Advanced ParFlow Short Course: Solvers and Parallel Scaling

October 3-4, 2019

University of Arizona

Laura Condon, University of Arizona
Nick Engdahl, Washington State University
Reed Maxwell, Colorado school of Mines

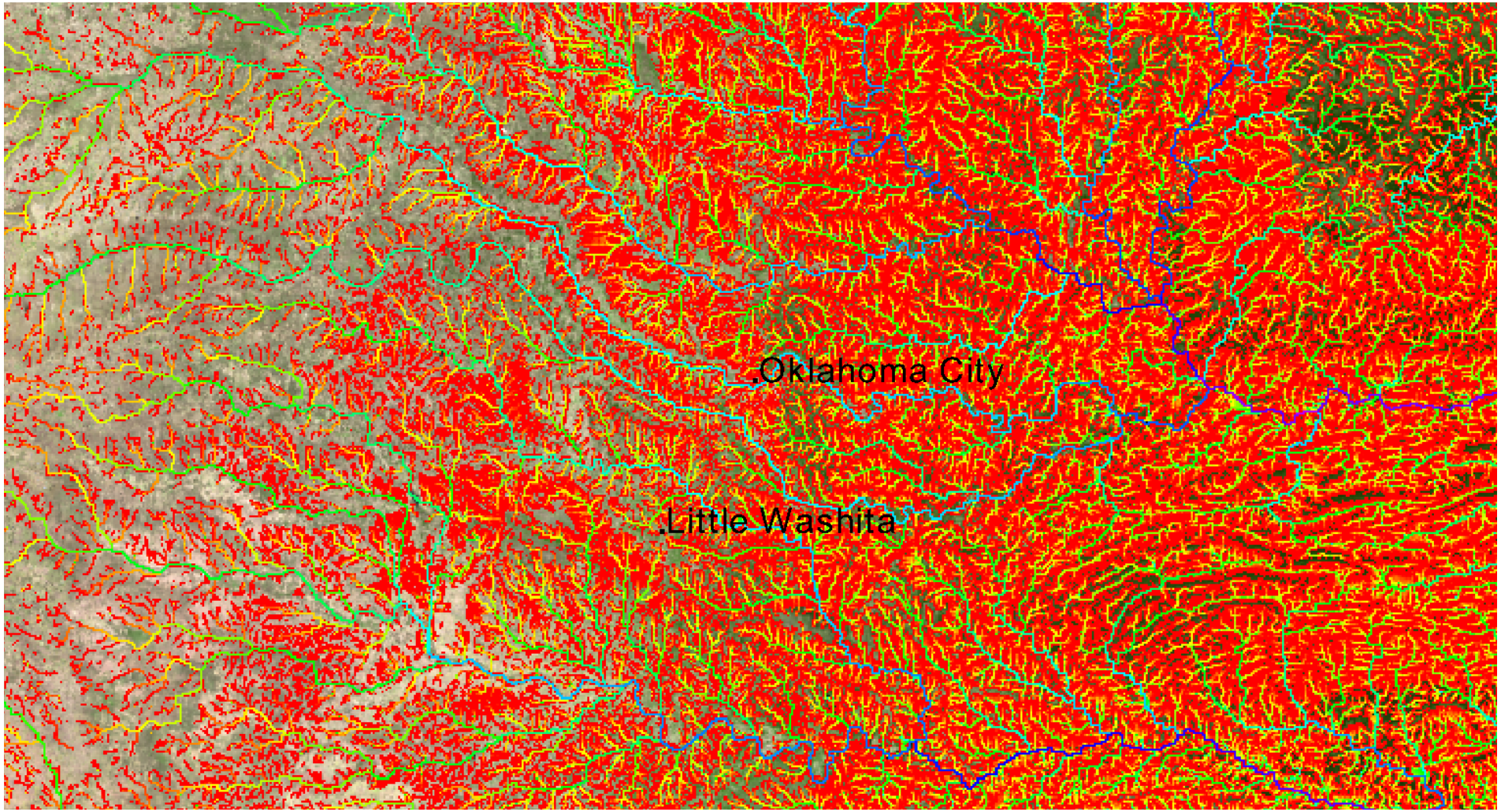
This material and short course was supported by the National Science Foundation and Department of Energy

CUAHSI
Universities Allied for Water Research

U.S. DEPARTMENT OF ENERGY | Office of Science

IDEAS productivity

Solvers and Parallel Scaling



Let's explore a simple 1-D Kinematic Wave example

We can say the momentum terms are so *small* that we just neglect them and we can equate *bed slope* with *friction slope*:

$$S_f = S_0$$

$$\frac{\partial V}{\partial t} + V\frac{\partial V}{\partial x} + g\left(\frac{\partial h}{\partial x} + S_f - S_0\right) + \frac{iV}{h} = 0$$

0 0 0 0

This gives us: $$\frac{\partial h}{\partial t}+\frac{\partial}{\partial x}\left(Vh\right)=0$$

for continuity and $$V=\frac{\sqrt{S_0}h^{\frac{2}{3}}}{n}$$

for velocity. Combining we get

$$\frac{\partial h}{\partial t}+\frac{\partial}{\partial x}\left(\frac{\sqrt{S_0}h^{\frac{5}{3}}}{n}\right)=0$$

Kinematic wave becomes:

$$\frac{\partial h}{\partial t} + \frac{\sqrt{S_0}}{n}\frac{\partial (h^{\frac{5}{3}})}{\partial x} = 0$$

Taylor series discretization

$$\frac{\partial h}{\partial t} = \frac{h_i^{j+1} - h_i^j}{\Delta t} \qquad \frac{\partial h}{\partial x} = \frac{h_{i+1}^j - h_i^j}{\Delta x}$$

Discretized kinematic & Manning's equation

$$\frac{h_i^{j+1} - h_i^j}{\Delta t} + \frac{\sqrt{S_0}}{n}\frac{(h_{i+1}^{j\frac{5}{3}} - h_i^{j\frac{5}{3}})}{\Delta x} = 0$$

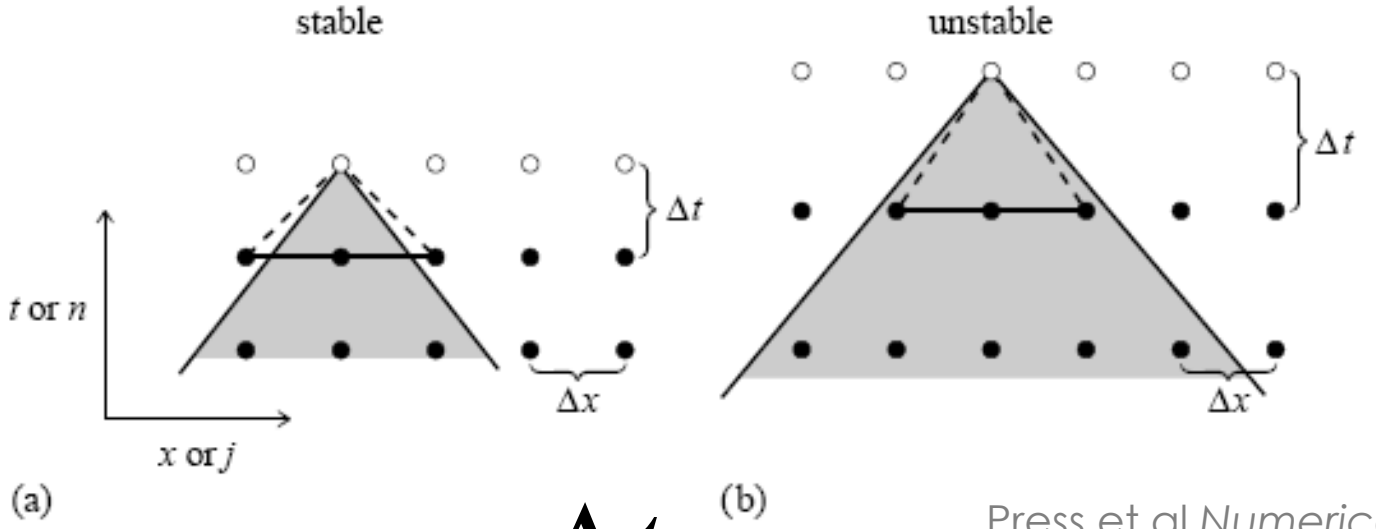
Solving for h at future time, $h^{j+1}{}_i$

$$h_i^{j+1} = h_i^j - \frac{\Delta t\sqrt{S_0}}{n}\frac{(h_{i+1}^{j\frac{5}{3}} - h_i^{j\frac{5}{3}})}{\Delta x} + q$$

Stability is an important consideration

Stability - Courant – Friedrichs – Lewy Condition (CFL Condition)

The CFL Condition requires that the finite-difference solution include the domain of the associated partial differential equation.

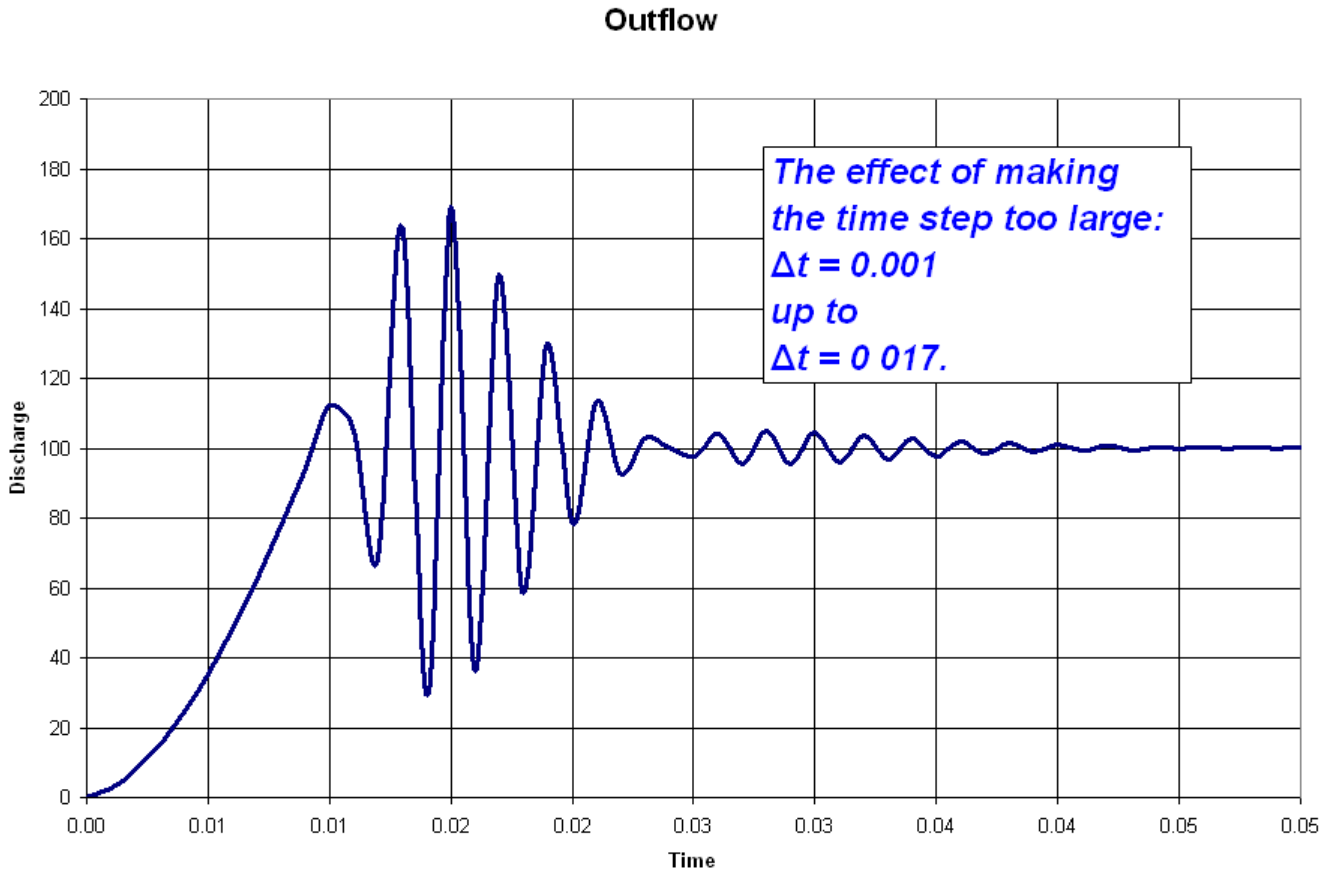


Press et al *Numerical Recipes*

$$\frac{v\Delta t}{\Delta x} \leq 1$$

Finite-Difference Calculus

Stability - Courant – Friedrichs – Lewy Condition (CFL Condition)



Newton's Method

- Newton's method finds the root of an equation
- It is iterative and requires the *function* and the *derivative* of that function at an arbitrary point x
- It is generally demonstrates quadratic convergence

For some function *f(x)* and the derivative of that function *f'(x)* we will geometrically extend a tangent of some current point x_k to zero choosing the next point x_{k+1} and repeating. We start with the Taylor series (again):

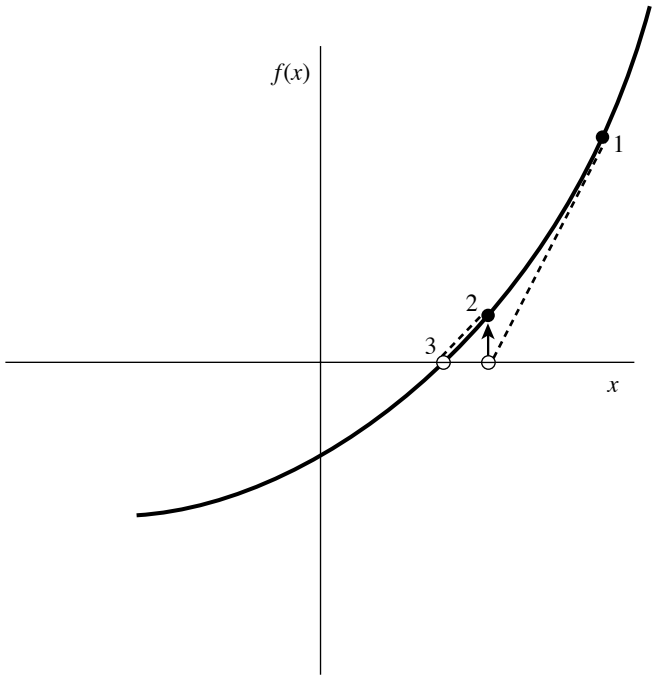
$$f(x+\delta) \approx f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \ldots$$

Where for small enough δ and well-behaved functions $\delta = \frac{f(x)}{f'(x)}$

What this means is if we want to find a value for x then we can iterate as follows:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Where *k* is in iterate as we improve our solution.



An example

$$x^3 = 400$$

$$f(x) = x^3 - 400$$

$$f'(x) = 3x^2$$

x^3=400

f(x)=x^3-400

f'(x)=3x^2

xk+1=xk-f(x)/f'(x)

	X	f(x)	f'(x)
x0	11	931	363
x1	8.43526171	200.199576	213.46092
x3	7.49738711	21.4342278	168.63244
x4	7.37028091	0.36132949	162.963122
x5	7.36806366	0.00010869	162.865086
x6	7.368063	9.8908E-12	162.865057

Why do we care about root finding?

- We can re-write our differential system for non-linear systems as a function
- Our solution is found when our function goes to zero
- This allows us to solve nonlinear systems without ever, well solving them
- For example, let's write our overland flow system *implicitly*

$$\frac{h_i^{j+1} - h_i^j}{\Delta t} + \frac{\sqrt{S_0}}{n} \frac{\left(\left(h_{i+1}^{j+1}\right)^{\frac{5}{3}} - \left(h_i^{j+1}\right)^{\frac{5}{3}}\right)}{\Delta x} = 0$$

Now we can't isolate *h* as we could using the forward different approximation and we can't factor pressure at the current time, so we can write our *function* as

$$f(h_i^{j+1})_k = h_i^{j+1} - h_i^j + \frac{\Delta t \sqrt{S_0}}{n} \frac{\left(\left(h_{i+1}^{j+1}\right)^{\frac{5}{3}} - \left(h_i^{j+1}\right)^{\frac{5}{3}}\right)}{\Delta x}$$

Where our derivative would look something like

$$f'(h_i^{j+1})_k = 1 - \frac{5\Delta t\sqrt{S_0}}{3n\Delta x}\left(h_i^{j+1}\right)^{\frac{2}{3}}$$

and we could use

$$h_{k+1} = h_k - \frac{f(h_k)}{f'(h_k)}$$

to solve our system

What does this look like for ParFlow?

$$F(h^n_{i,j,k}) = \Delta x_i \Delta y_i \Delta y_i \Delta z_k S_{s,i,j,k} S_w(h)^n_{i,j,k} (h^n_{i,j,k} - h^{n-1}_{i,j,k}) + \Delta x_i \Delta y_i \Delta z_k \phi_{i,j,k} [S_w(h)^n_{i,j,k} - S_w(h)^{n-1}_{i,j,k}] - \Delta t^n \Delta x_i \Delta y_j \Delta z_k q^n_{r,i,j,k} - \Delta t^n \Delta x_i \Delta y_j \Delta z_k \left[\frac{q^x_{i+1/2,j,k} - q^x_{i-1/2,j,k}}{\Delta x_i} + \frac{q^y_{i,j+1/2,k} - q^y_{i,j-1/2,k}}{\Delta y_j} + \frac{q^z_{i,j,k+1/2} - q^z_{i,j,k-1/2}}{\Delta z_k} \right]$$

$$q^x_{i+1/2,j,k} = -[\cos\theta_x K_{s,x} k_r(h)]_{i+1/2,j,k} \left[\frac{h^n_{i+1,j,k} - h^n_{i,j,k}}{\Delta x_{i+1/2}} \right] - [\sin\theta_x K_{s,x} k_r(h)]_{i+1/2,j,k}$$

ParFlow uses an Inexact Newton method to solve this problem

- Starting with h^0 we want h^* such that $F(h^*) = 0$
- We solve the linear system $J(h^k)s^k = -F(h^k)$
- $F(h)$ is the **function evaluation**, $J(h)$ is the **jacobian**

Jones and Woodward *AWR* 2001

The Jacobian can be formulated **two ways**: Finite Difference and Analytical

```
pfset Solver.Nonlinear.UseJacobian                    True
pfset Solver.Nonlinear.UseJacobian                    False
```

- Finite difference (*False*)

$$J(h^k) \approx \frac{F(h^k + \epsilon) - F(h^k)}{\epsilon}$$

```
pfset Solver.Nonlinear.DerivativeEpsilon              1e-12
```

- Or analytical (*True*):

We can *analytically* take the derivative of this function

$$\frac{\partial F\left(h_{i,j,k}^{n}\right)}{\partial h_{i,j,k}^{n}} = \Delta x_i \Delta y_j \Delta z_k S_{s,i,j,k} \left[\frac{\partial S_w(h)_{i,j,k}^{n}}{\partial h_{i,j,k}^{n}} h_{i,j,k}^{n} + S_w(h)_{i,j,k}^{n} \right]$$
$$+ \Delta x_i \Delta y_j \Delta z_k \phi_{i,j,k} \left[\frac{\partial S_w(h)_{i,j,k}^{n}}{\partial h_{i,j,k}^{n}} \right]$$
$$- \Delta t^n \Delta x_i \Delta y_j \Delta z_k \left[\frac{\frac{\partial q_{i+1/2,j,k}^{x}}{\partial h_{i,j,k}^{n}} - \frac{\partial q_{i-1/2,j,k}^{x}}{\partial h_{i,j,k}^{n}}}{\Delta x_i} + \frac{\frac{\partial q_{i,j+1/2,k}^{y}}{\partial h_{i,j,k}^{n}} - \frac{\partial q_{i,j-1/2,k}^{y}}{\partial h_{i,j,k}^{n}}}{\Delta y_j} + \frac{\frac{\partial q_{i,j,k+1/2}^{z}}{\partial h_{i,j,k}^{n}} - \frac{\partial q_{i,j,k-1/2}^{z}}{\partial h_{i,j,k}^{n}}}{\Delta z_k} \right]$$

$$\frac{\partial q_{i+1/2,j,k}^{x}}{\partial h_{i,j,k}^{n}} = -\frac{(\cos(\theta_x) K_s)_{i+1/2,j,k}}{\Delta x_{i+1/2}} \left[\frac{\partial k_r(h)_{i+1/2,j,k}}{\partial h_{i,j,k}^{n}} (h_{i+1,j,k}^{n} - h_{i,j,k}^{n}) + k_r(h)_{i+1/2,j,k} \right]$$
$$- \left(\sin(\theta_x) K_s \frac{\partial k_r(h)}{\partial h_{i,j,k}^{n}} \right)_{i+1/2,j,k}$$

But this derivative is non-symmetric

$$\frac{\partial q^x_{i+1/2,j,k}}{\partial h^n_{i,j,k}} = -\frac{(\cos(\theta_x)K_s)_{i+1/2,j,k}}{\Delta x_{i+1/2}}\left[\frac{\partial k_r(h)_{i+1/2,j,k}}{\partial h^n_{i,j,k}}(h^n_{i+1,j,k} - h^n_{i,j,k}) + k_r(h)_{i+1/2,j,k}\right] - \left(\sin(\theta_x)K_s\frac{\partial k_r(h)}{\partial h^n_{i,j,k}}\right)_{i+1/2,j,k}$$

$$\frac{\partial q^x_{i+1/2,j,k}}{\partial h^n_{i+1,j,k}} = -\frac{(\cos(\theta_x)K_s)_{i+1/2,j,k}}{\Delta x_{i+1/2}}\left[-\frac{\partial k_r(h)_{i+1/2,j,k}}{\partial h^n_{i+1,j,k}}(h^n_{i+1,j,k} - h^n_{i,j,k}) + k_r(h)_{i+1/2,j,k}\right] + \left(\sin(\theta_x)K_s\frac{\partial k_r(h)}{\partial h^n_{i,j,k}}\right)_{i+1/2,j,k}$$

The *kinsol* log file has information about solver performance that ties to the solution approach

How many *k*'s?

How big is F(h)?

How many F's?

```
KINSOL starting step for time 100.000000
scsteptol used:         1e-07
fnormtol  used:         1e-06
KINSolInit nni=    0  fnorm=       0.0054603604964216 8  nfe=     1
KINSol nni=    1 fnorm=       0.0017423844768606 79 nfe=     2
KINSol nni=    2 fnorm=       0.0001551603719722471 nfe=     3
KINSol nni=    3 fnorm=       1.9728351791287 6e-08 nfe=     4
KINSol return value 1
---KINSOL_SUCCESS

-----------------------------------------------
                    Iteration             Total
Nonlin. Its.:               3                77
Lin. Its.:                  9               445
Func. Evals.:               4               191
PC Evals.:                  3                77
PC Solves:                 12               522
Lin. Conv. Fails:           0                 0
Beta Cond. Fails:           0                 0
Backtracks:                 0                 0
-----------------------------------------------
```

Solver settings that control nonlinear convergence

```
pfset Solver.Nonlinear.MaxIter                    100
pfset Solver.Nonlinear.ResidualTol                1e-6
```

```
KINSOL starting step for time 100.000000
scsteptol used:         1e-07
fnormtol  used:         1e-06
KINSolInit nni=    0  fnorm=       0.005460360496421668  nfe=     1
KINSol nni=    1 fnorm=       0.0017423844768606 79 nfe=     2
KINSol nni=    2 fnorm=       0.0001551603719722471 nfe=     3
KINSol nni=    3 fnorm=       1.972835179128746e-08 nfe=     4
KINSol return value 1
---KINSOL_SUCCESS
```

This will be less than ResidualTol

This will stop at MaxIter

You can interpret solver performance from the *kinsol* log file

	Iteration	Total
Nonlin. Its.:	3	77
Lin. Its.:	9	445
Func. Evals.:	4	191
PC Evals.:	3	77
PC Solves:	12	522
Lin. Conv. Fails:	0	0
Beta Cond. Fails:	0	0
Backtracks:	0	0

Linear Iterations

How many F's?

The out.log file has timing information

Total Nonlinear Solver Time

```
KINSol:
  wall clock time   = 35.849100 seconds
  wall MFLOPS = 1879.165739 (6.73664e+10)
```

Function Evaluation Time

```
NL_F_Eval:
  wall clock time   = 3.322800 seconds
  wall MFLOPS = 0.000048 (159)
```

Linear Iteration Time

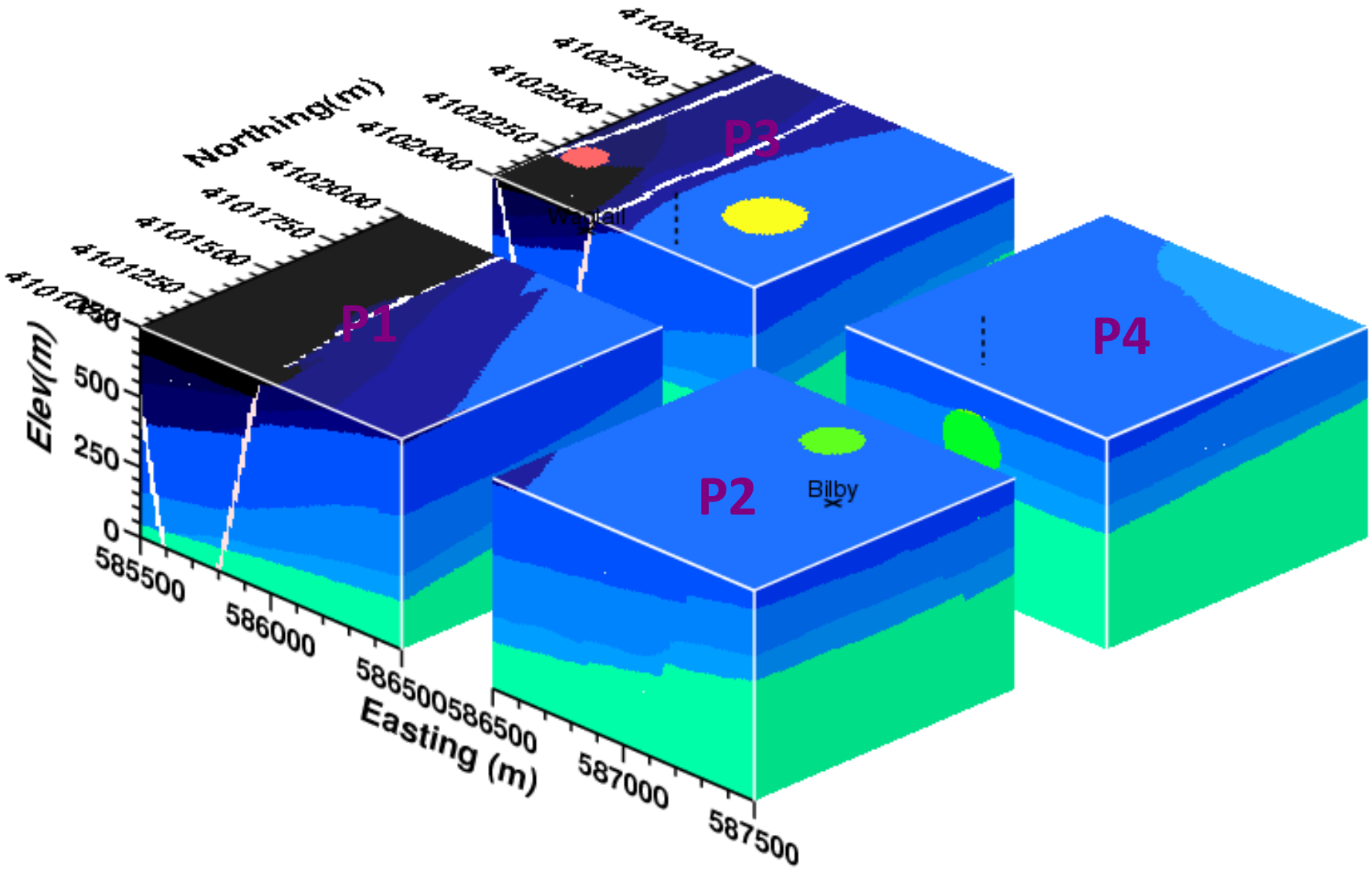
```
PFMG:
  wall clock time   = 11.003900 seconds
  wall MFLOPS = 0.000000 (0)
```

There are different preconditioners used in the *Linear* solve of the *Nonlinear* system

```
pfset Solver.Linear.Preconditioner                PFMG
pfset Solver.Linear.Preconditioner                PFMGOctree
pfset Solver.Linear.Preconditioner                MGSemi
pfset Solver.Linear.Preconditioner                SMG
```

- **PFMG** ParFlow MultiGrid
- **PFMGOctree** PFMG for domains with lots of inactive cells
- **MGSemi** Internal multigrid, semi coarsening
- *PFMG, MGSemi are* robust, light-weight multigrid preconditioners for use in heterogeneous domains (Ashby and Falgout, Nuclear Science & Eng., 1996)
- **SMG** robust multigrid solver for anisotropic and heterogeneous domains (Schaffer, SIAM J. on Scientific Computing, 1998)

Parallelization



Parallel performance goes hand in hand with Solver Performance

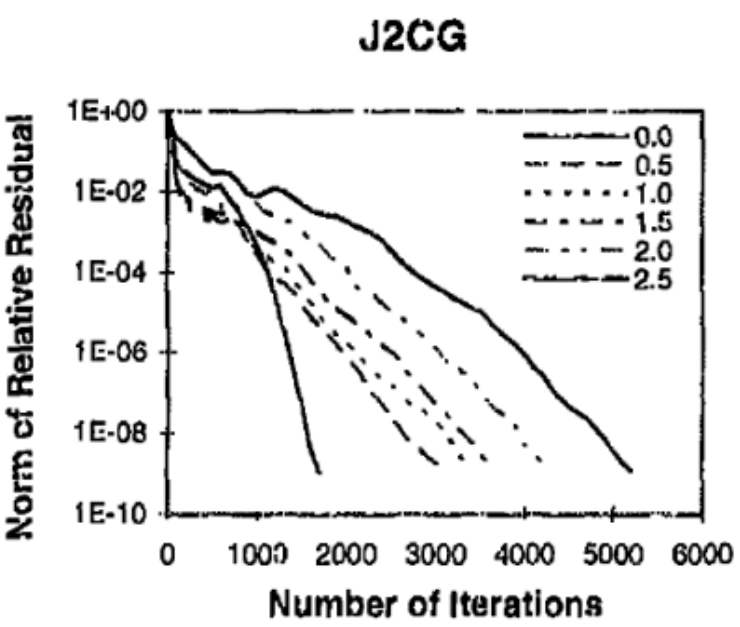
- Performance and parallel performance are intricately linked
- To get good parallel performance the numerical algorithm must scale linearly with problem size
- If we want to run large problems and our solver does not scale parallel performance will not be sustained

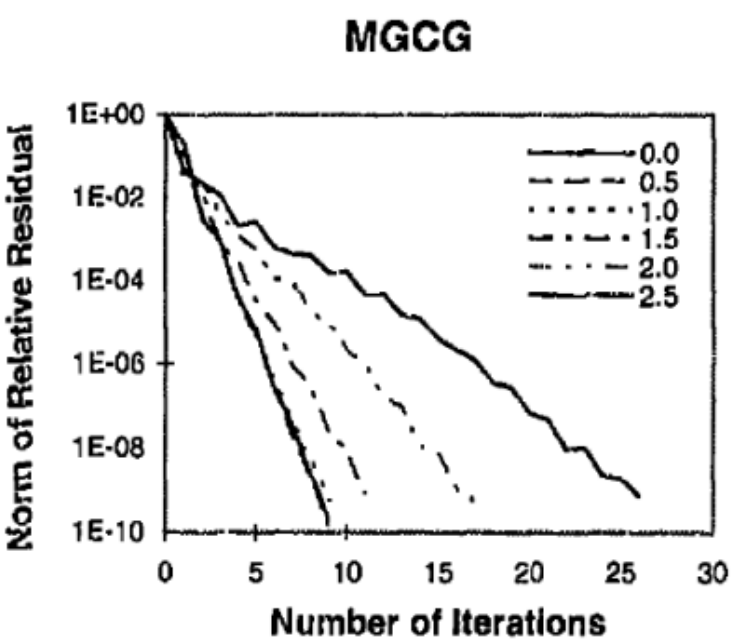
Performance: Making the problem "harder"

Ashby and Falgout (1996)

TABLE 5.5
Varying the degree of heterogeneity.

Heterogeneity		J2CG		MGCG		MG	
σ	σ_K^2	iters	time	iters	time	iters	time
0.0	0×10^0	1701	354.4	9	10.4	13	12.8
0.5	6×10^0	3121	650.3	9	10.4	13	12.8
1.0	7×10^1	3388	705.7	9	10.4	12	11.8
1.5	1×10^3	3670	764.6	11	12.5	22	21.6
2.0	4×10^4	4273	889.5	17	18.8	diverged	
2.5	4×10^6	5259	1094.4	26	28.2	diverged	





Performance: Making the problem bigger

TABLE 5.4

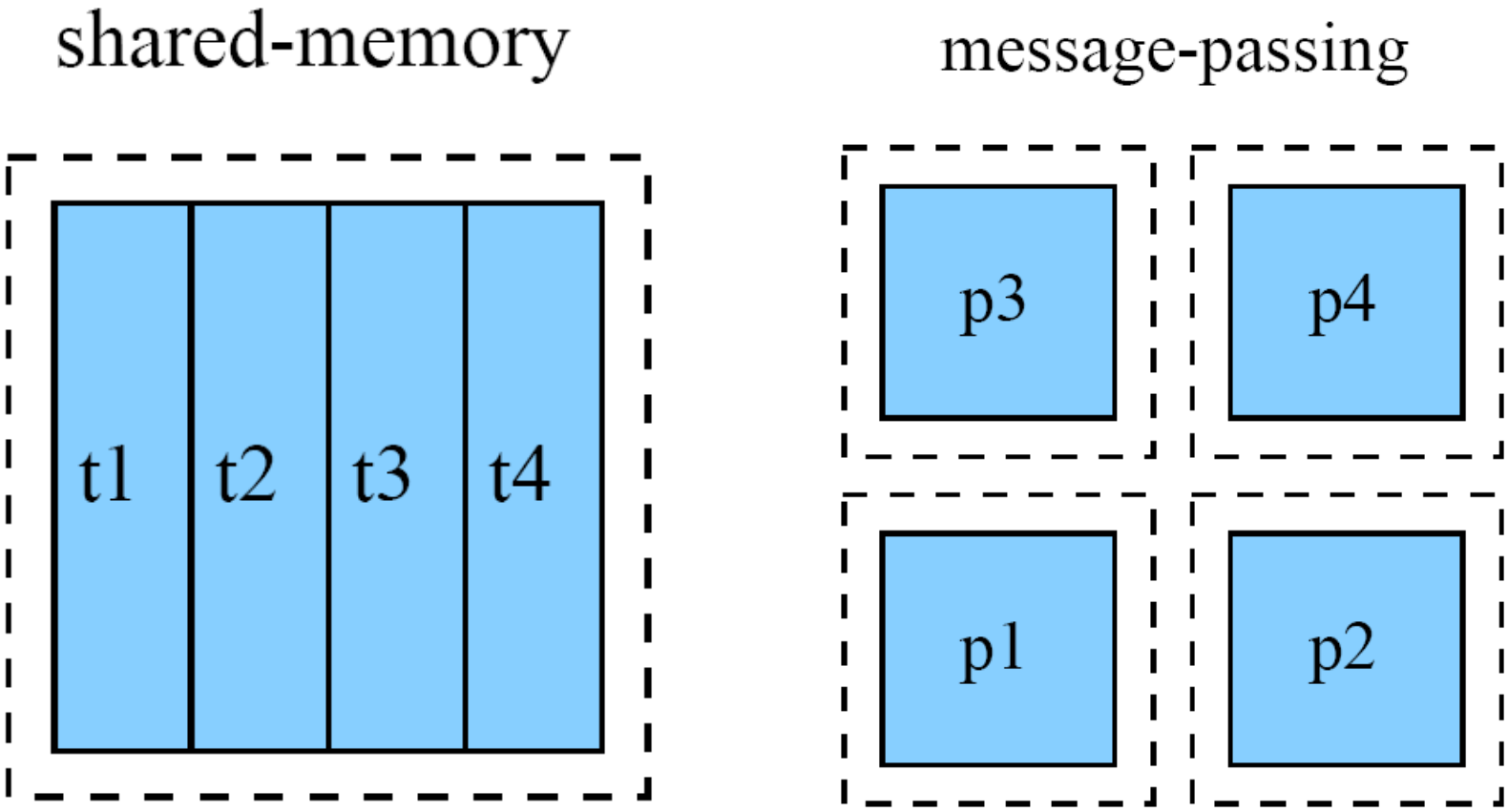
Enlarging the domain size: the grid spacing is fixed while the number of grid points is increased.

Problem Size			J2CG		MJCG		MGCG		MG	
n_x	n_y	n_z	iters	time	iters	time	iters	time	iters	time
17	17	9	453	1.1	11	0.3	9	0.4	12	0.4
33	33	17	957	5.7	13	0.5	10	0.7	14	0.9
65	65	33	1860	56.0	16	2.0	10	2.1	19	3.6
129	129	65	3665	763.4	18	12.1	11	12.6	21	20.6
257	257	129	6696	*1403.8	NA		13	*15.1	22	*22.8

*These times are for 256 processors ($P = 4 \times 8 \times 8$)

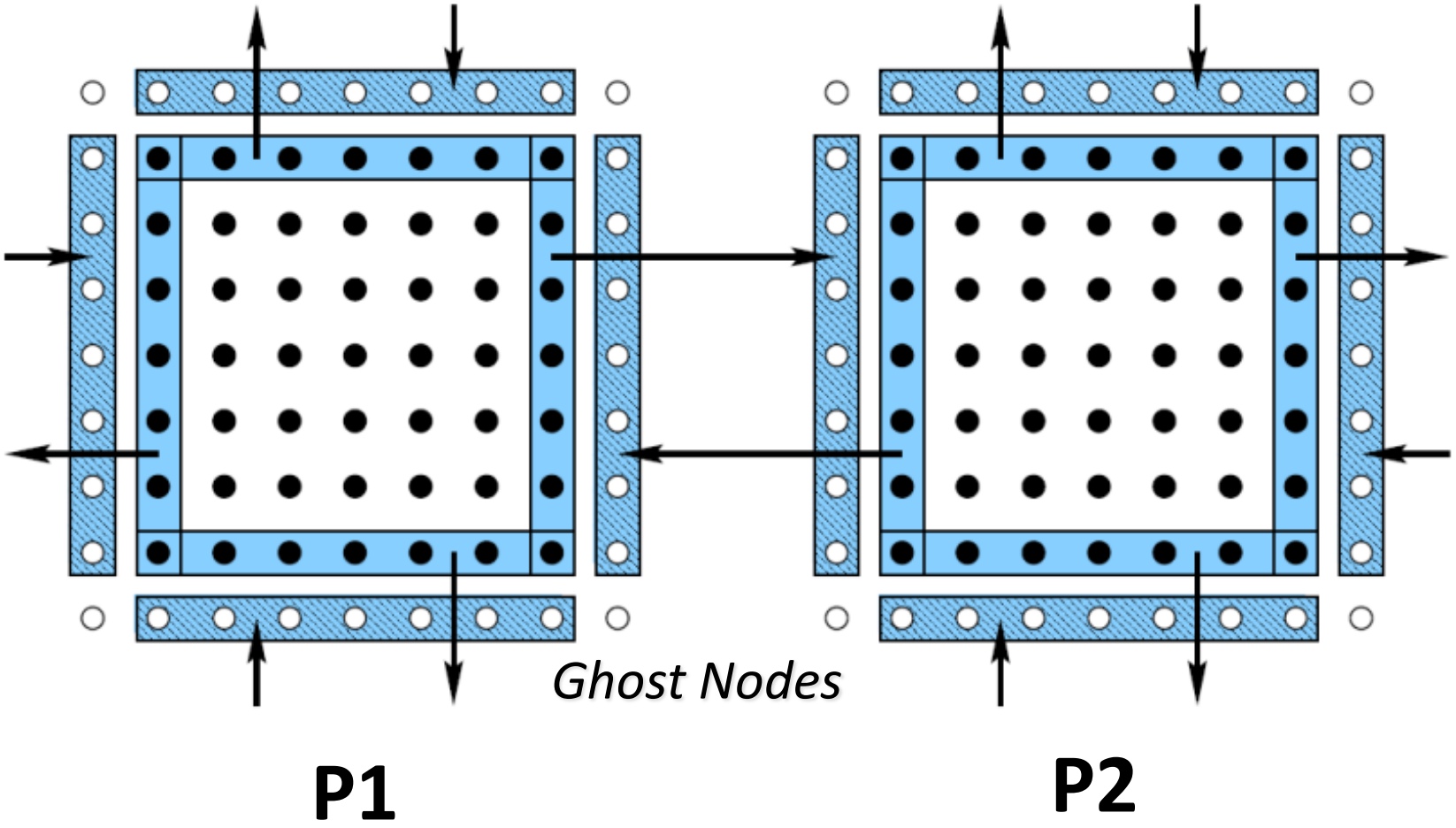
Ashby and Falgout (1996)

Parallelization



Falgout and Jones (1999)

Parallelization- Distributed Memory



Falgout and Jones (1999)

Scaled Parallel Efficiency- Scaled Speedup

Scaled parallel efficiency, *E*, is defined as the ratio of time to run a problem of varying size as we keep the per-processor work constant

$$E(n,p) = \frac{T(n,1)}{T(pn,p)}$$

T = run time

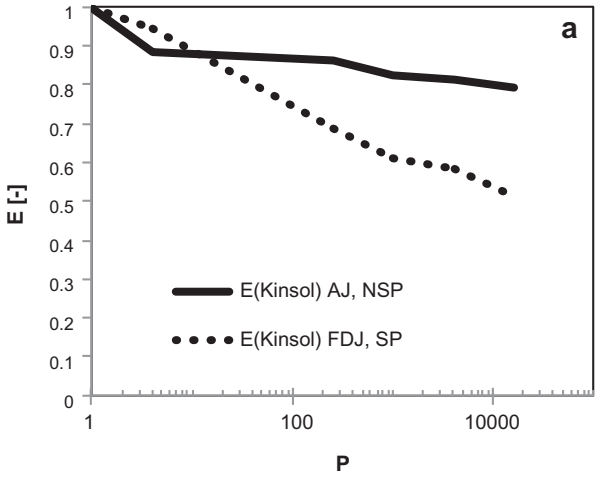
n = problem size

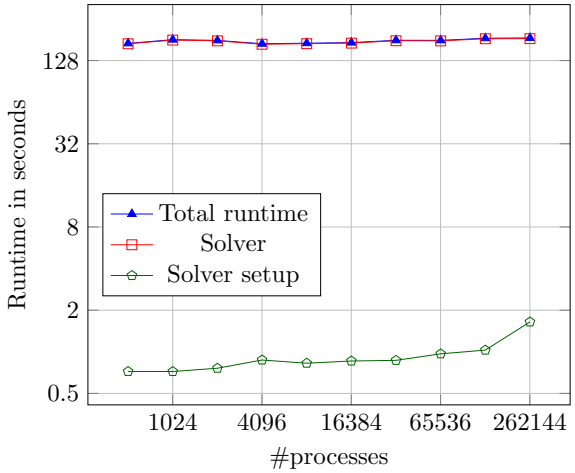
p = number of processors

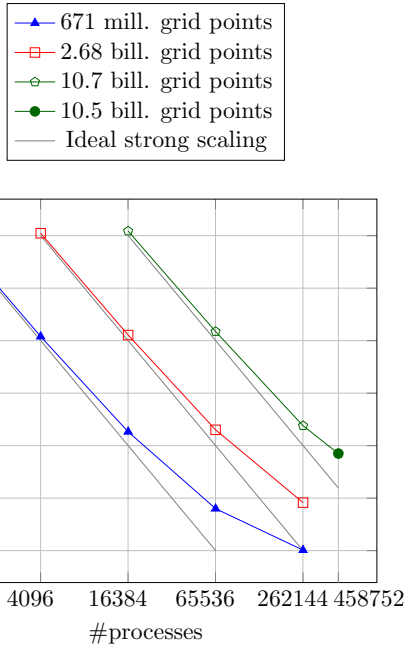
ParFlow demonstrates excellent parallel scalability

***Weak scaling**, where the problem size increases with the number of processors*

***Strong scaling**, where the problem size remains constant*



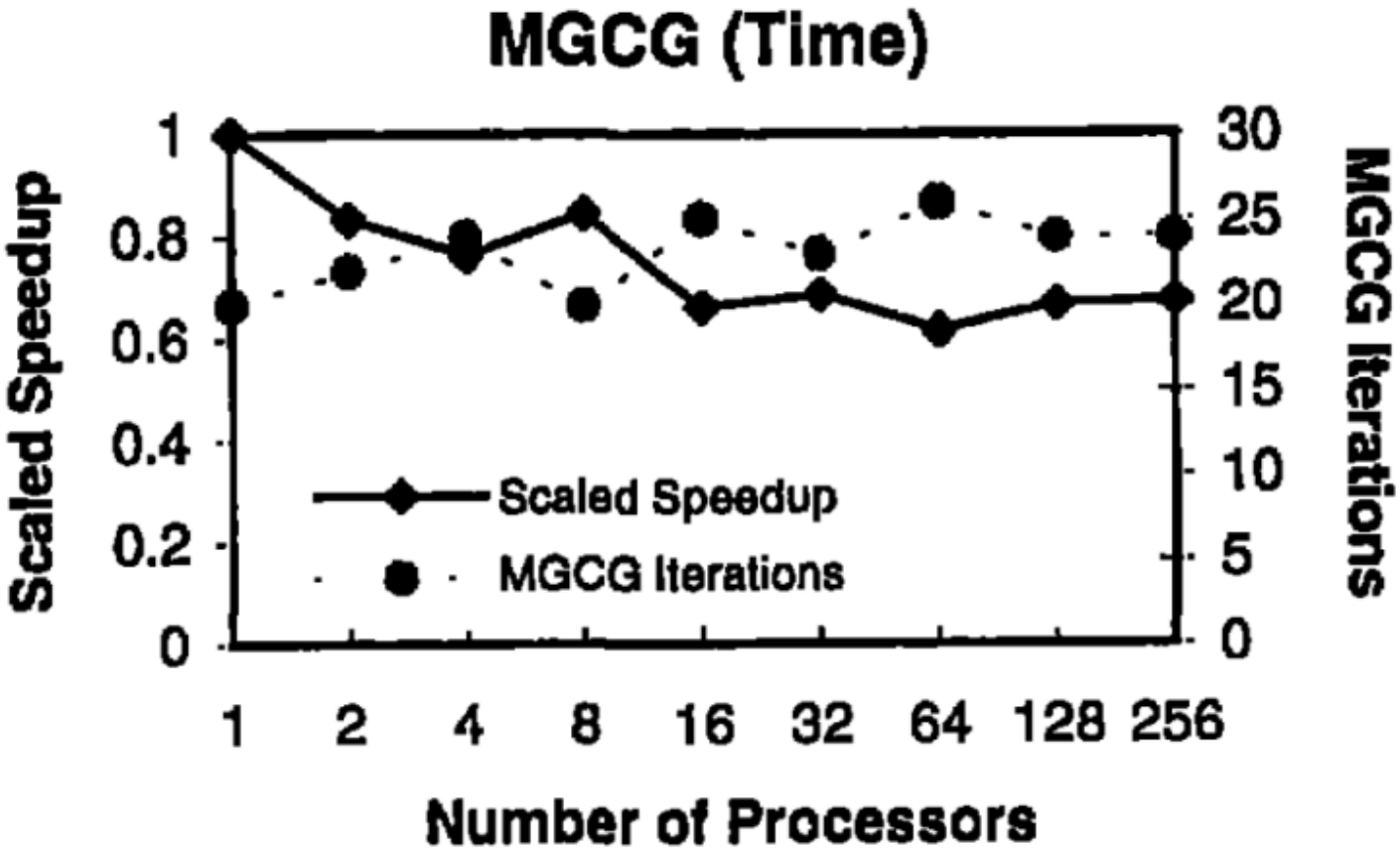




Maxwell, AWR 2013

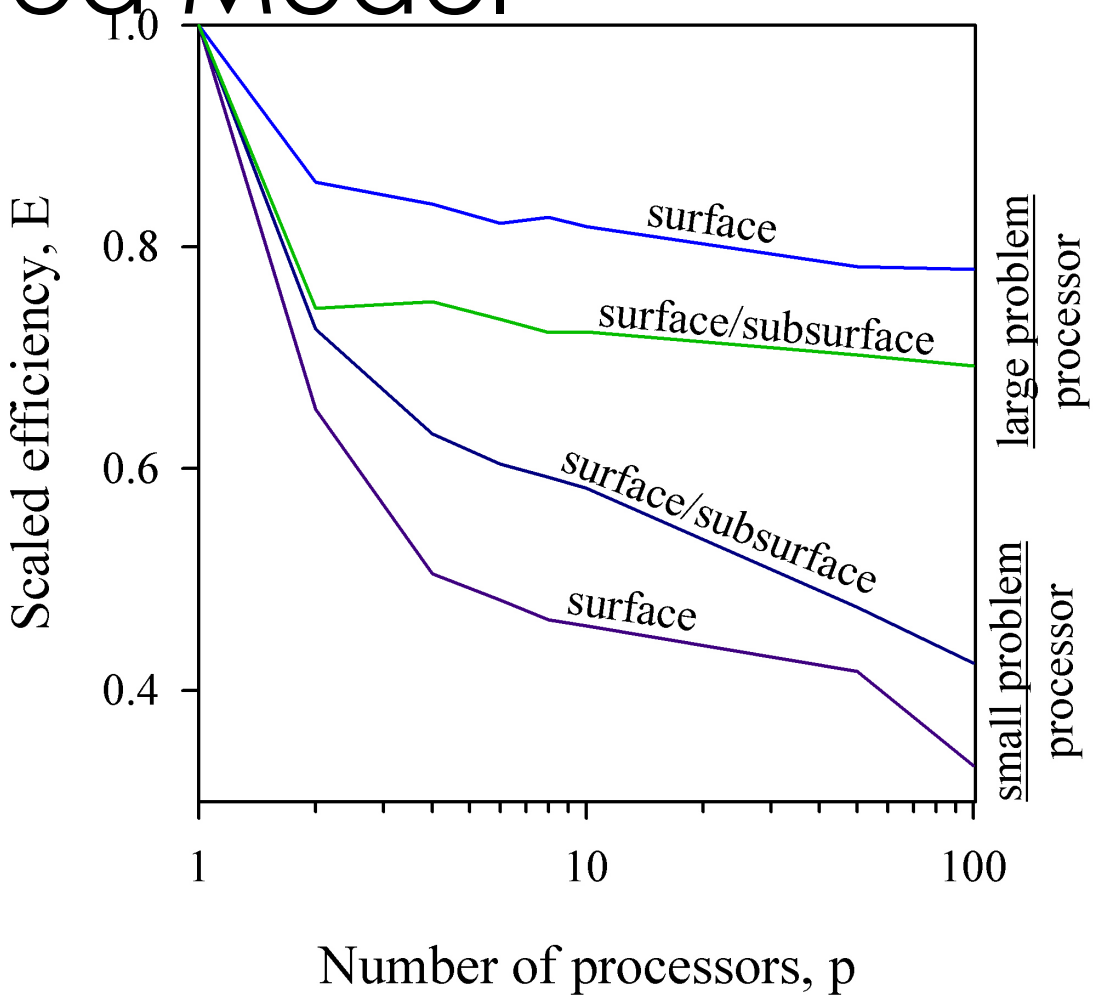
Burstedde, Fonseca, Kollet, CompGeo 2018

Parallel Performance: Scaled Speedup of the Linear Problem



Ashby and Falgout (1996)

Scaled Parallel Efficiency of Coupled Model



Perfect efficiency: double problem size and processor # same run time => E = 1

Kollet and Maxwell, AWR (2006)