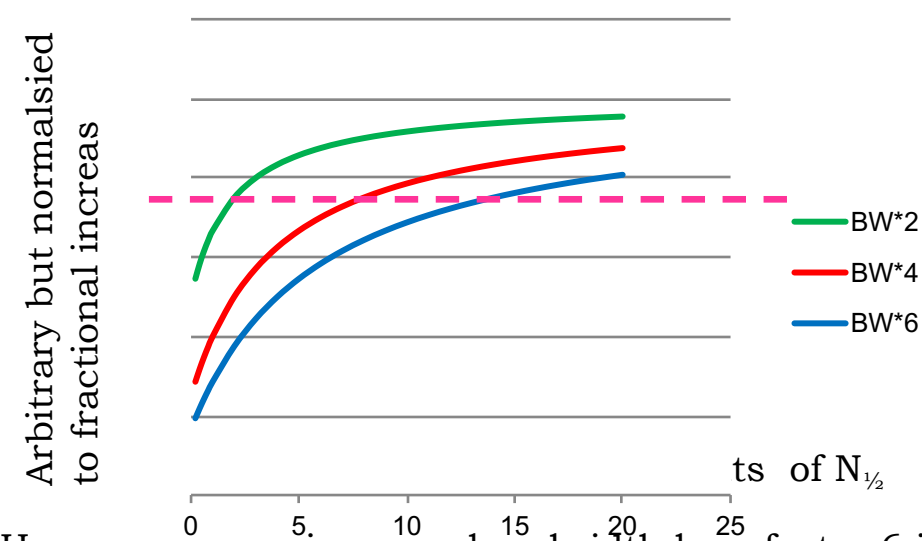
$\kappa_{\text{eff}} = \kappa/2 = N_{1/2} / (\lambda + N_{1/2}/\kappa)$   
 $N_{1/2} = \lambda\kappa$  involves latency and bandwidth

$\kappa_{\text{eff}} = 1 + N / N_{1/2}$   
If we increase the bandwidth by a factor  $f$  – then  
 $\kappa_{\text{eff}} = 1 + N / f \cdot N_{1/2}$  and we can look at the increased throughput seen by increasing the bandwidth by some factor.



When blocks are small, effective bandwidth is dominated by latency and increasing bandwidth is unprofitable. As blocks get bigger the investment is worth while - **but**

To see better what is the “value for money” normalise each curve by improvement.



Here we can see increase bandwidth by a factor 6 is only 1/3 as good as increasing by 2, at low blocksize. As bandwidth increases it needs a larger blocksize to reach the same value for money.

Finally many systems have a maximum blocksize so there is a limit to how far right you can move up the curve.