CS6230

Performance Evaluation of Codes

Performance Metrics

Aim – to understanding the algorithmic issues in obtaining high performance from large scale parallel computers

Topics for Conisderation

- General speedup formula
- Revisiting Amdahl’s Law
- Gustafson-Barsis’ Law
- Karp-Flatt metric
- Isoefficiency metric
- Isotime and Isomemory metrics

Speedup Formula

Speedup $S(p) = t_s \,/\, t_p$

Execution Time Components

Inherently sequential computations: *Ser(n)*

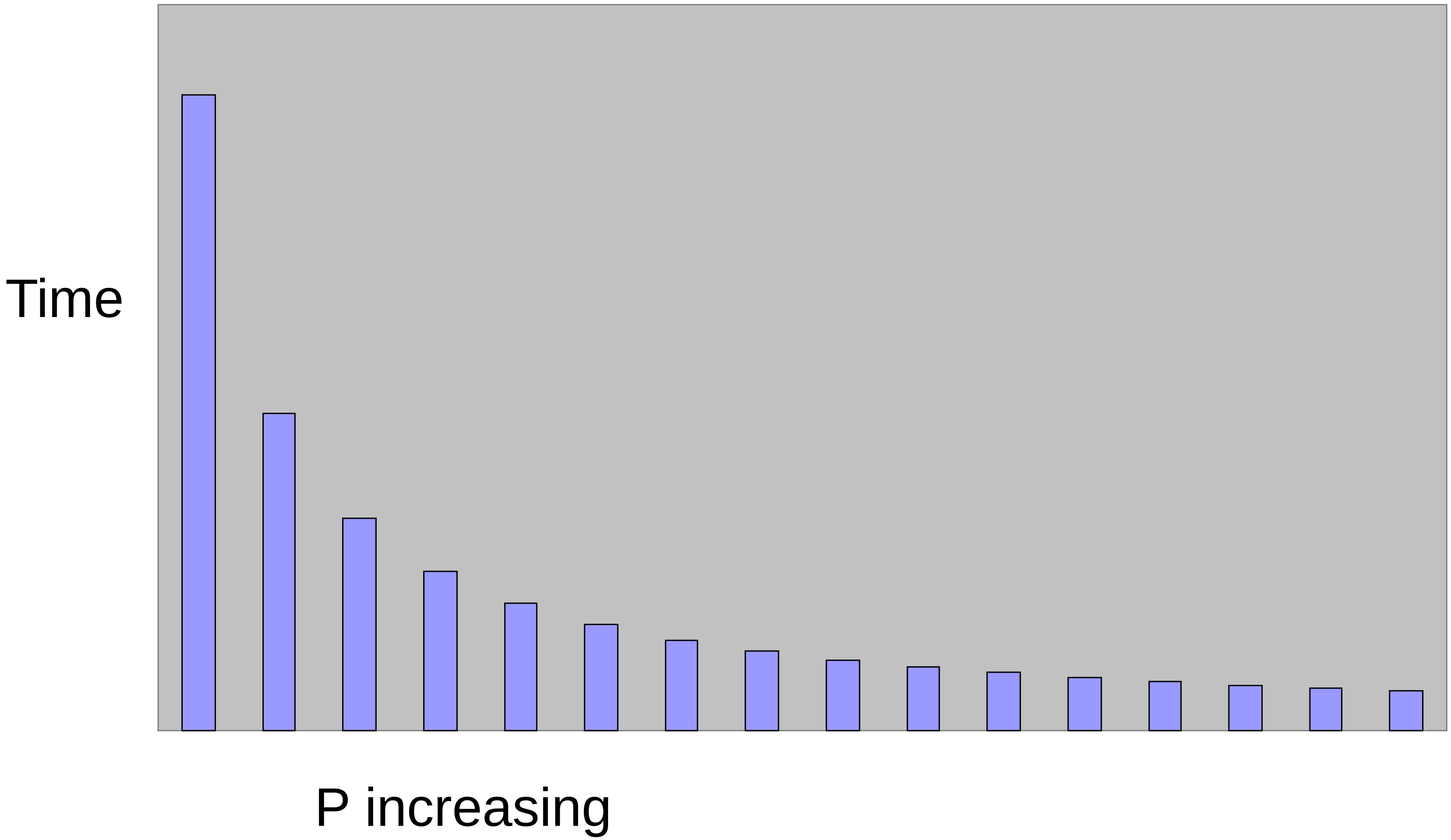
Potentially parallel computations: *Par(n)*

Communication operations: *Com(n,p)*

$$S(p) = \frac{Ser(n) + Par(n)}{Ser(n) + Par(n)/p + Com(n,p)}$$

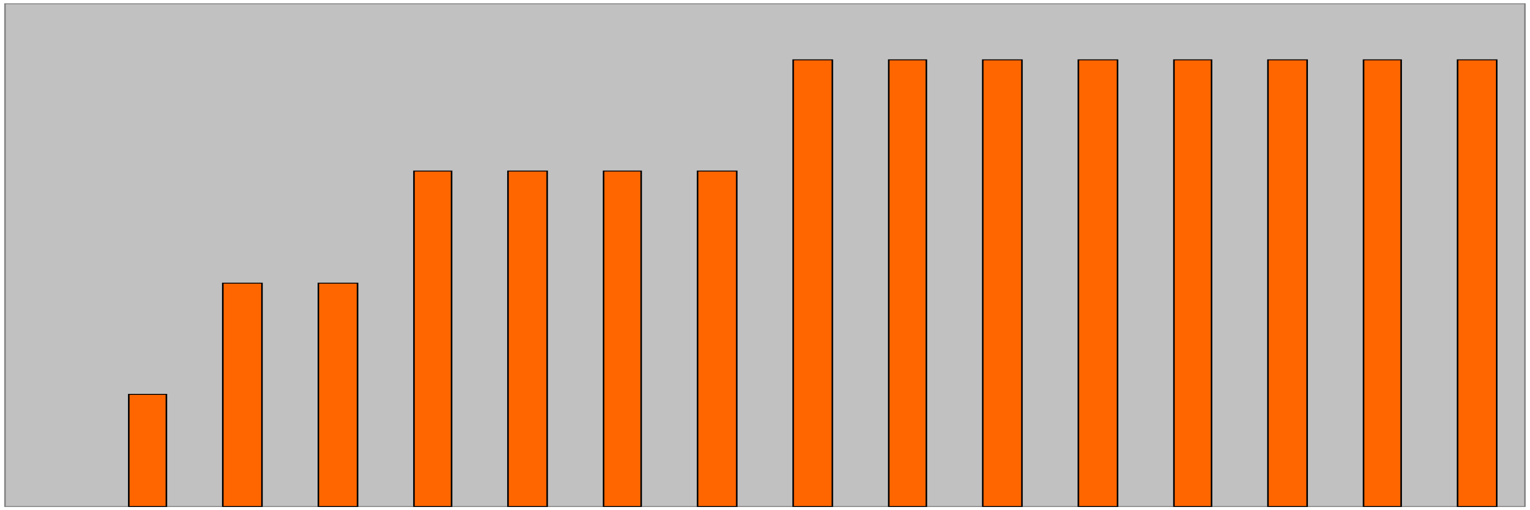
Example

Let *par(n)/p be*



And let communications be *com(n,p)*

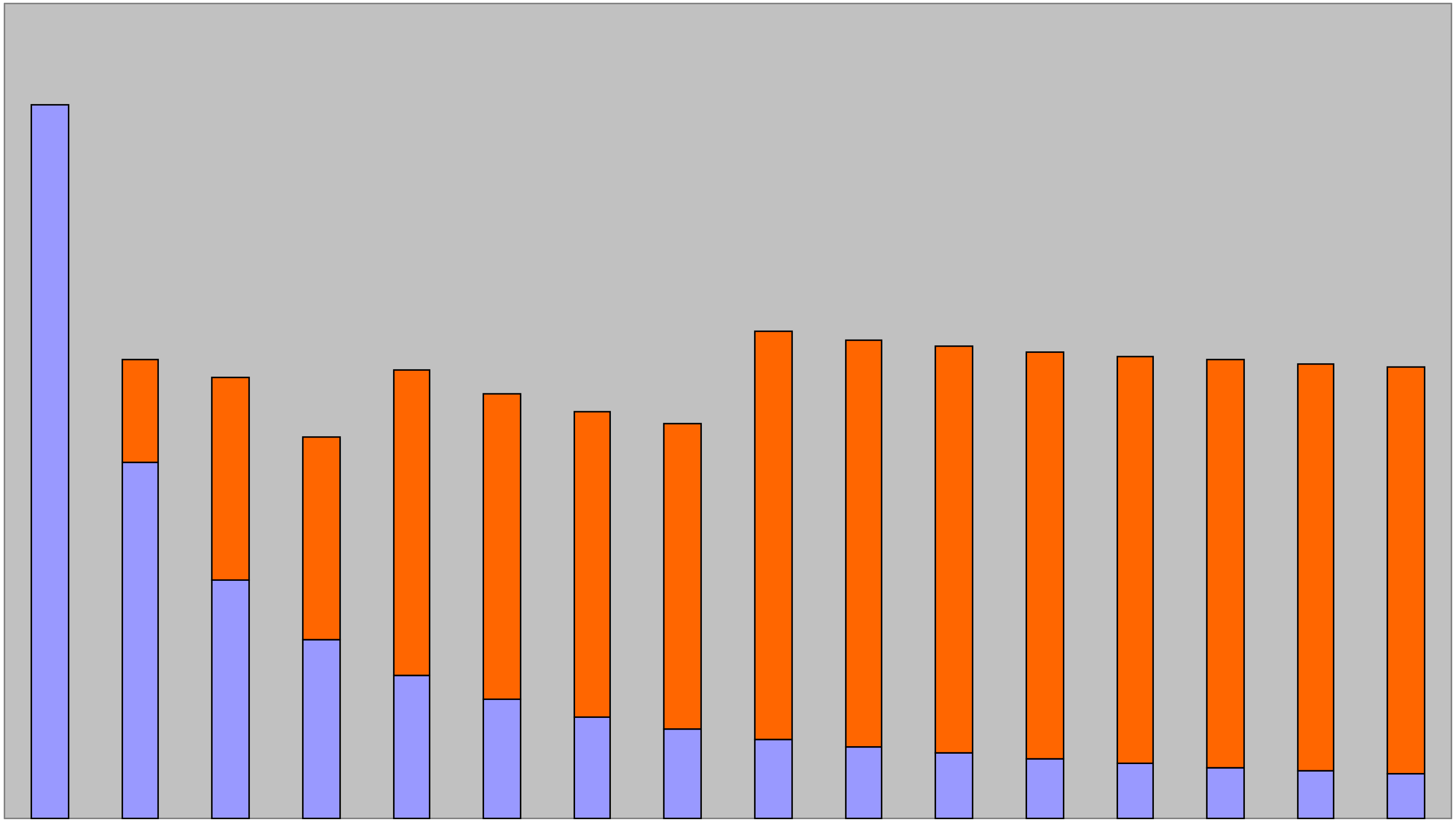
Time



Processors

Then $\mathrm{par}(n)/p + \mathrm{com}(n,p)$

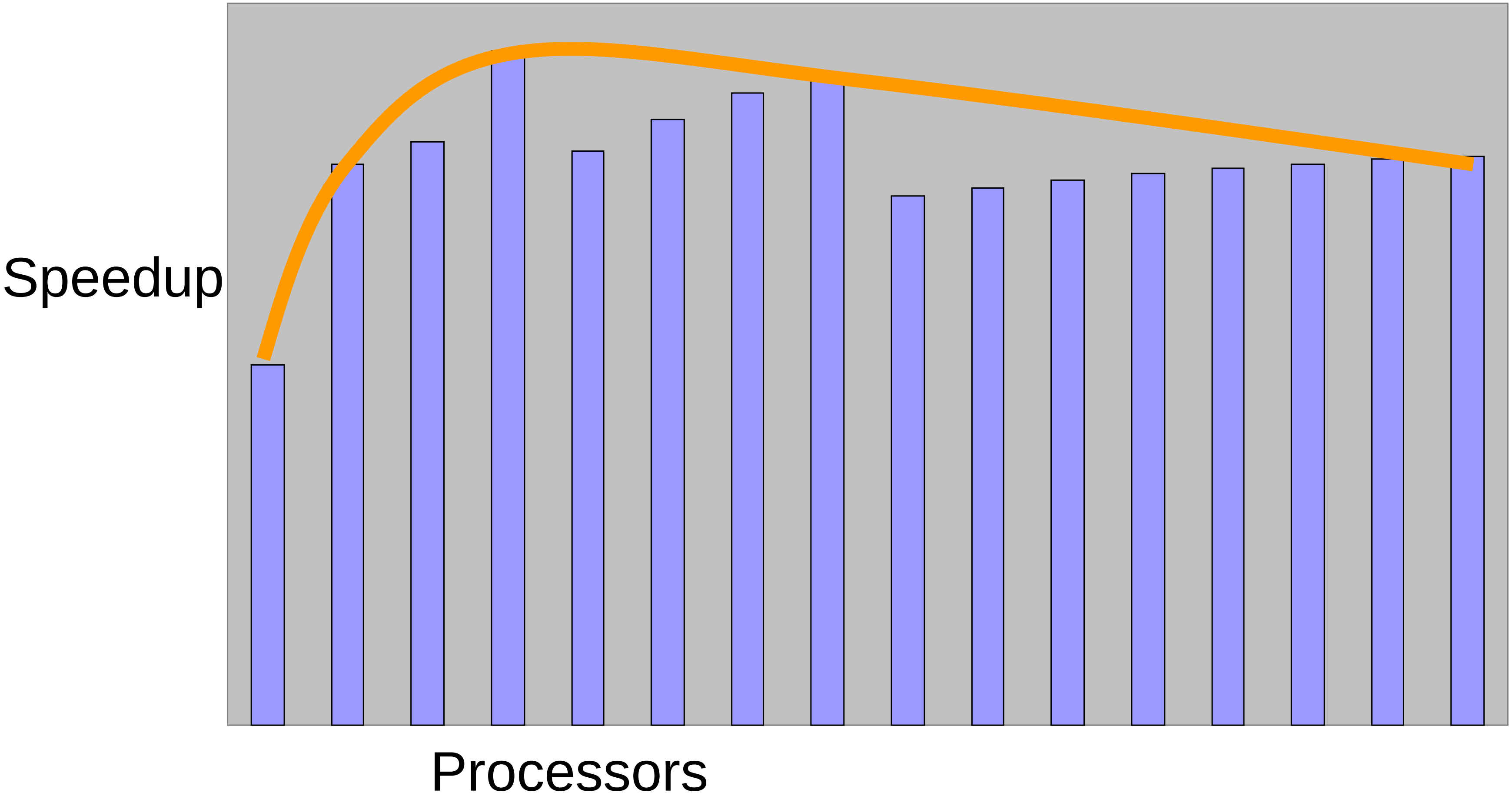
Time



Processors

Speedup Plot

Speedup increases then flattens out and decreases



Efficiency, E(n,p)

$$\text{Efficiency} = \frac{\text{Sequential execution time}}{\text{Processors } x \text{ Parallel execution time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

Efficiency is a fraction:

$$0 \leq E(n,p) \leq 1$$

Amdahl's Law

Ignore communications part of efficiency to get upper bound

$$S(n,p) \le \frac{Ser(n) + Par(n)}{Ser(n) + Par(n)/p + com(n,p)}$$

$$\le \frac{Ser(n) + Par(n)}{Ser(n) + Par(n)/p}$$

Let f = Ser(n)/(Ser(n) + Par(n)); i.e., f is the fraction of the code which is inherently sequential

$$S(n,p) \le \frac{1}{f + (1-f)/p}$$

Example 1

- 95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$S(n, p) \leq \frac{1}{0.05 + (1 - 0.05) / 8} \cong 5.9$$

Example 2

- 20% of a program's execution time is spent within inherently sequential code. What is the limit to the speedup achievable by a parallel version of the program?

$$\lim_{p \to \infty} \frac{1}{0.2 + (1 - 0.2) / p} = \frac{1}{0.2} = 5$$

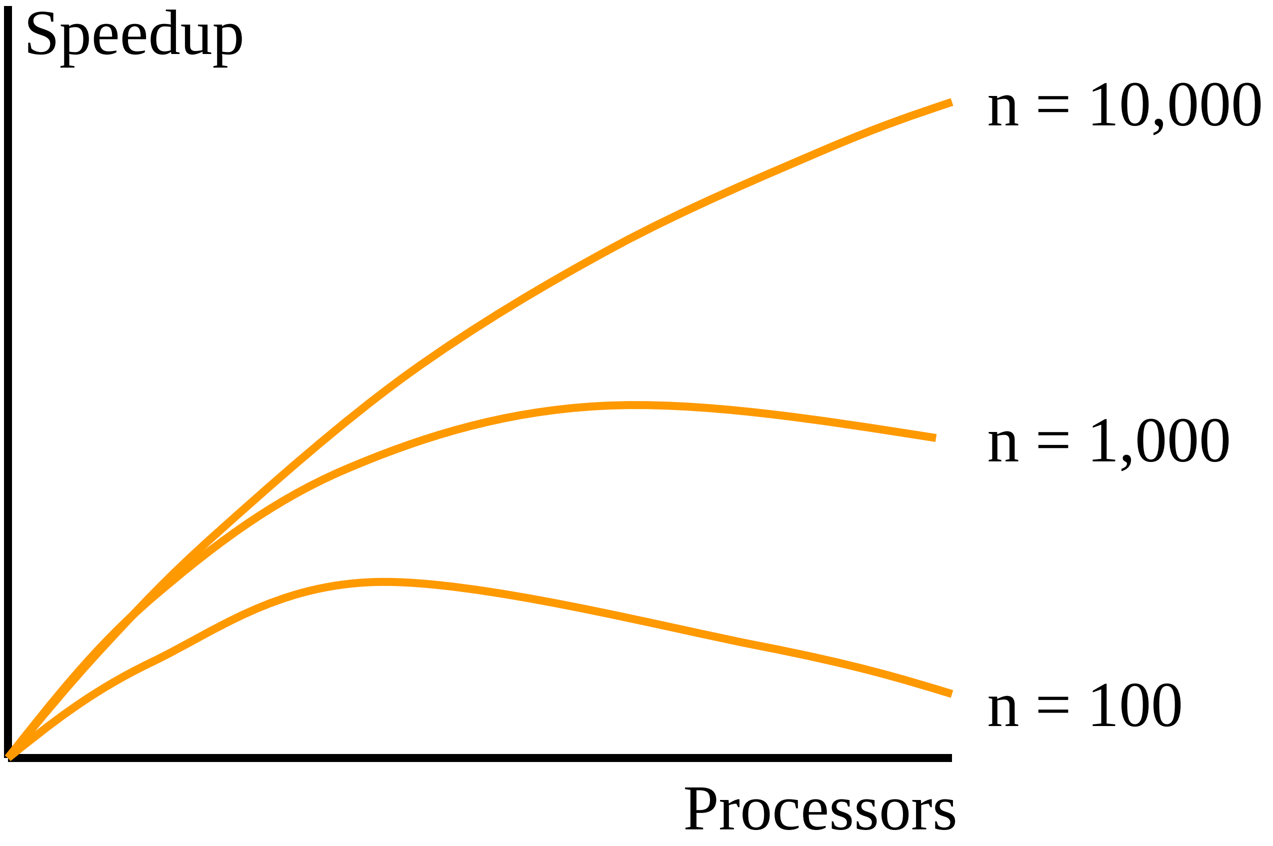
Limitations of Amdahl's Law

- Ignores *Com*(*n*,*p*) - overestimates speedup
- Assumes f serial fraction constant, so underestimates speedup achievable
- Often *Ser*(*n*) and *Com*(*n*,*p*) have lower complexity than *Par*(*n*)/*p*
- As *n* increases, *Par*(*n*)/*p* dominates *Ser*(*n*) & *Com*(*n*,*p*)
- *As* ***n increases, speedup increases***
- *As* ***n increases, sequential fraction f decreases.***

Illustration of Amdahl Effect

Treats problem size as a constant

Shows how execution time eventually decreases as number of processors increases



Gustafson-Barsis's Law

- We often use faster computers to solve larger problem instances
- In such cases the amount of algorithmic overhead is fixed
- Hence allow problem size to increase with number of processors

Gustafson-Barsis's Law

$$S(n,p) \le \frac{Ser(n) + Par(n)}{Ser(n) + Par(n) / p}$$

Let $T_p = Ser(n)+Par(n)/p = 1$ unit

Let *s* be the fraction of *time* that a parallel program spends executing the serial portion of the **parallel** code.

$$s = Ser(n)/(Ser(n)+Par(n)/p)$$

Then,

$$S(n,p) \le T_1/T_p = s + p*(1-s)$$ (the *scaled speedup*)

Thus, sequential time would be p times the parallelized portion of the code plus the time for the sequential portion. Rearranging the above equation gives

$$S(n,p) \le p + (1-p)s$$

We assume that **s** may be small

Gustafson-Barsis's Law

- Begin with parallel execution time and estimate the time spent in sequential portion.
- Predicts **scaled speedup bound on speedup**
- Estimate sequential execution time to solve same problem (s)
- Assumes that s remains fixed irrespective of how large is p - thus overestimates speedup.
- Problem size (s + p*(1-s)) is an increasing function of *p*

Example 1

- An application running on 10 processors spends 3% of its time in serial code. What is the scaled speedup of the application?

$$S = 10 + (1 - 10)(0.03) = 10 - 0.27 = 9.73$$

…except 9 do not have to execute serial code

Execution on 1 CPU takes 10 times as long…

Example 2

- What is the maximum fraction of a program's parallel execution time that can be spent in serial code if it is to achieve a scaled speedup of 7 on 8 processors?

$$7 = 8 + (1-8)s \Rightarrow s \approx 0.14$$

Pop Quiz

- A parallel program executing on 32 processors spends 5% of its time in sequential code. What is the scaled speedup of this program?

The Karp-Flatt Metric

- Amdahl's Law and Gustafson-Barsis' Law ignore $Com(n,p)$
- They can overestimate speedup or scaled speedup
- Karp and Flatt proposed another metric
- Their metric enables serial overhead to be measured

Experimentally Determined Serial Fraction, f, Karp-Flatt Metric

$$f = \frac{Ser(n)}{Ser(n) + Par(n)} = \frac{\text{Inherently serial component of parallel computation}}{\text{Single processor execution time}}$$

From Amdahl's Law

$$S(n,p) = \frac{1}{f + (1-f)/p}$$

And so

$$f = \frac{1/S(n,p) - 1/p}{1 - 1/p}$$

Derivation – difficult in Wikipedia, and [Quinn]
Easier in original paper

Experimentally Determined Serial Fraction, f, Karp-Flatt Metric Derivation

From Amdahl's Law

$$S(n,p) = \frac{1}{f + (1-f)/p}$$

and so

$$\frac{1}{S(n,p)} = f + \frac{1}{p} - \frac{f}{p}$$

$$\frac{1}{S(n,p)} - \frac{1}{p} = f\left(1 - \frac{1}{p}\right)$$

And so

$$f = \frac{1/S(n,p) - 1/p}{1 - 1/p}$$

As required

Experimentally Determined Serial Fraction

- Takes into account parallel overhead
- Detects other sources of overhead or inefficiency ignored in speedup model
  - Process startup time
  - Process synchronization time
  - Imbalanced workload
  - Architectural overhead

Example 1

p	2	3	4	5	6	7	8
S	1.8	2.5	3.1	3.6	4.0	4.4	4.7

What is the primary reason for speedup of only 4.7 on 8 CPUs?

f	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Since f is constant, large serial fraction is the primary reason.

Example 2

p	2	3	4	5	6	7	8
S	1.9	2.6	3.2	3.7	4.1	4.5	4.7

What is the primary reason for speedup of only 4.7 on 8 CPUs?

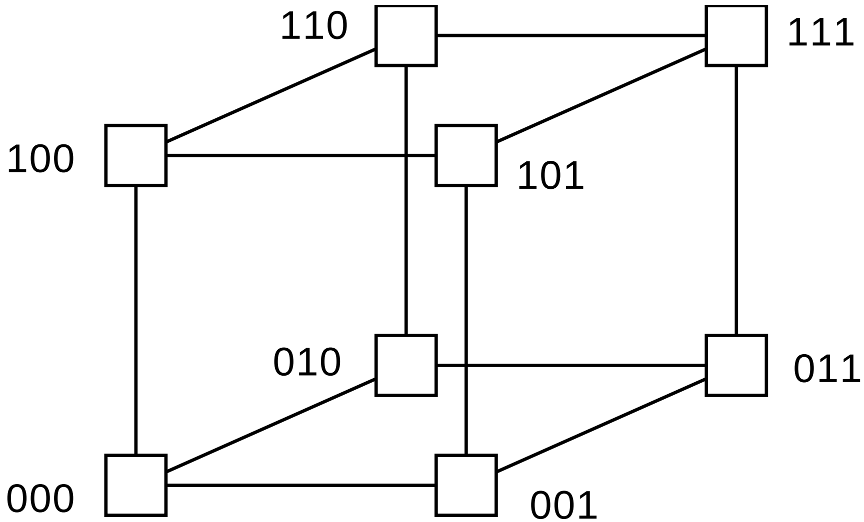
f	0.070	0.075	0.080	0.085	0.090	0.095	0.100

Since f is steadily increasing, overhead is the primary reason.

Isoefficiency Metric

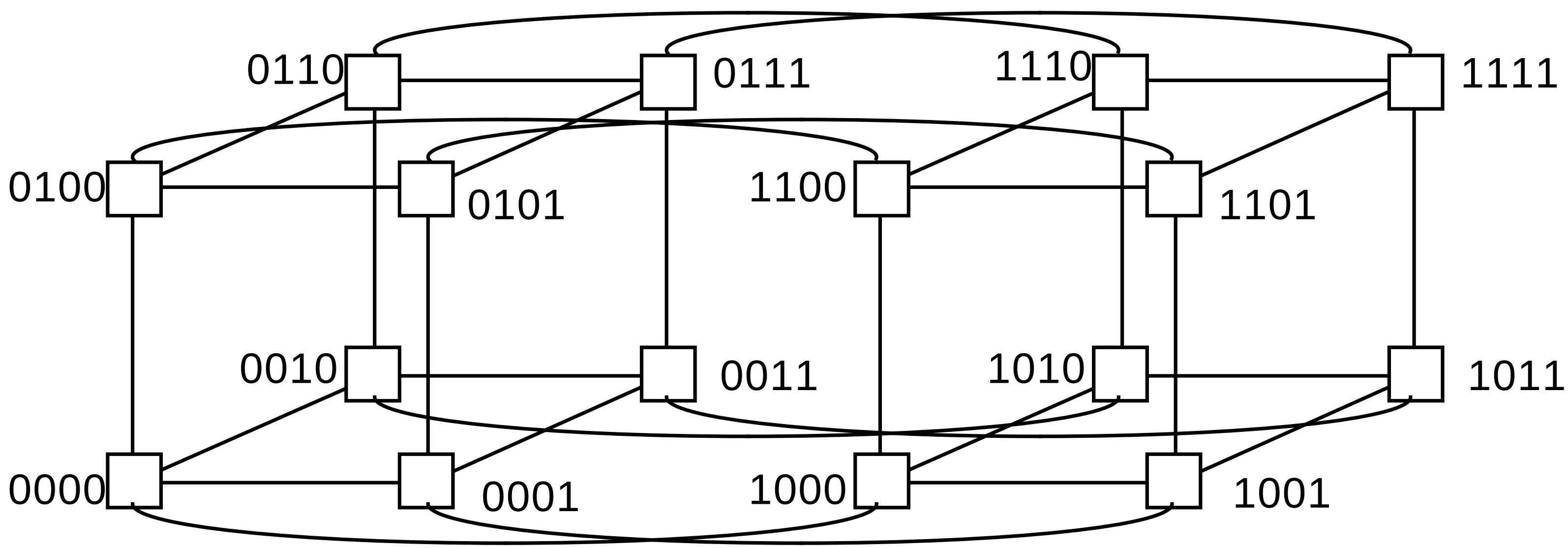
- Parallel system: parallel program executing on a parallel computer
- Scalability of a parallel system: measure of its ability to increase performance as number of processors increases
- A scalable system maintains efficiency as processors are added
- Isoefficiency: way to measure scalability

Adding numbers on a hypercube



3D Hypercube is 2 2D Hypercubes joined together

4D hypercube is 2 3D hypercubes joined together3D



Method A - one number on each processor

Algorithm form partial sums at each level (dimension) of Hypercube

- each step is one add + communicate

E.g n = 16 and so 4 = $log(n)$ steps

Parallel time = const $\times log(n)$

Speed up S(n) = *serial time/parallel time* = $n/log(n)$

Efficiency = E = $S(n)/n$ = $1/log(n)$

Hence for n = 32 $log(n)$ = 5 hence E = 1/5

The machine is only being used at 20% efficiency.

Example:

Adding *n* Numbers on a *p* processor hypercube, $n > p$

Method B – n/p numbers summed on each proc. – then p partial sums in $log(p)$ steps

E.g $n = 16$, $p = 4$ and so $2 = log(4)$ steps are used

Parallel time = const $\times log2\ p + n/p$ adds

Speed up S(n) = *serial time/parallel time* = $n/ (log(p)+n/p)$

Efficiency = E = $(n/p) / (log(n)+n/p) = 1 / (1+p\ log(p)/n)$

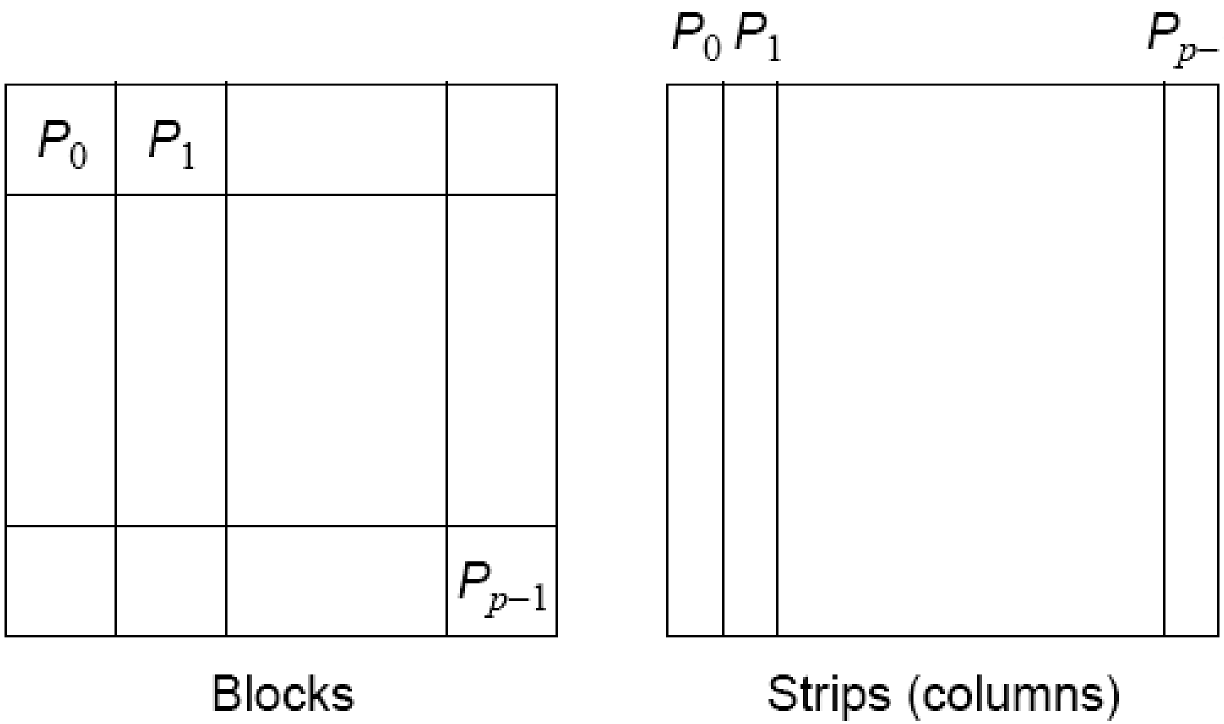
Hence for p = 32 $n = 32\times100$ $\boldsymbol{S(n)} = 32/ (1+5\times0.001)$

Efficiency = E = $1/ (1+5\times10-2)$ giving 95% efficiency

Laplace’s equation Partitioning

Normally allocate more than one point to each processor, because many more points than processors.

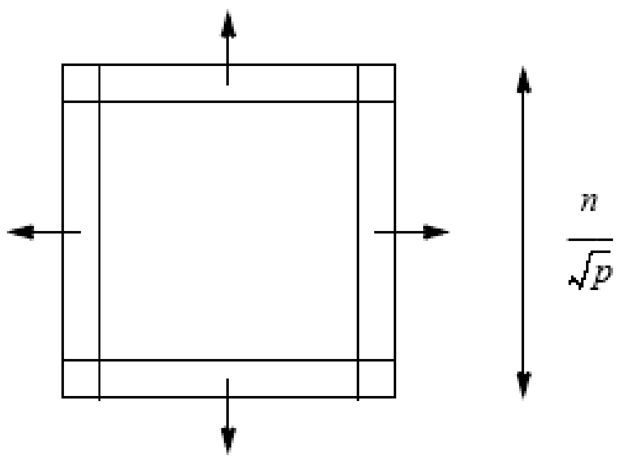
Points could be partitioned into square blocks or strips:



Block partition

Four edges where data points exchanged.
Communication time given by

$$t_{\text{commsq}} = 8\left(t_{\text{startup}} + \frac{n}{\sqrt{p}}t_{\text{data}}\right)$$

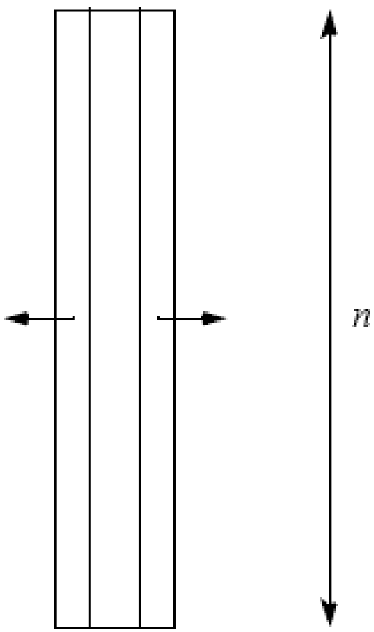


Square blocks

Total cost = 7×(*n***2/*p*) *tarith*+ *tcomsq* per iteration

Strip partition

Two edges where data points are exchanged.Communication time is given by $t_{\text{commcol}} = 4(t_{\text{startup}} + nt_{\text{data}})$



Strips

Total cost = 7×*n*×(*n/p*) *tarith*+ *tcommcol* per iteration

RESULTS ON RAVEN CLUSTER for STRIP PARTITION CASE 1000 Iterations AMD Processors

n = grid size (nxn) , p = no of processors, times in seconds.

n—p	2	4	8	16	24
1024	94.5	53.8	33.2	22.07	x
700	44.89	28.91	19.95	15.28	x
350	11.33	11.86	11.27	x	x

RESULTS ON CSWUK1 for 1000 iterations

STRIP PARTITION CASE n: grid size (nxn) , p = no of processors, times in seconds.

n - p	2	4	8	16
700	166.04	85.51	44.17	23.71
350	42.81	22.31	11.73	5.83
175	10.44	4.18	2.58	1.95
87	2.09	1.45	1.25	1.22

BLOCK PARTITION CASE n: grid size is (nxn) , p = no of processors, times in seconds.

n -p	2	4	8	16
700	161.04	83.06	43.20	23.41
350	40.95	21.60	11.6	5.69
175	10.12	4.34	2.71	2.12
87	1.81	1.51	1.32	1.42

Efficiency Comparison

STRIP Partition.

$$\textit{Speedup} = \frac{7\, n \times n\ \textit{tarith}}{7(nxn/p)\ \textit{tarith} + 4(\textit{tstart} + n\ \textit{tdata})}$$

Let $ts = (tstart)/(tarith)$ and Let $td = (tdata)/(tarith)$ then as

Efficiency = Speedup / p

$$\textit{Efficiency} = \frac{1}{1 + 4/7\,(ts + n\, td)\,(p/(n \times n))}$$

Hence for constant efficiency (*p/n* constant if tstart small. If tstart large *(nxn) /p* stays constant.

10**6	100	100	100	100	100	100	100	100	100	100	100	100
10**5	100	100	100	100	100	100	100	100	100	100	99	95
10**4	100	100	100	100	100	100	99	98	93	77	45	17
10**3	100	100	99	98	95	87	65	33	11	3	0	0
10**2	89	78	61	38	17	6	2	0	0	0	0	0
10**1	7	4	2	0	0	0	0	0	0	0	0	0
N/**Proc**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**

Strip Partition EFFICIENCY for Ts = 1000 Td = 50, N = nxn

Efficiency Comparison

Block Partition.

$$Speedup = \frac{7\, n \times n\ tarith}{7(nxn/p)\ tarith + 8(\ tstart + n/\sqrt{p}\ \ tdata)}$$

Let ts = ($tstart$)/($tarith$) and Let td = ($tdata$)/($tarith$) then as

Efficiency = Speedup / p

$$Efficiency = \frac{1}{1 + 8/7\ (\ ts + n/\sqrt{p}\ td)\ (p/(n \times n))}$$

Hence for constant efficiency ($p/(nxn)$)constant so load on each processor $(nxn)\ /p$ stays constant.

10**6	100	100	100	100	100	100	100	100	100	100	100	100
10**5	100	100	100	100	100	100	100	100	100	100	100	100
10**4	100	100	100	100	100	100	100	100	99	98	95	88
10**3	100	100	99	98	96	93	85	71	49	25	9	3
10**2	81	68	51	34	19	9	4	1	0	0	0	0
10**1	4	2	0	0	0	0	0	0	0	0	0	0
N/Proc	2	4	8	16	32	64	128	256	512	1024	2048	4096

Block Partition EFFICIENCY for Ts = 1000 Td = 50

Isoefficiency Isomemory and Isotime

- Begin with speedup formula
- Assume efficiency remains constant
- Determine relation between strong and scalability and isoefficiency (constant efficiency)
- Define scalability function
- Define relationship between weak scaling and isotime
- Explain result regarding isoefficiency isotime and computational complexity

Strong Scalability : Isoefficiency if and only if strong scalability

Strong Scalability for fixed n exists p such that parallel time is reduced like 1/p as p increases i.e.

$$Ser(n) + Par(n)/p + com(n,p) = \frac{const_1}{p} \qquad **$$

Constant Efficiency (Isoefficiency) means that

$$E(n,p) = const_2 = \frac{Ser(n) + Par(n)}{(Ser(n) + Par(n)/p + com(n,p))\,p}$$

Using equation ** to simplify the bottom part of the fraction gives

$$const_2 = \frac{Ser(n) + Par(n)}{const_1}$$

For strong scalability this is fixed

Isoefficiency if and only if strong scalability

As Ser(n) + Par(n) is **just the fixed serial execution time** that we are reducing by using more cores the two constants are related by

$$const_2 = \frac{Ser(n) + Par(n)}{const_1}$$

Hence for each n that there is strong scalability the strong scalability equation implies that E(n,p)=constant.

Similarly for each value of fixed n and associated p values that E(n,p) constant as (Ser(n)+Par(n)) also constant it follows that strong scalability equation holds

Scalability Function

- Suppose isoefficiency relation is $n \geq f(p)$
- Let $M(n)$ denote memory required for problem of size n
- $M(f(p))/p$ shows how memory usage **per processor** must increase to maintain same efficiency
- We call $M(f(p))/p$ the scalability function

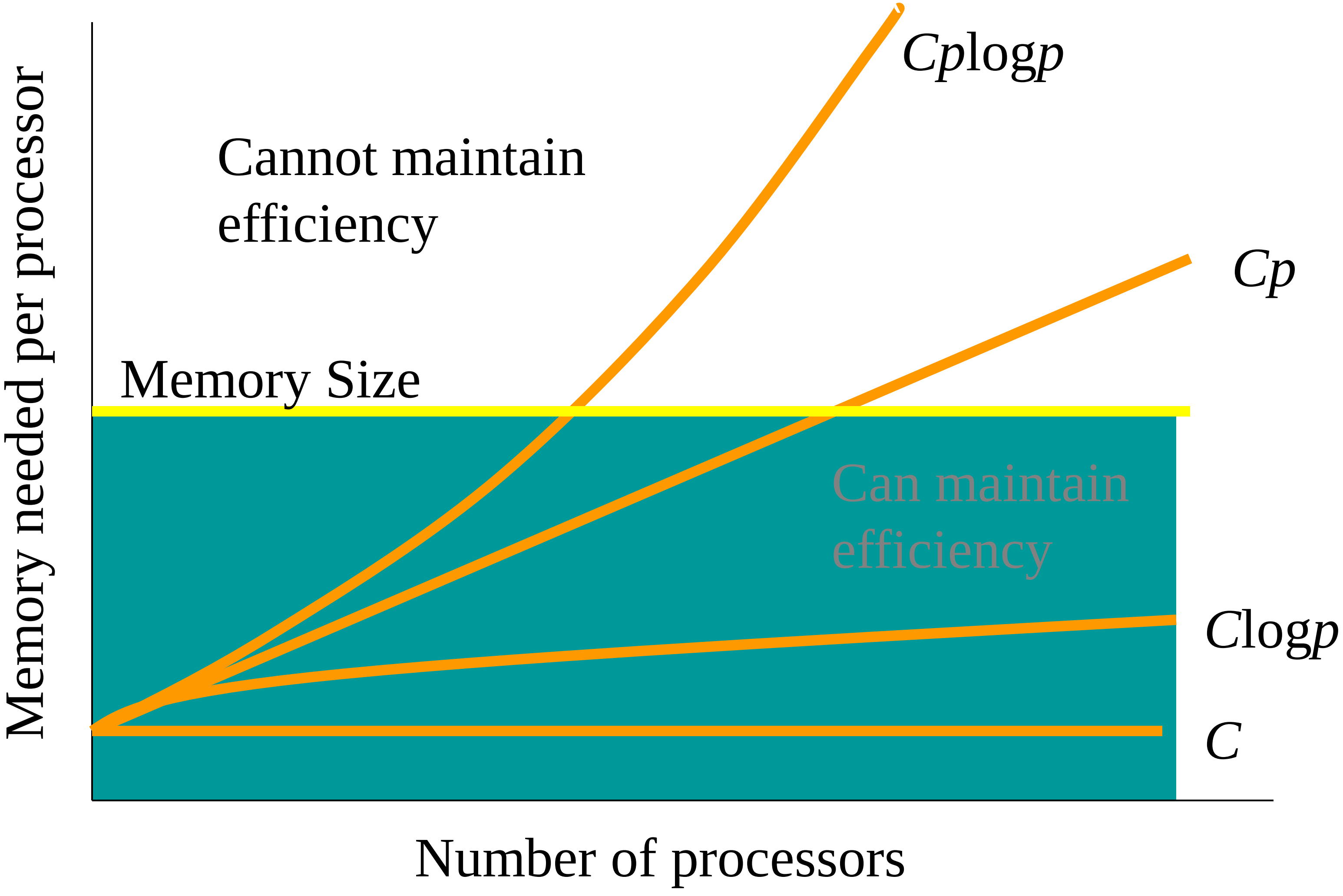
Meaning of Scalability Function

- To maintain efficiency when increasing *p*, we must increase *n*
- Maximum problem size limited by available memory, which is linear in *p*
- Scalability function shows how memory usage per processor must grow to maintain efficiency
- Scalability function a constant means parallel system is perfectly scalable

Isomemory and scalability function

- I**SOMEMORY**: How fast does the problem size have to grow as the number of processors grows to maintain constant memory use per processor.
- Let *M(n)* denote memory required for problem of size *n. M(f(p))/p* shows how memory usage **per processor** must increase to maintain same efficiency
- In contrast Isomemory requires that we pick *n* so that *M(n)/p* is constant

Interpreting Scalability Function



Isotime and weak scalability are identical

Weak Scalability:
for *n* and *p* such that *n/p* is
constant and $Ser(n) + Par(n)/p + com(n,p) = const_1$

then weak scalability holds

Isotime just means that n and p are chosen so that the computation time is the same,
except that n/p need not be constant

Weak scalability is thus a special case of Isotime

For base case n_0 $Ser(n_0) + Par(n_0) = const_1$

Isoefficiency function $f_E(p)$

$f_E(p)$ is rate at which problem size should be increased wrt number of processors to maintain constant efficiency and is $O(p^k)$, $k > 1$

Isotime function $f_T(p)$

$f_T(p)$ is rate at which problem size should be increased wrt number of processors to maintain constant execution time and is $O(p^k)$, $k > 1$

Isomemory function $f_M(p)$

$f_M(p)$ is rate at which problem size should be increased wrt number of processors to maintain constant memory per processor $O(p)$

Relationship between Efficiency and Execution time

As Efficiency E = $(T(n,1)/p) / (T(n,1) / p + T_0(p,n)/p)$

(i) If isotime function keeps $(T(n,1)/p+T_0(p,n)/p)$ constant, isotime model keeps constant efficiency and parallel system is scalable

(ii) If parallel execution time is a function of (n/p), the isotime and isoefficiency functions grow linearly with processors and parallel system is scalable

(iii) Isotime function grows linearly if and only if the algorithm has linear complexity

(iv) If Isotime function grows linearly then isoefficiency function grows linearly and system is scalable.

(v) if isoefficiency grows linearly and the computational complexity is linear then isotime grows linearly and the system is scalable.

See references [1], [2] and [3]

Key Result

The problem is only perfectly scalable if and only it has linear complexity See references [1] and [2]
Reference [3] provides a more general discussion of Isoefficiency

REFERENCES

[1] Ignacio Martin and Fransisco Tirado.
Relationships Between Efficiency and Execution time of Full Multigrid Methods on Parallel Computers.IEEE Trans. on Parallel and Distributed Systems Vol 8 no 6 97 562–573.

[2] Ignacio Martin, Fransisco Tirado and L.Vazquez.
Some Aspects about Scalability of Scientific Applications on Parallel Computers Parallel Computing Vol 22 96 1169–1195.

[3] Anath Grama, Anshul Gupta, George Karypis and Vipin Kumar
Introduction to Parallel Computing (Second edition) Addison Wesley

Linearity in weak scaling

- For problems like the grid calculations on n by n by n grids we can have complexity of $n^3 \ \ but \ \ with \ \ n^3 \ \ unknowns$

- Such problems scale linearly as all we have to do when we double *n is to increase the core count by eight so that there is still constant memory use per core.*

- The complexity of $\frac{(2n)^3}{8p}$ is still $\frac{n^3}{p}$ and we can get weak scaling

Weak and Strong Scaling

For **strong scaling** $T(n^*,p) = f(n^*)/p$

for a fixed n^* and some function $f(.)$

For **weak scaling** $T(n,p)$ = const where $n = g(p)$ for some function $g(p)$.

Hence for weak and strong scaling $f(g(p)) = p$ and suppose $f(n) = n^q$

then $p = n^q$

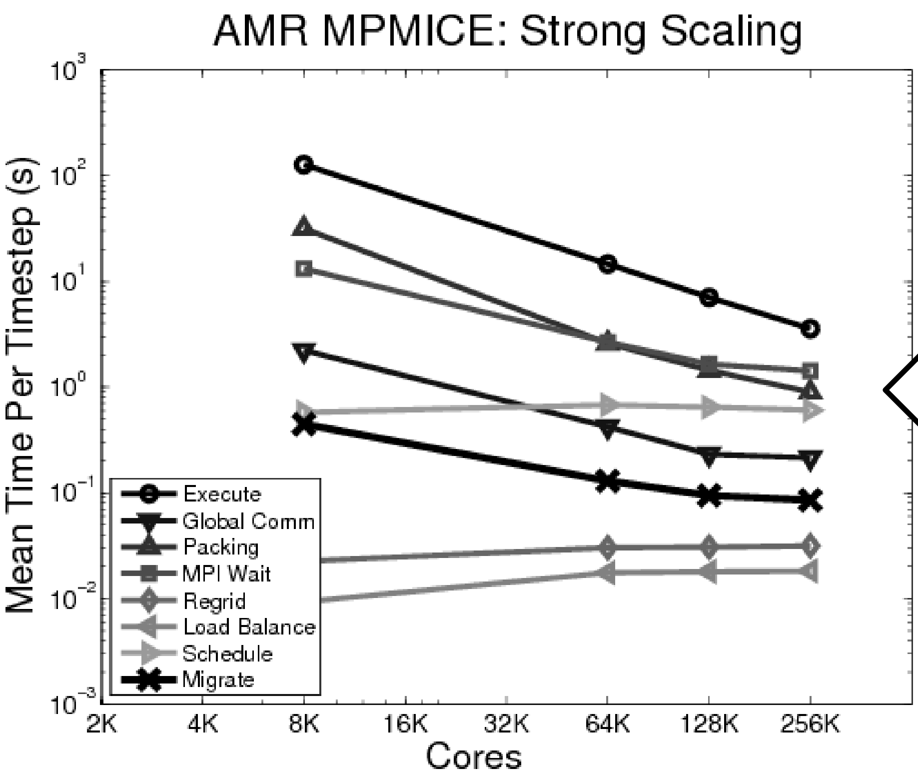
and the number of processors grows much faster than n to achieve both weak and strong scalability. n=3 is common in our applications. While not perfectly scalable this is satisfactory and is in fact linear complexity in terms of the total number of unknowns $n \times n \times n$.

Linearity in weak scaling

- Let $t = Ser(n) + \frac{n^{\alpha}}{p} + log(p)$
- If we double *p* then for weak scaling it follows that *n must be multiplied by a constant k such that* $(kn)^{\alpha} = 2n^{\alpha}$
- Hence $k = 2^{1/\alpha}$ **so the problem size doesn`t double**
- This means that the problem size grows more slowly than it should
- Here n is the total problems size
- Note that for grid problems if the complexity is n^3 i.e. *n by n by n and te total problem size is* n^3 we go to 8x cores when doubling n. This gives constant time and constant memory per core

Scaling Large Software Frameworks

- Theoretical models are fine for understanding small problems and even models of large codes
- The reality of scaling large codes is that we have to use a measurement based approach and time every component

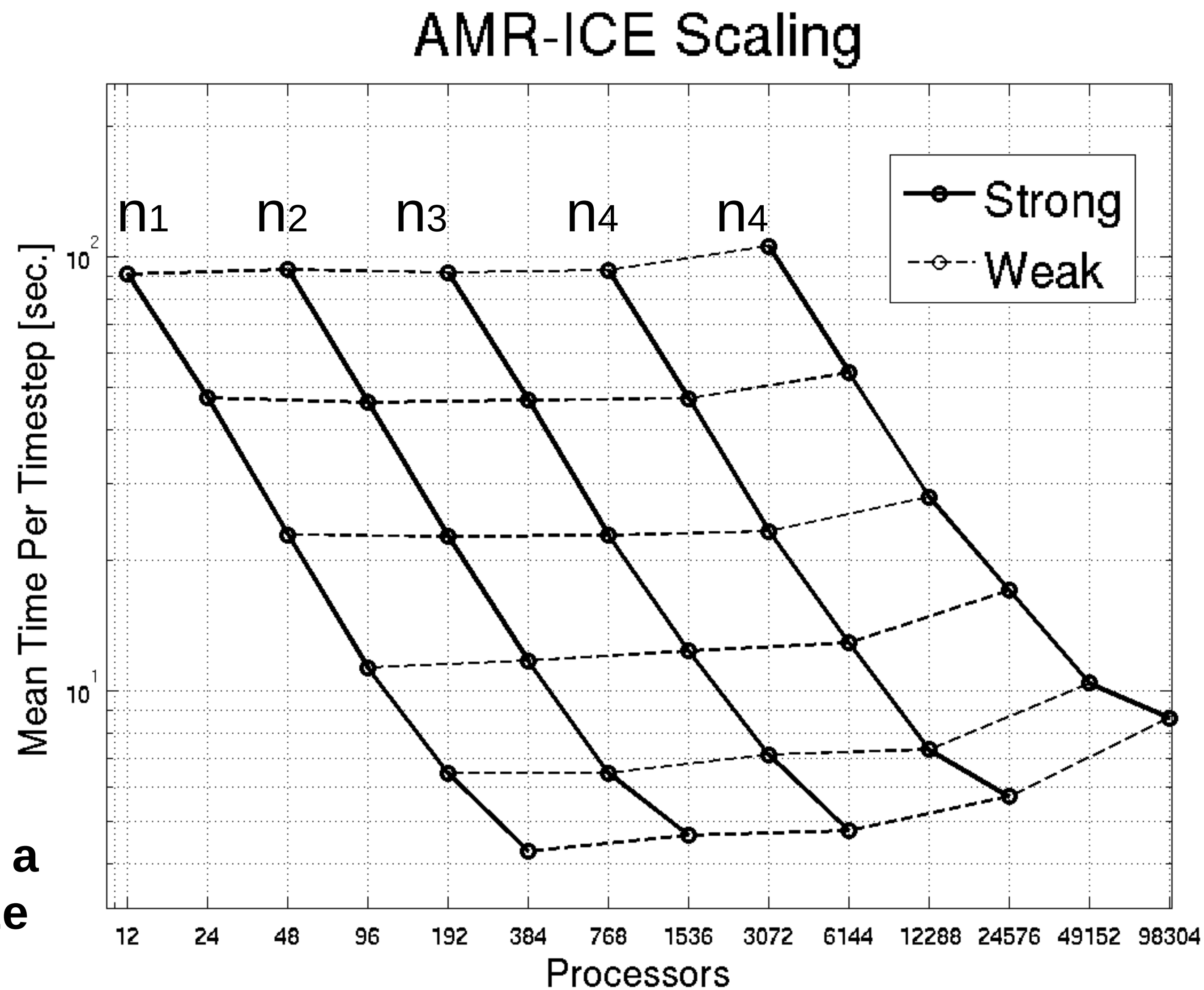


Strong Scaling Breakdown

UINTAH SCALABILITY

At 98k Proc
1 16x16x16
patch per
Core and so
Scalability fades

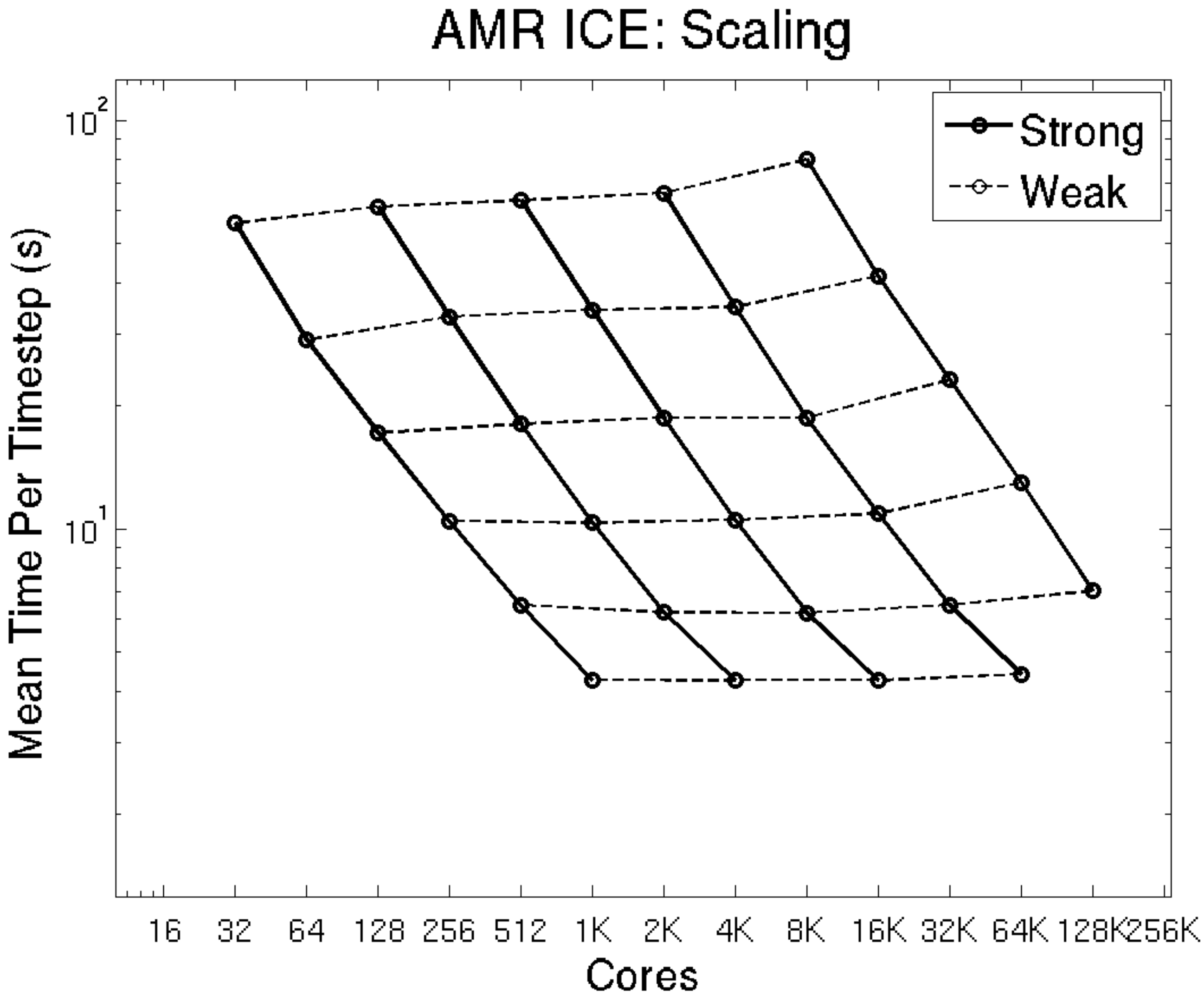
Problem is essentially an advected blob, moving across a domain



Each solid line is a fixed problem size Instance n_i

NSF NICS Kraken 6-core AMD based machine

Scalability on Titan



Distributed Controller

One flow with particles moving
3-level AMR ICE

Weak and Strong Scalability: Problem size n on P processors

Strong Scalability $T(n,p) = T(n,1)/p$

Weak Scalability $T(np,p) = T(n,1)$

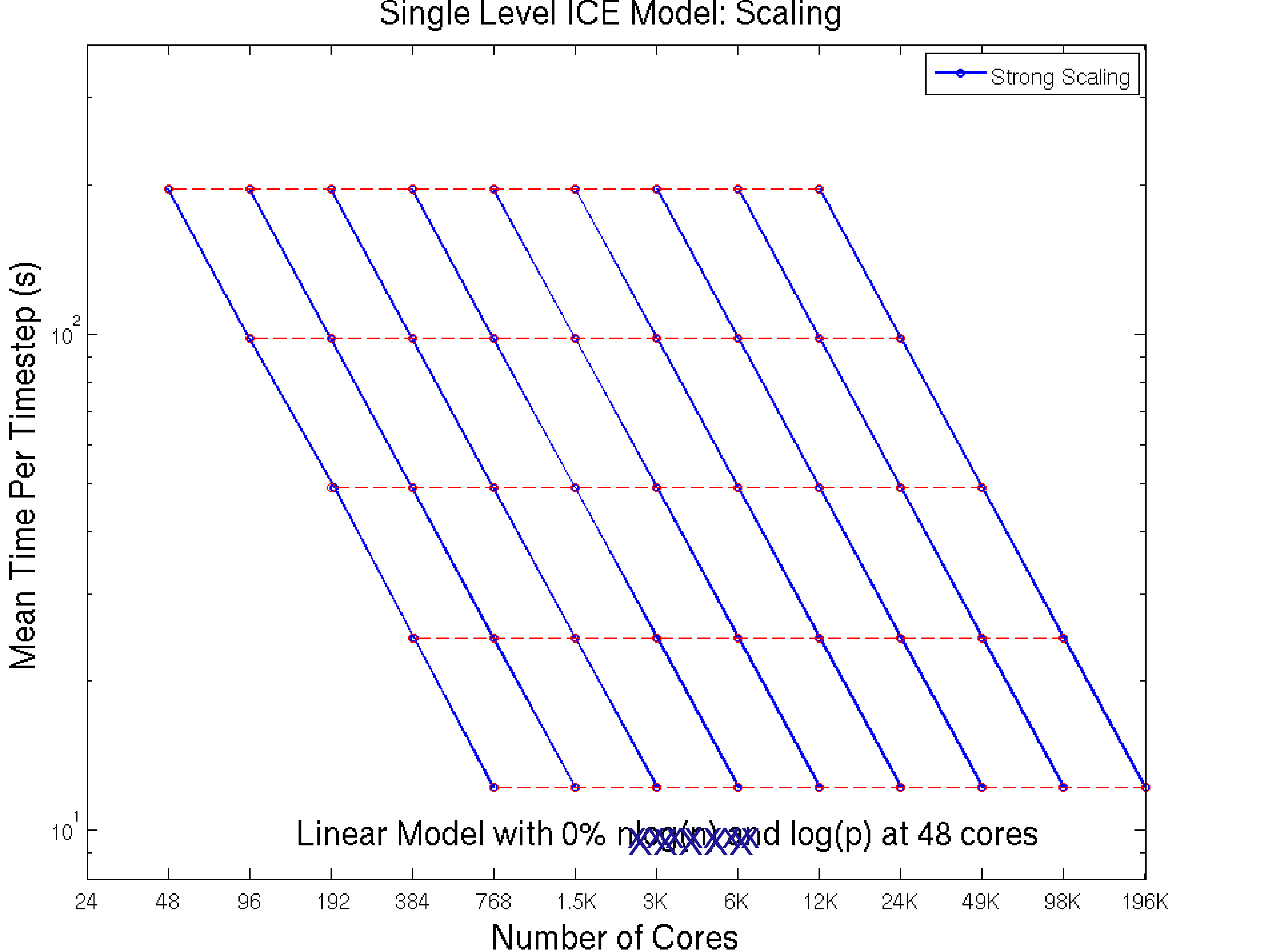
Constant time *T(kn, kp)* for larger problem *kn* on *k* more cores

Both weak and strong scalability only if $T(n,1) = \alpha n$

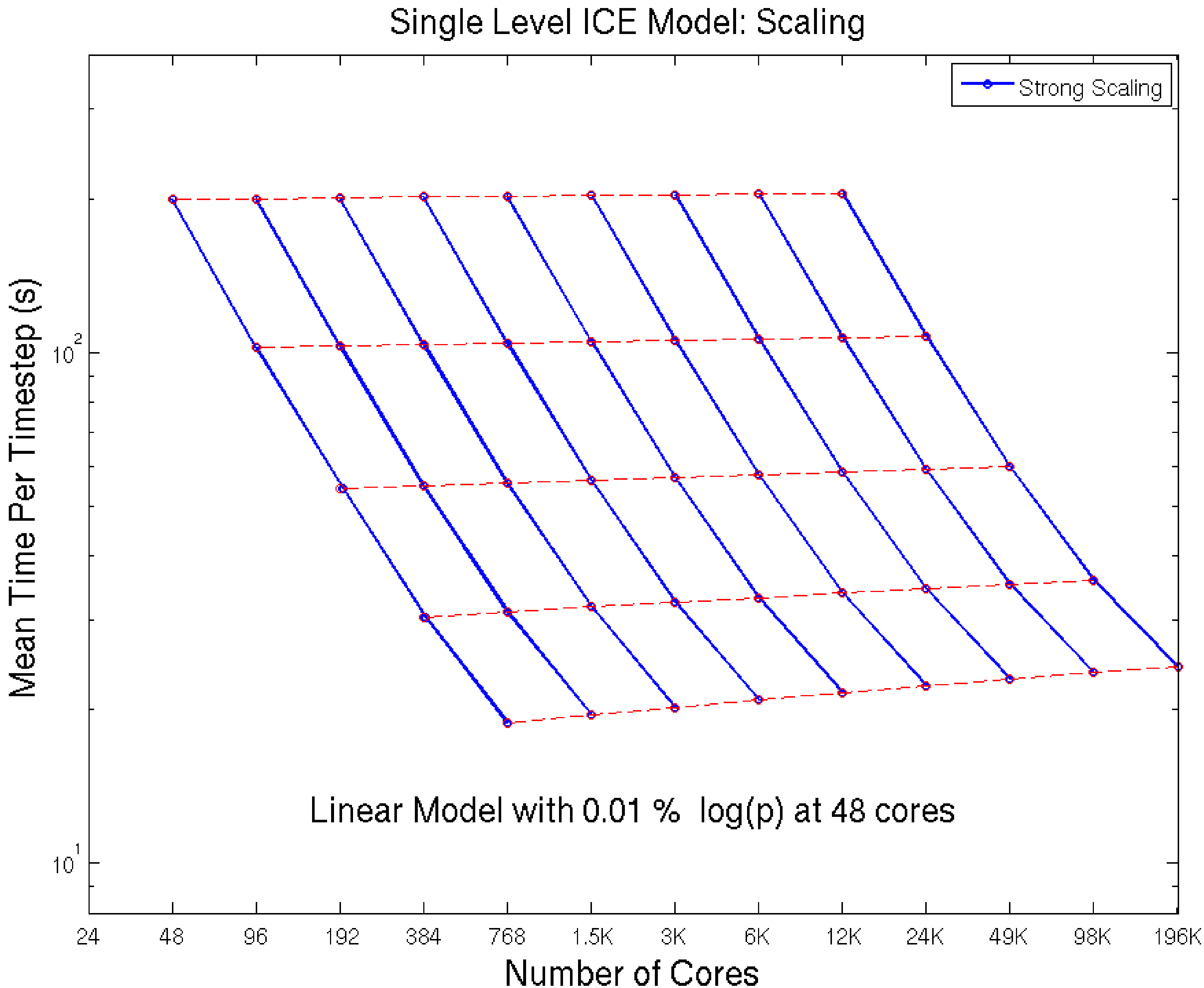
More realistic model including global collectives

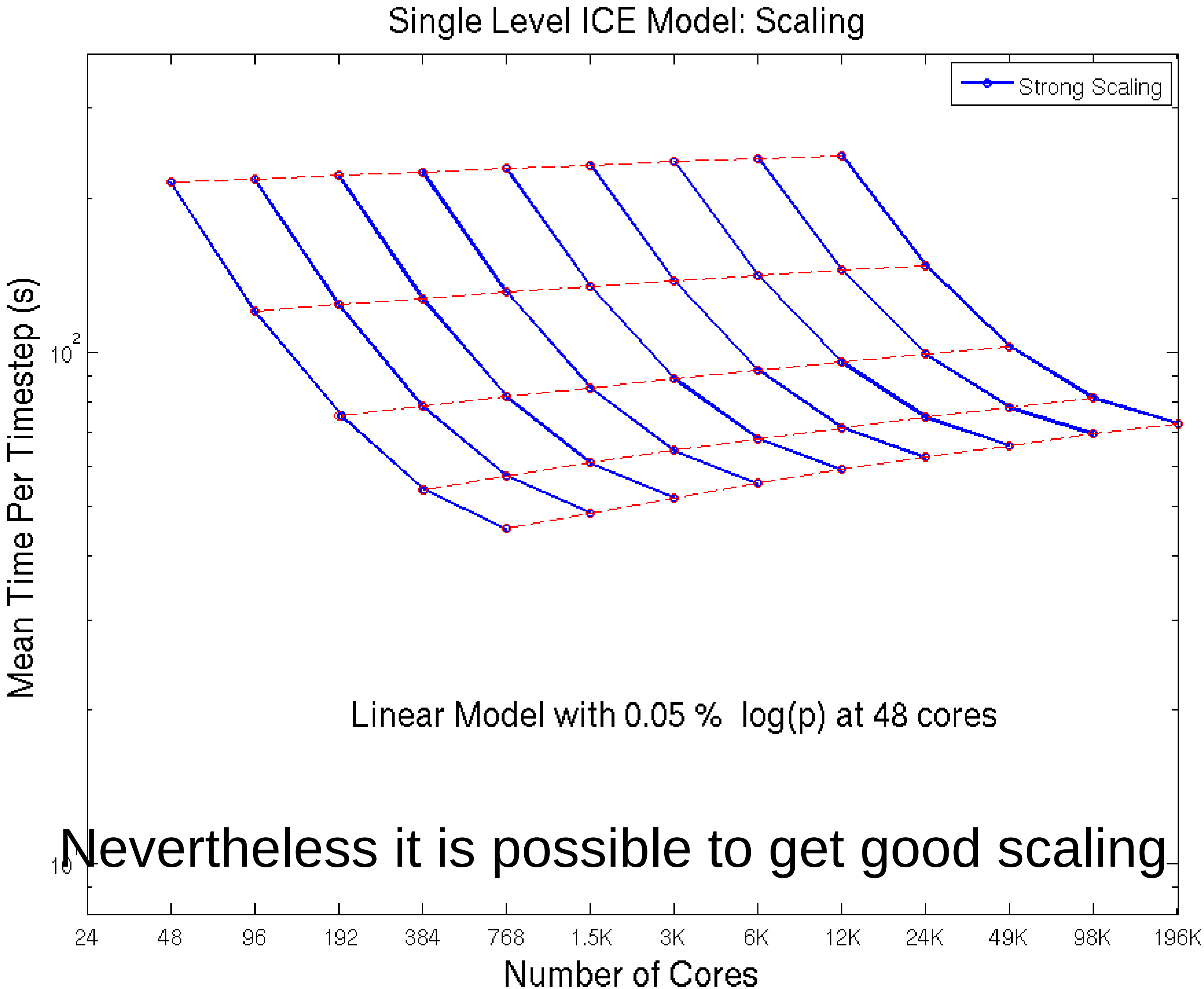
$$T(n,p) = \alpha n + \gamma \log(p)$$

$\log(p_0)\gamma/(\alpha n_0)$ Is fraction of time spent in global collectives at $n_0 p_0$



Single Level ICE Model: Scaling
Strong Scaling
Mean Time Per Timestep (s)
10^2
10^1
Linear Model with 0% nlog(n) and log(p) at 48 cores
24
48
96
192
384
768
1.5K
3K
6K
12K
24K
49K
98K
196K
Number of Cores

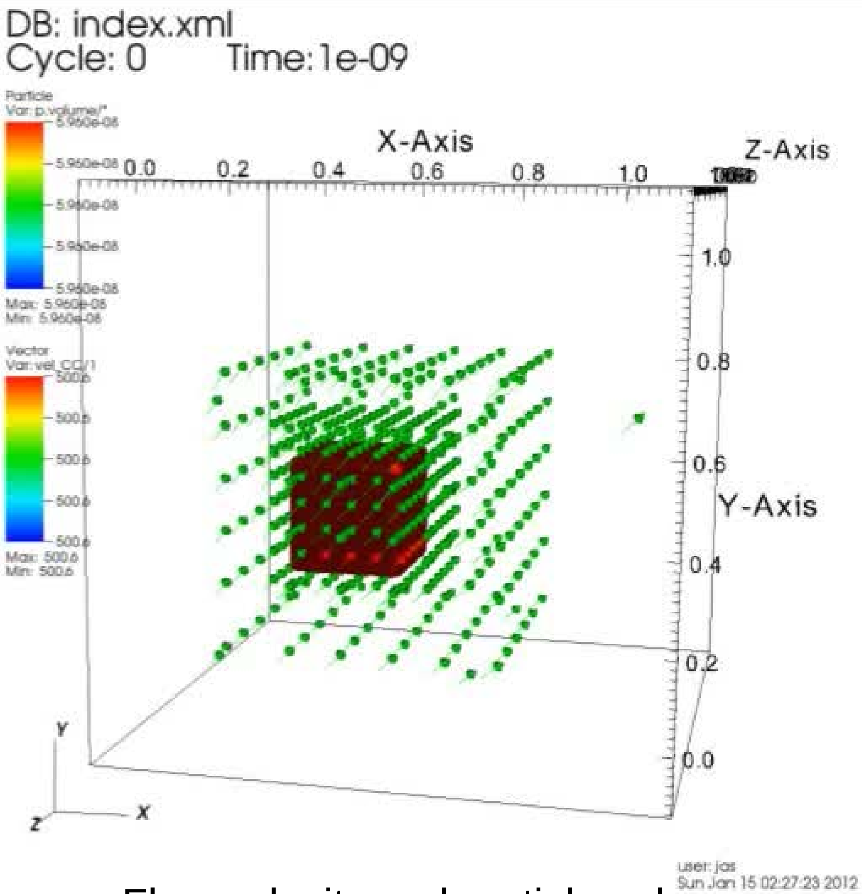

Single Level ICE Model: Scaling
Strong Scaling
Mean Time Per Timestep (s)
10^2
10^1
Linear Model with 0.01 % log(p) at 48 cores
24
48
96
192
384
768
1.5K
3K
6K
12K
24K
49K
98K
196K
Number of Cores



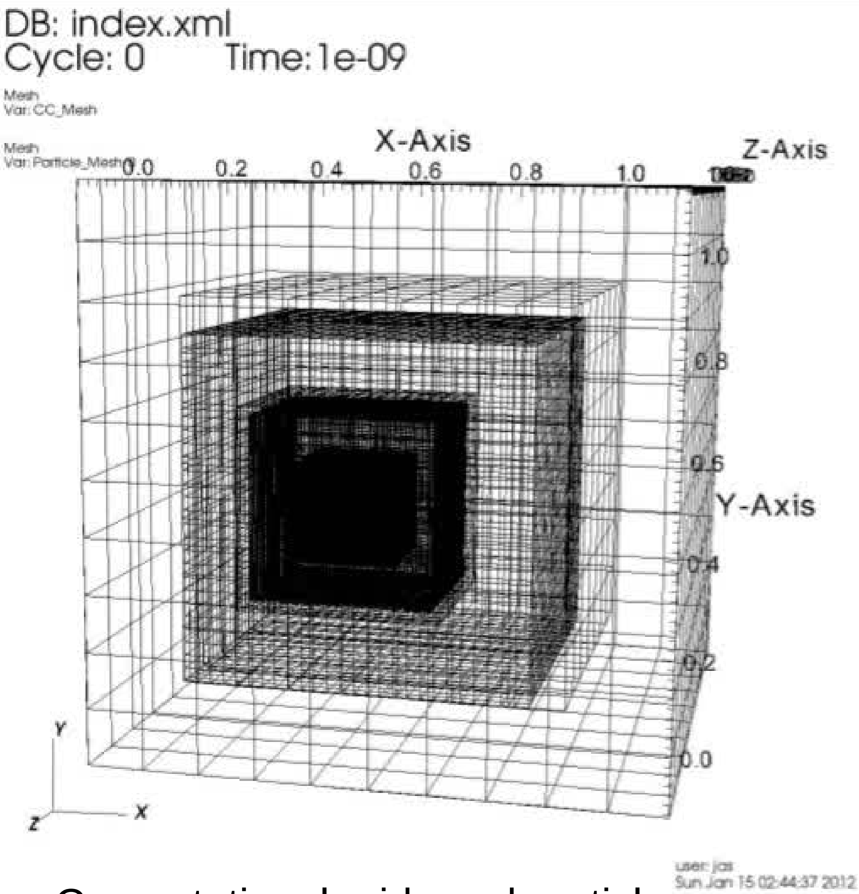
Nevertheless it is possible to get good scaling

Fluid Structure Interaction Example: AMR MPMICE

A PBX explosive flow pushing a piece of its metal container



Flow velocity and particle volume

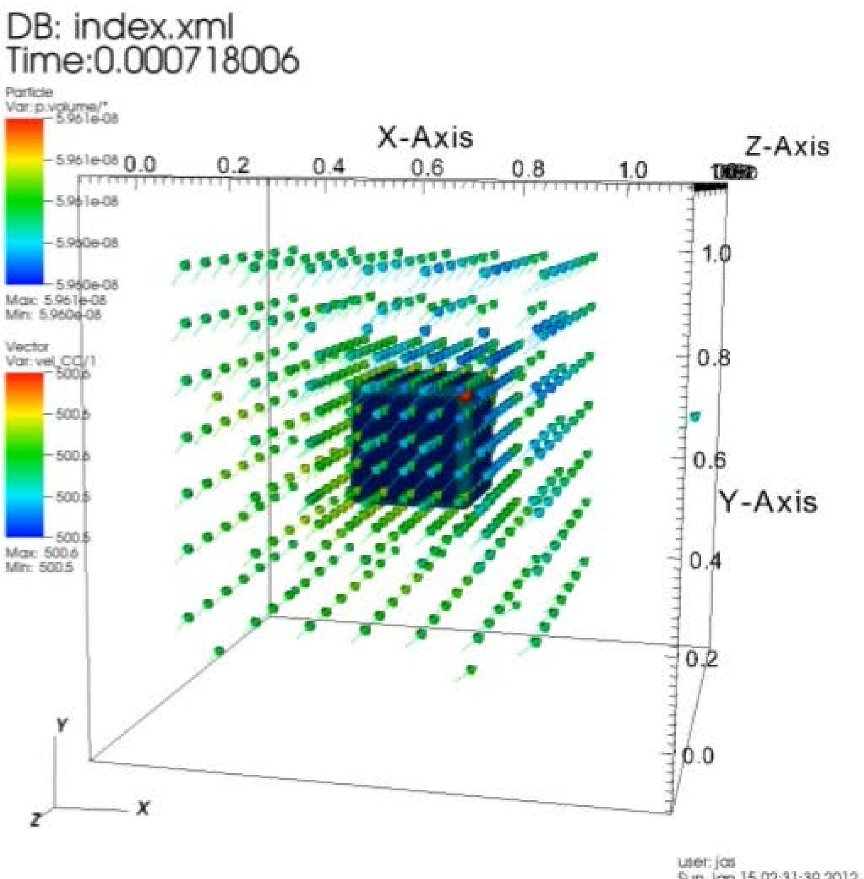


Computational grids and particles

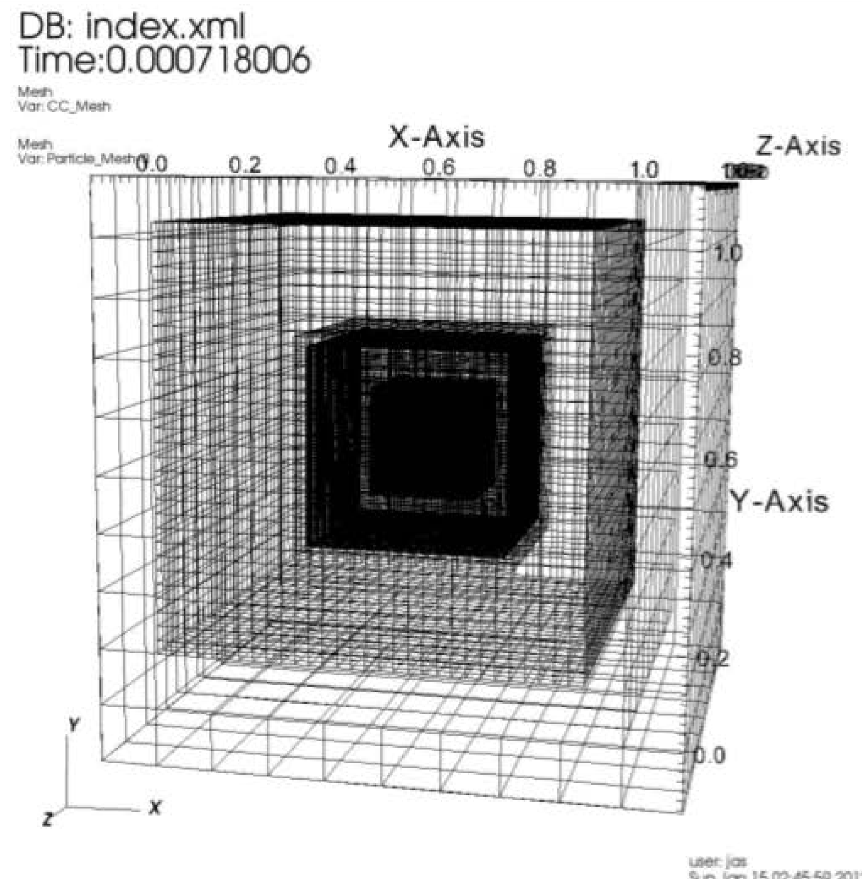
Grid Variables: Fixed number per patch, relative easy to balance
Particle Variables: Variable number per patch, hard to load balance
SEVERE DYNAMIC LOAD IMBALANCES DUE TO PARTICLE MOVEMENT

Fluid Structure Interaction Example: AMR MPMICE

A PBX explosive flow pushing a piece of its metal container



Flow velocity and particle volume



Computational grids and particles

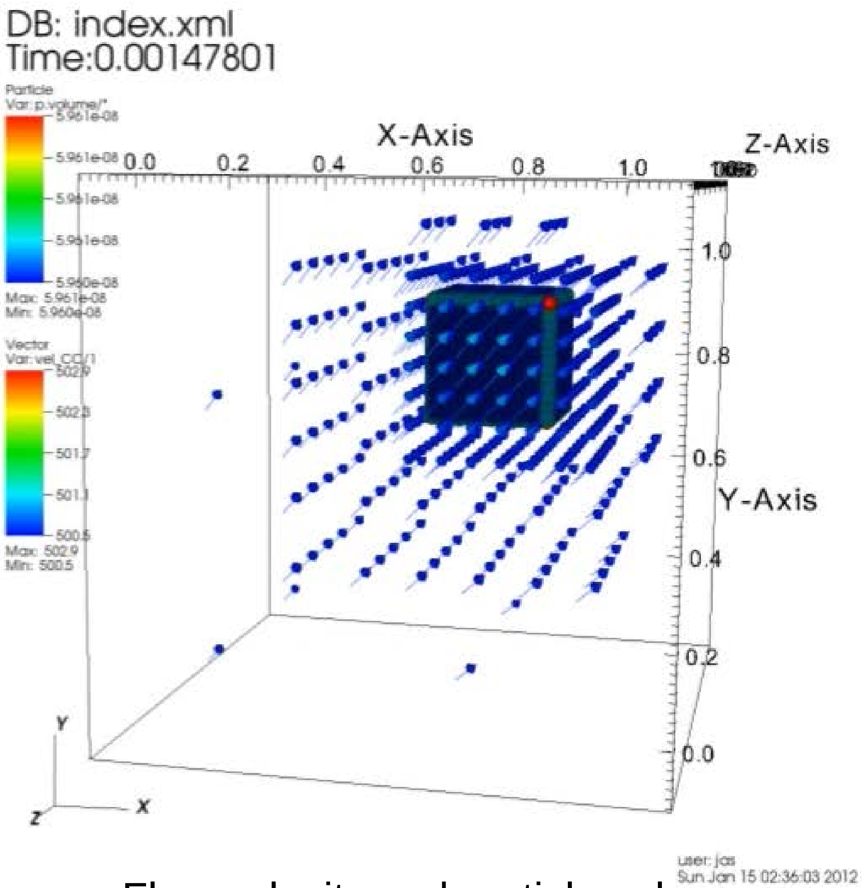
Grid Variables: Fixed number per patch, relative easy to balance

Particle Variables: Variable number per patch, hard to load balance

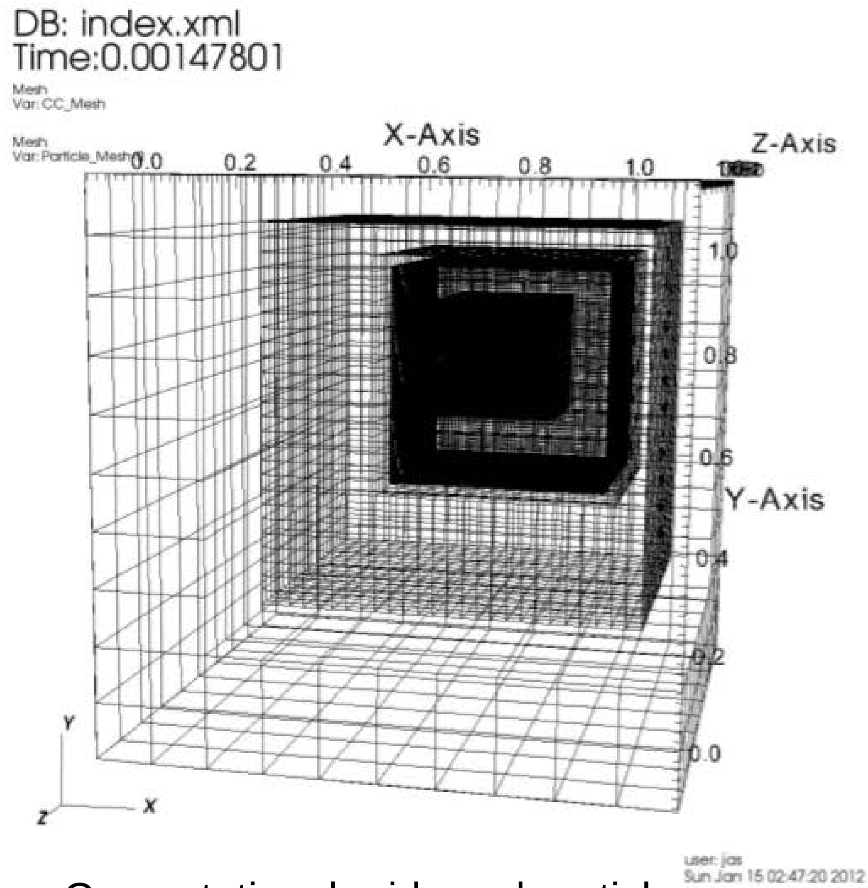
SEVERE DYNAMIC LOAD IMBALANCES DUE TO PARTICLE MOVEMENT

Fluid Structure Interaction Example: AMR MPMICE

A PBX explosive flow pushing a piece of its metal container

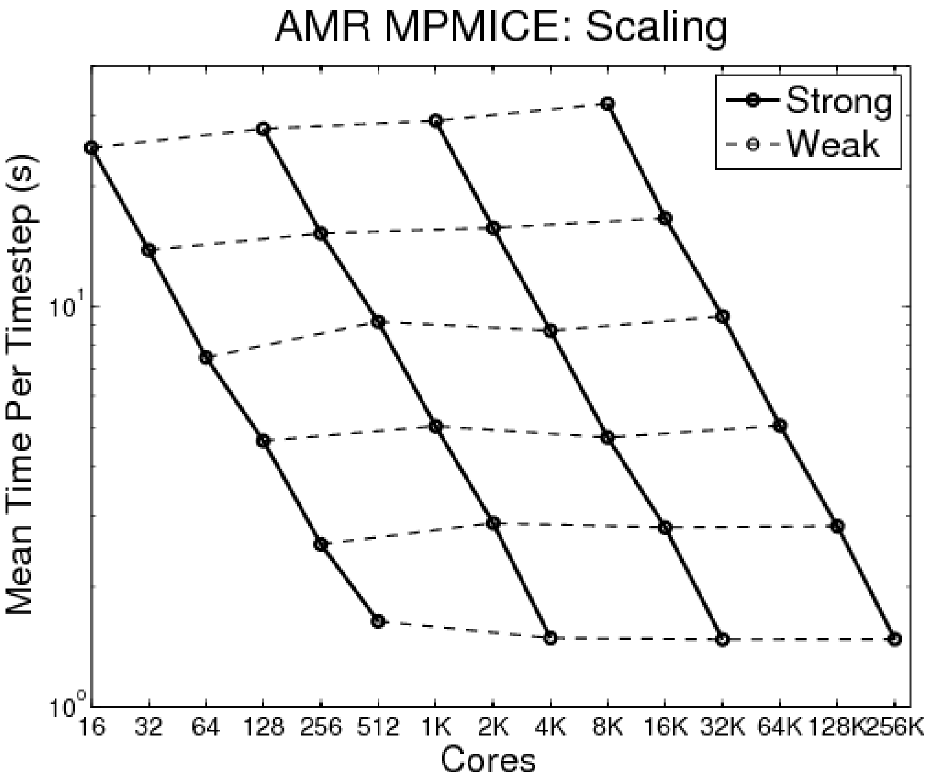


Flow velocity and particle volume



Computational grids and particles

Grid Variables: Fixed number per patch, relative easy to balance
Particle Variables: Variable number per patch, hard to load balance
SEVERE DYNAMIC LOAD IMBALANCES DUE TO PARTICLE MOVEMENT

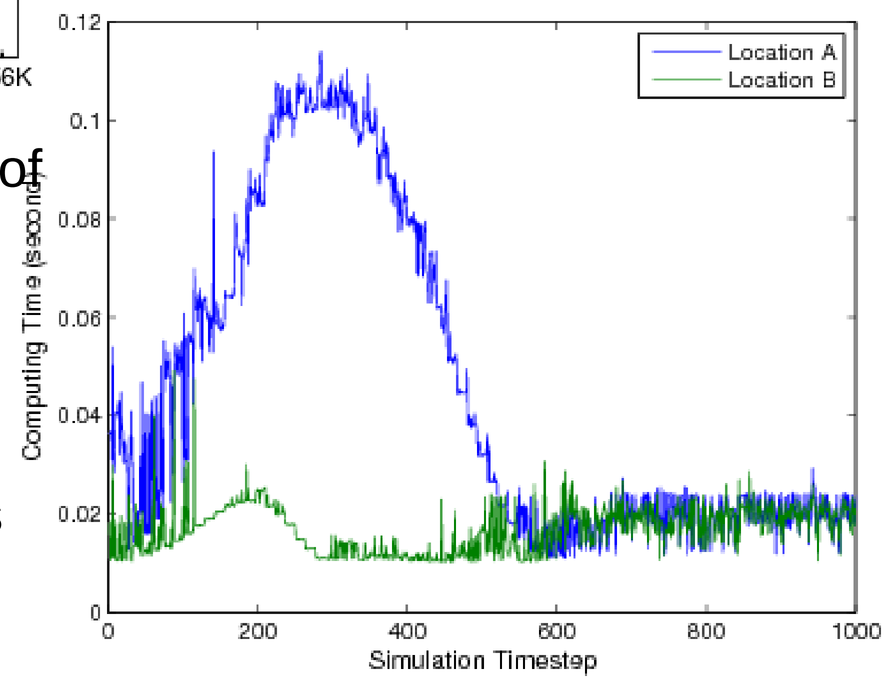


Scalability on Titan CPUs

One flow with particles moving 3-level AMR MPM ICE 70% efficiency at 256K cores vs 16K cores

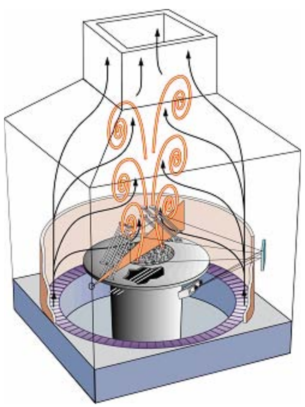
- Challenging scalability - combination of particles fluid-flow and AMR
- New runtime system successful 10x memory and 2x speed
- Shows our ability to make complex problems scale well on large systems

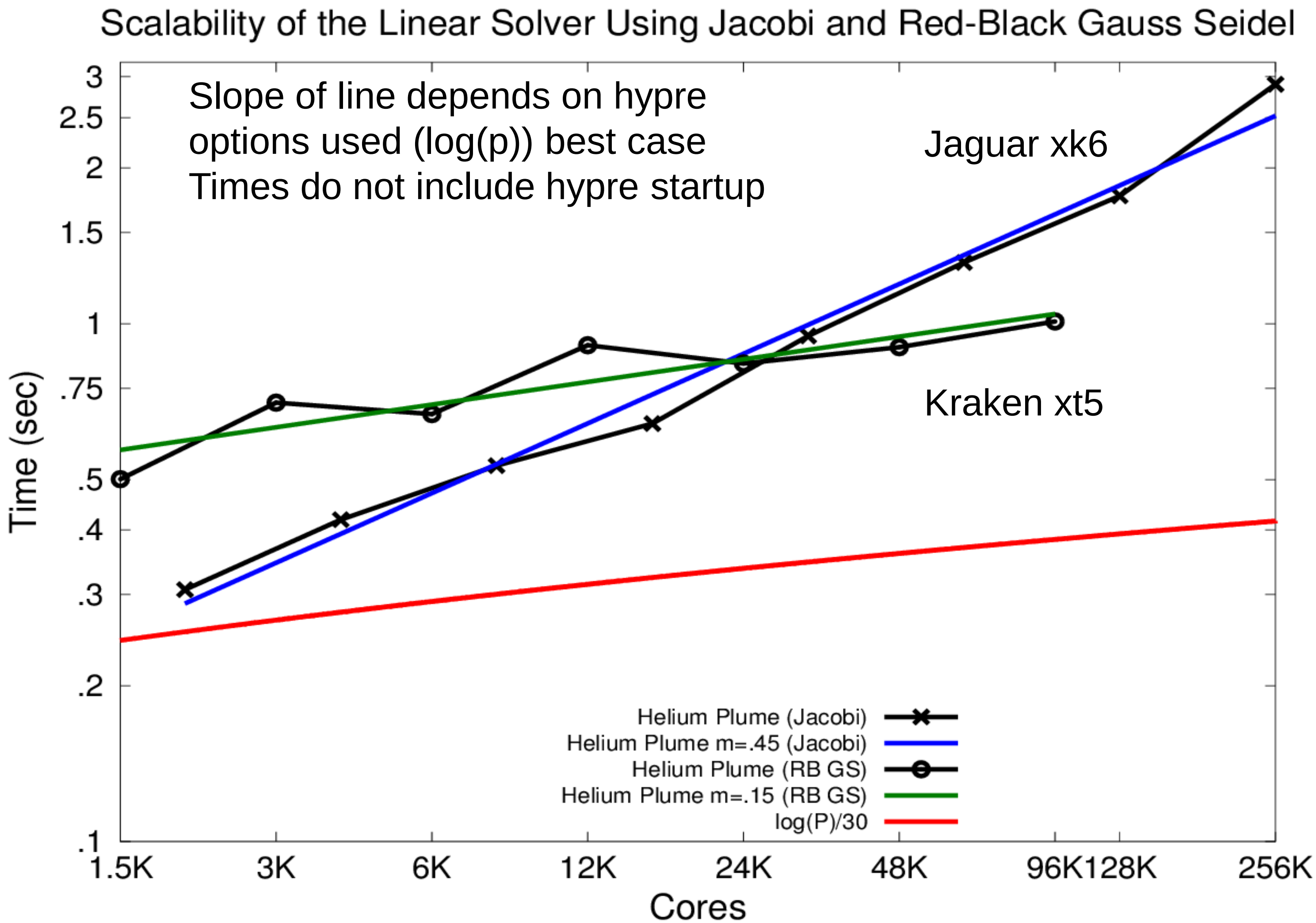
Uneven work load per patch



Results by Qingyu Meng 2012

Scalability with Utah Uintah Buoyant Helium Plume Model

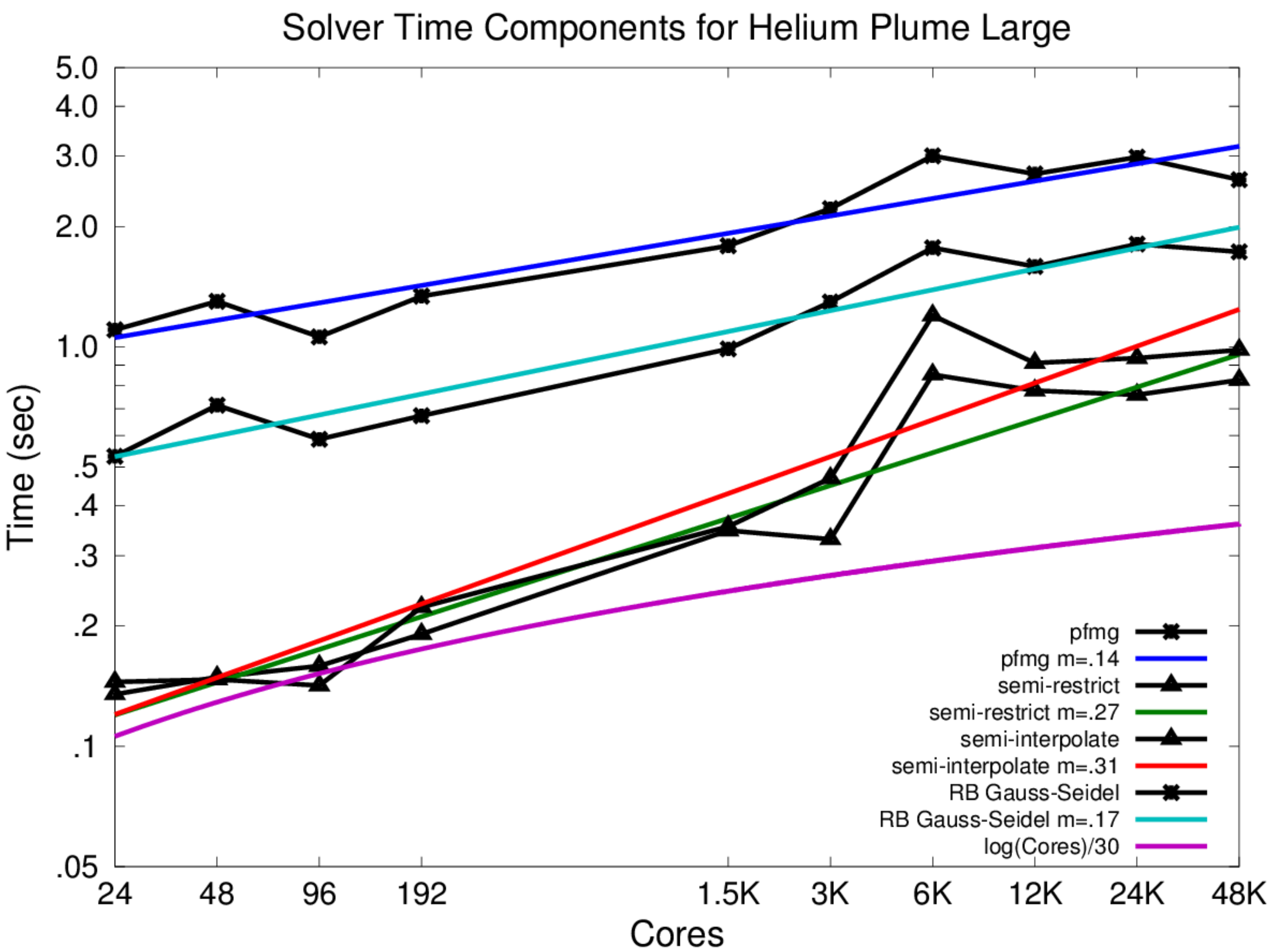




Weak Scalability for implicit calculations using hypre MG Precon CG in Uintah
Code generated by Wasatch DSL.

Results by John Schmidt

Scaling breakdown for hypre linear solver applied to Helium Plume Problem



Communications associated with 330K unknowns per patch & core on Kraken's Seastar network are problematic. Smaller patches ok

John Schmidt

Summary

- Performance terms Speedup, Efficiency
- Model of speedup Serial component
  - Parallel component
  - Communication component
- What prevents linear speedup?
  - Serial operations, communication operations
  - Process start-up, imbalanced workloads
  - Architectural limitations
- Analyzing parallel performance
  - Amdahl's Law, Gustafson-Barsis' Law, Karp-Flatt metric
  - Isoefficiency Isotime and Isomemory metrics
  - Practical Scalability based on measurements and worrying about log(P) Global collectives
-

Part of Example Exam Questions

Question

Given a decomposition of the runtime of a parallel program into
A serial part Ser(n) , a parallel part par(n,p) and a communications
Part comm(n,p):

(i) State Amdahls law and explain what it neglects

(ii) State Gustaffson's law and explain how it is an improvement over Amdahls law

(iii) Define what is mean by the terms
  (i) Speedup
  (ii) Inherently serial fraction, f

(iv) Using Amdahls law derive the Karp Flatt metric as given by $$f = \frac{1/S(n,p) - 1/p}{1 - 1/p}$$

(v) Explain why the Karp Flatt metric may be more useful than either of the other two approaches

(vi) Explain what is meant by Iso efficiency and strong scalability

(vii) Explain what is meant by weak scalability and show that a code with greater than linear computataional complexity cannot weak scale