A Parallel Implementation of Algebraic Multigrid

Robert D. Falgout

Van Emden Henson

Jim E. Jones

Ulrike Meier Yang

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

March 22, 1999

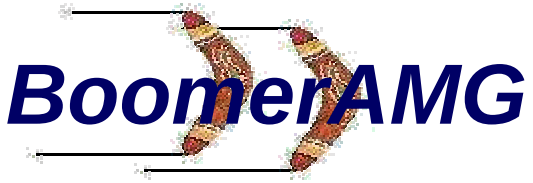
AMG has two phases:

- **Setup Phase**
  - Select Coarse "grids," $\Omega^{m+1}, m = 1, 2, \ldots$
  - Define **interpolation**, $I_{m+1}^{m}, \ m = 1, 2, \ldots$
  - Define **restriction** and **coarse-grid operators**

$$I_m^{m+1} = (I_{m+1}^m)^T \qquad A^{m+1} = I_m^{m+1} A^m I_{m+1}^m$$

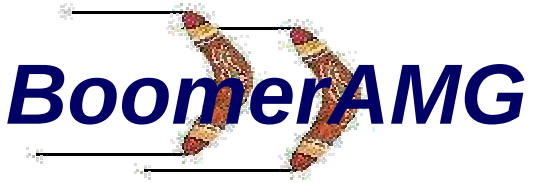
- **Solve Phase**
  - Standard multigrid operations, e.g., V-cycle, W-cycle, FMG, etc
- **Note:** Only the selection of coarse grids does not parallelize well using existing techniques!

We must parallelize these steps:



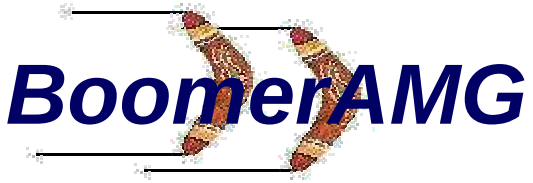
- The Setup Phase
  - Coarse Grid Selection
  - Construction of Prolongation operator, P
  - Construction of coarse-grid operators by Galerkin method, RAP, R=P'
- The Solve Phase
  - Residual Calculation
  - Relaxation
  - Prolongation
  - Restriction

Parallelizing the Solve Phase



- The Solve Phase
  - Residual Calculation
    - entails Axpy matvec: y<-aAx+by.
    - Relaxation: use hybrid Jacobi-Gauß-Seidel (Jacobi for off-processor data, GS for on-processor data)
  - Prolongation
    - requires Matvec
  - Restriction
    - requires MatvecT

Basic concept: Smooth error means "small" residuals



- **Error that is slow to converge obeys:**

$$e^{k+1} = (I - Q^{-1}A)\, e^{k} \text{; hence } (I - Q^{-1}A)\, e \approx e$$

$$\Rightarrow \quad Q^{-1}A\, e \approx 0 \quad \Rightarrow \quad r \approx 0$$

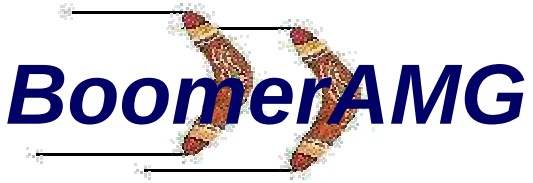
- **Define: *i depends on j* (and *j influences i*) if**

$$-a_{ij} \geq \theta \max_{k \neq i} \{-a_{ik}\}, \qquad 0 < \theta \leq 1$$

- **The set of dependencies of *i is given by***

$$S_i = \left\{ j : -a_{ij} > \theta \max_{j \neq i} -a_{ij} \right\}$$

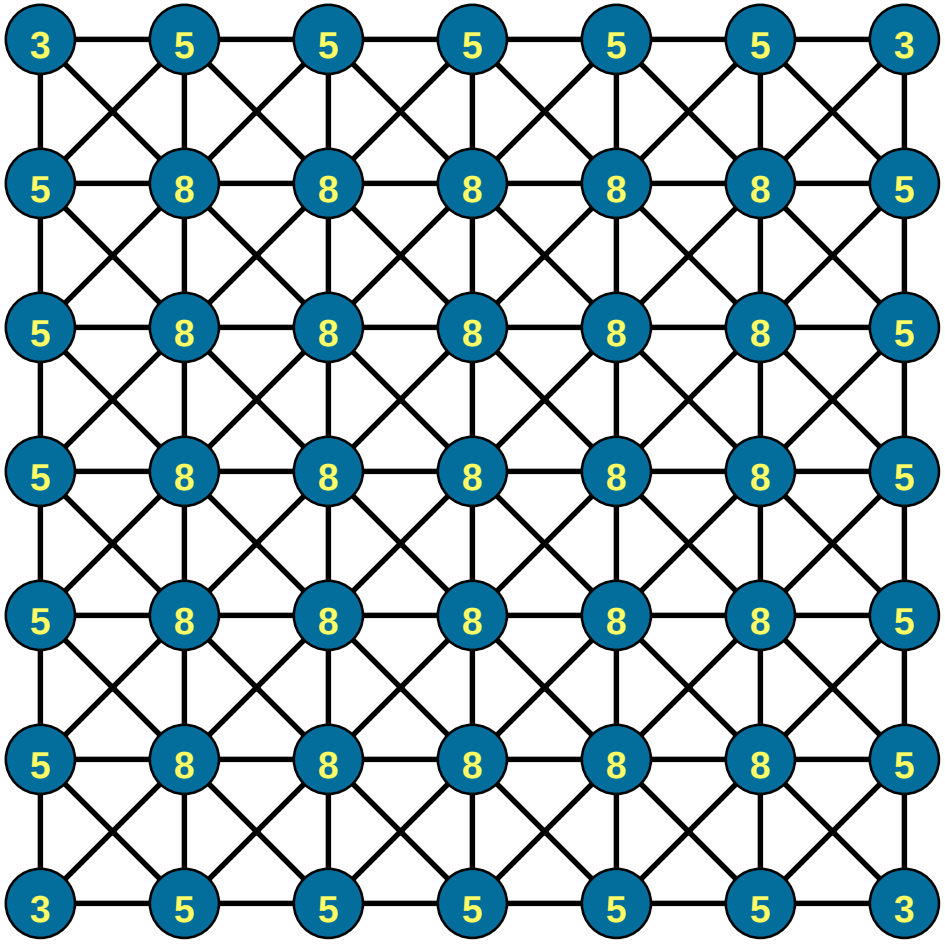
Choosing the Coarse Grid



- **Two Criteria**
  - **(C1) For each $i \in F$, each point $j \in S_i$ should either be in C or should be strongly connected to at least one point in C_i**
  - **(C2) C should be a maximal subset with the property that no two C-points are strongly connected to each other.**
- **Satisfying both (C1) and (C2) is sometimes impossible. We use (C2) as a guide while enforcing (C1).**

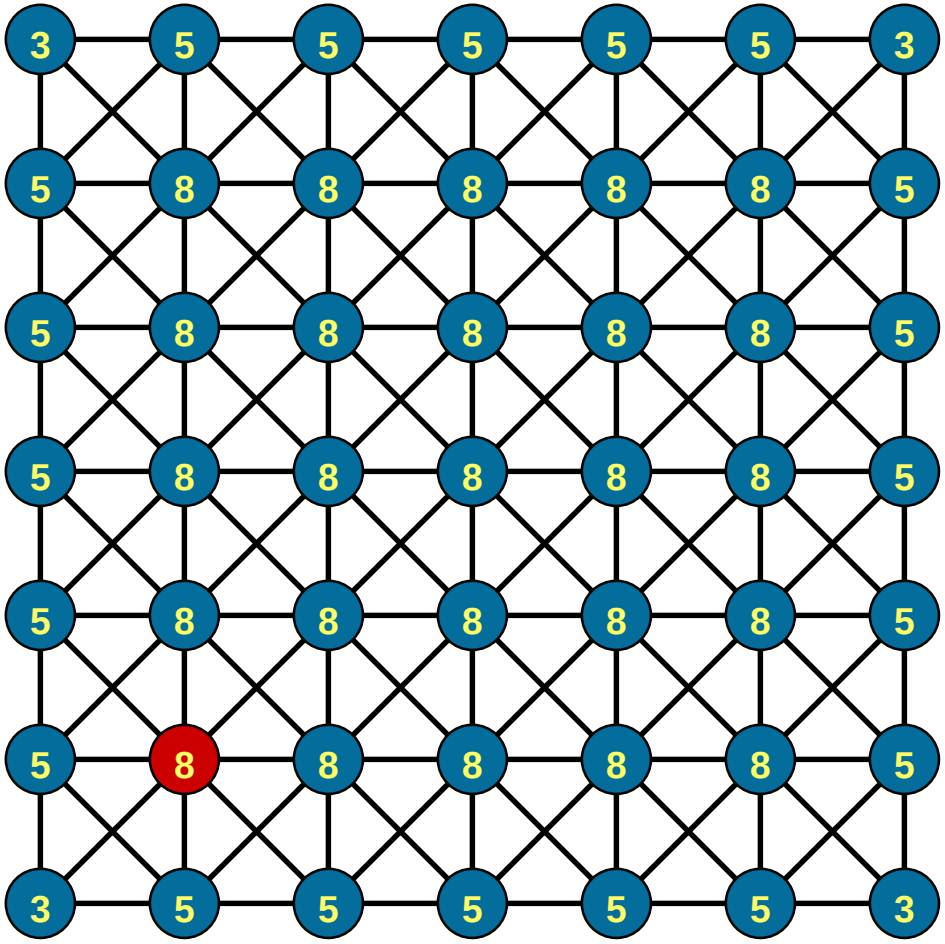
Ruge AMG: start

BoomerAMG



- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

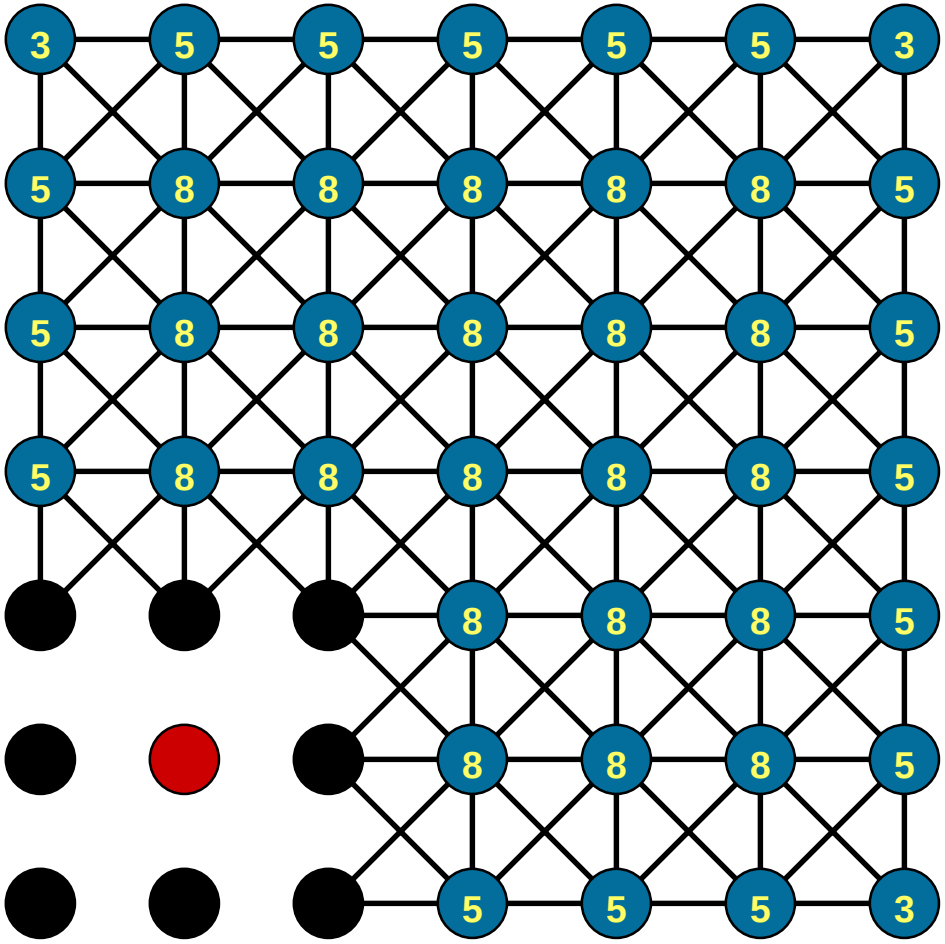
Ruge AMG: select C-pt 1



- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select F-pt 1

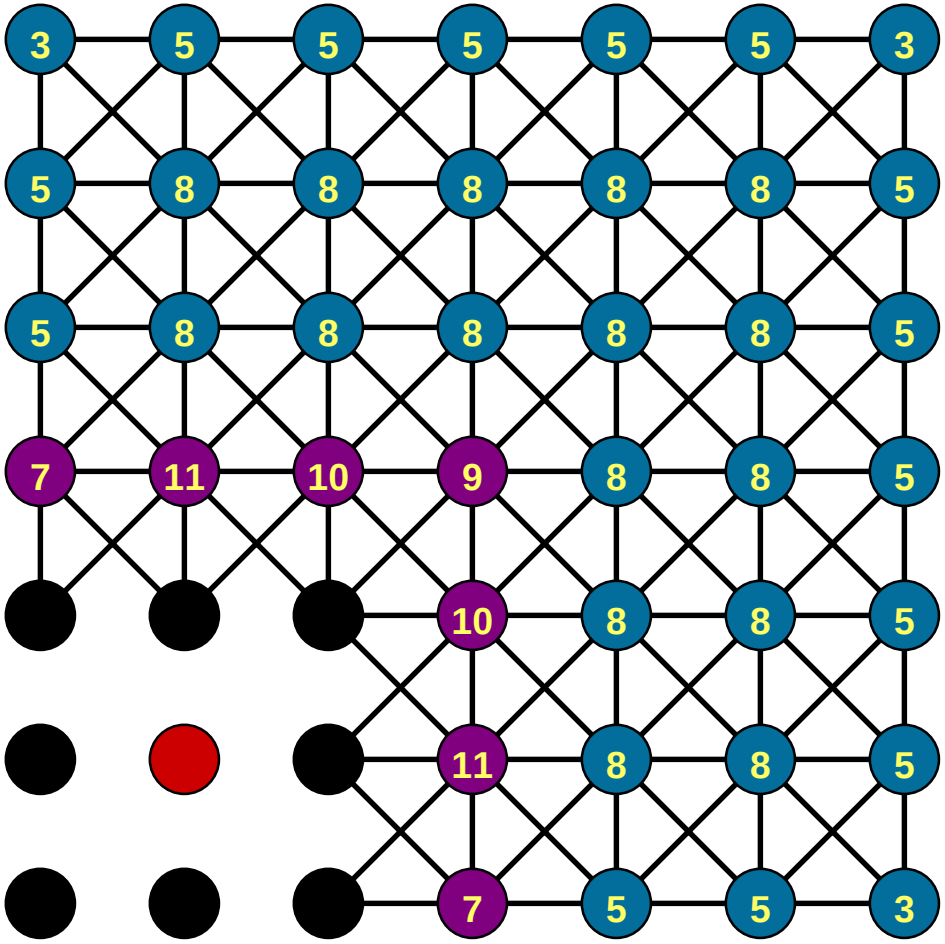
BoomerAMG



- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

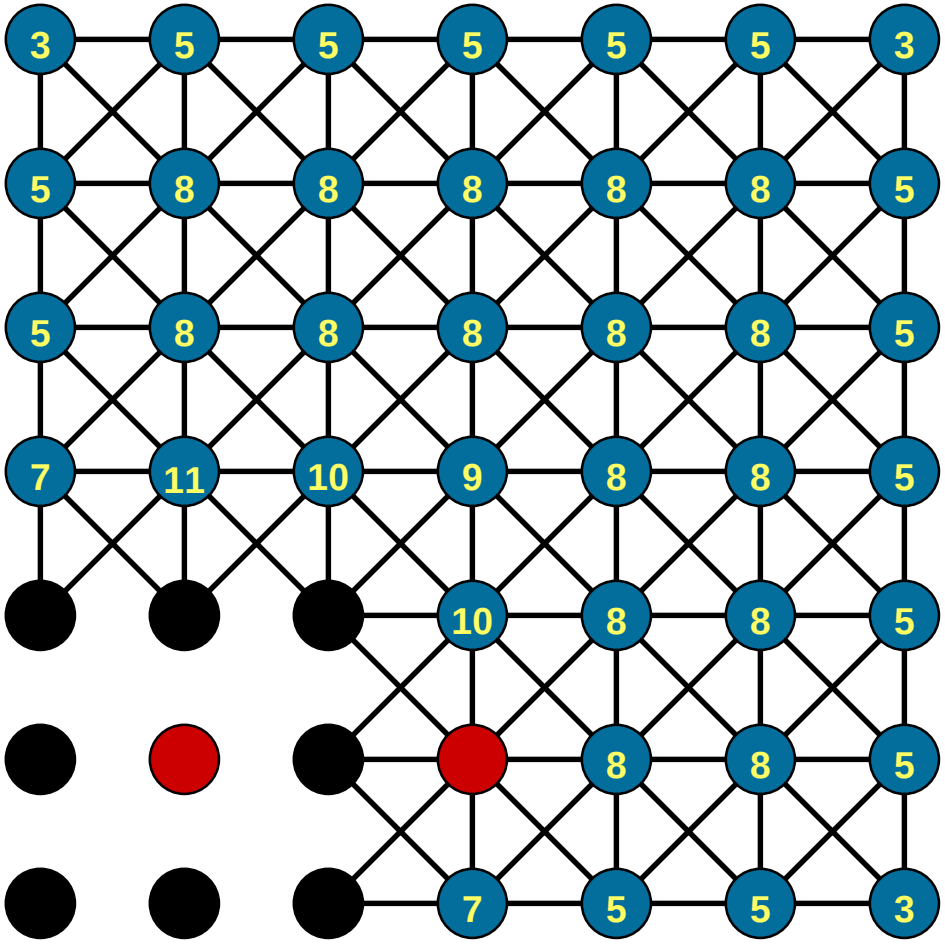
Ruge AMG: update F-pt neighbors 1

BoomerAMG



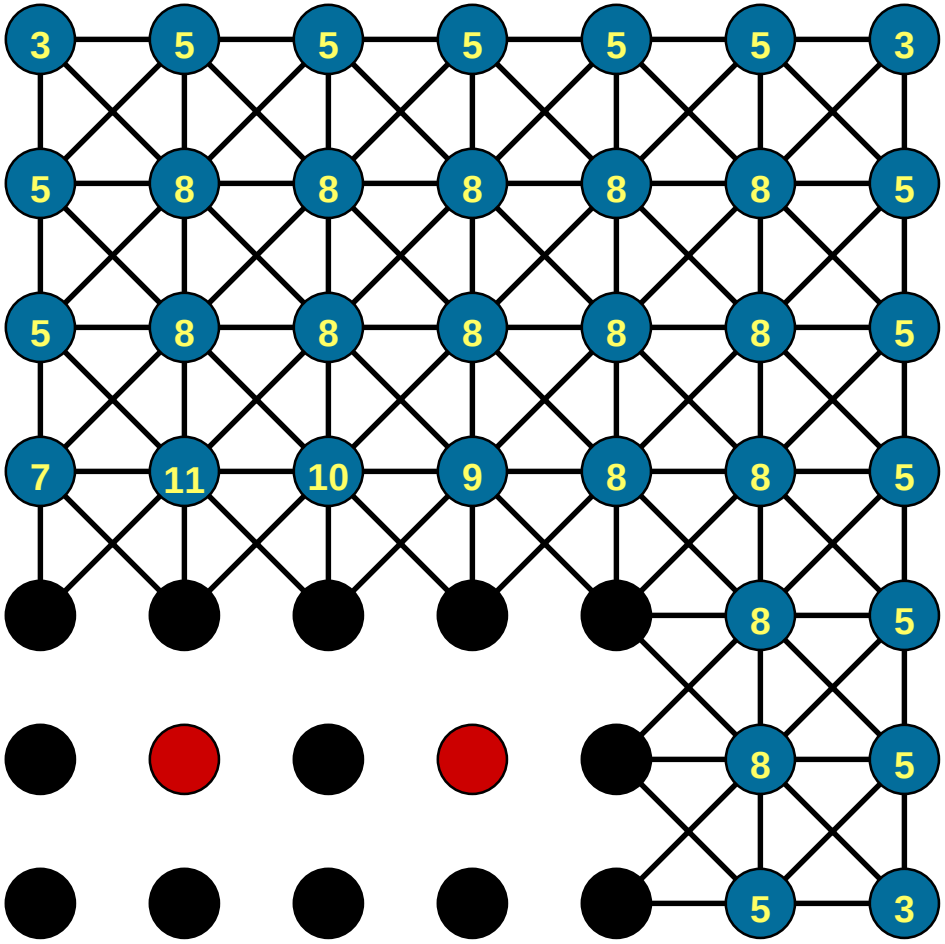
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select C-pt 2



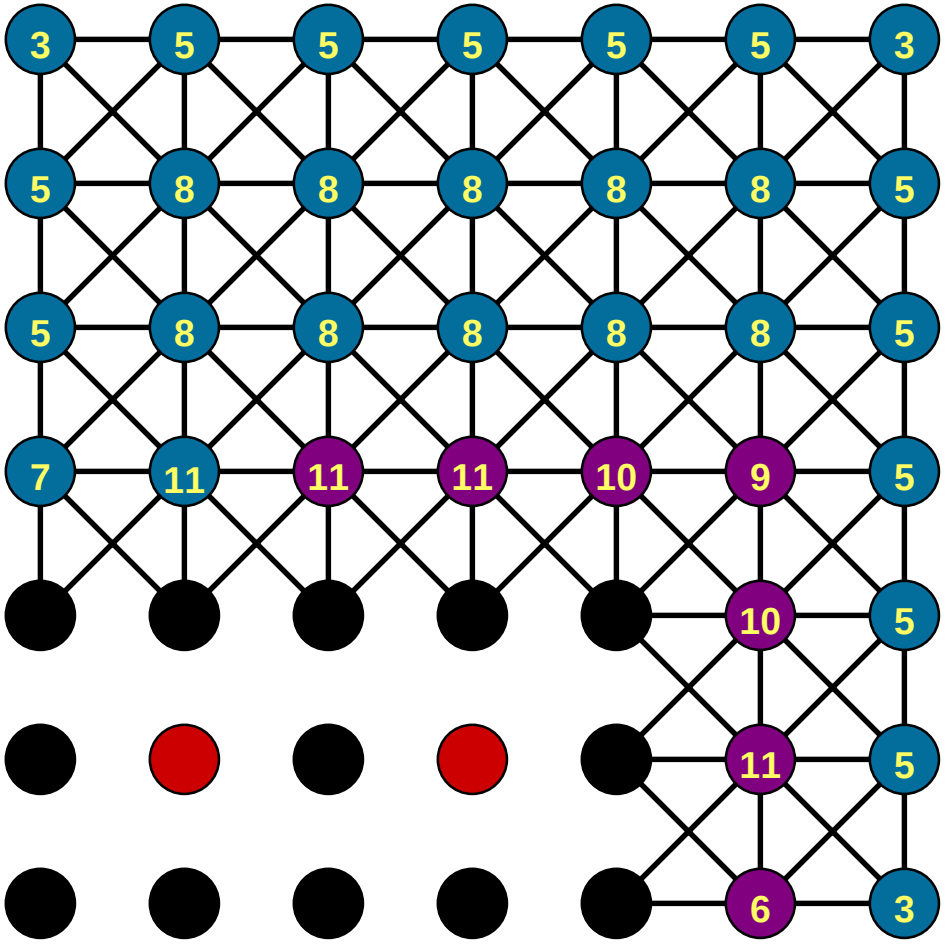
- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select F-pt 2



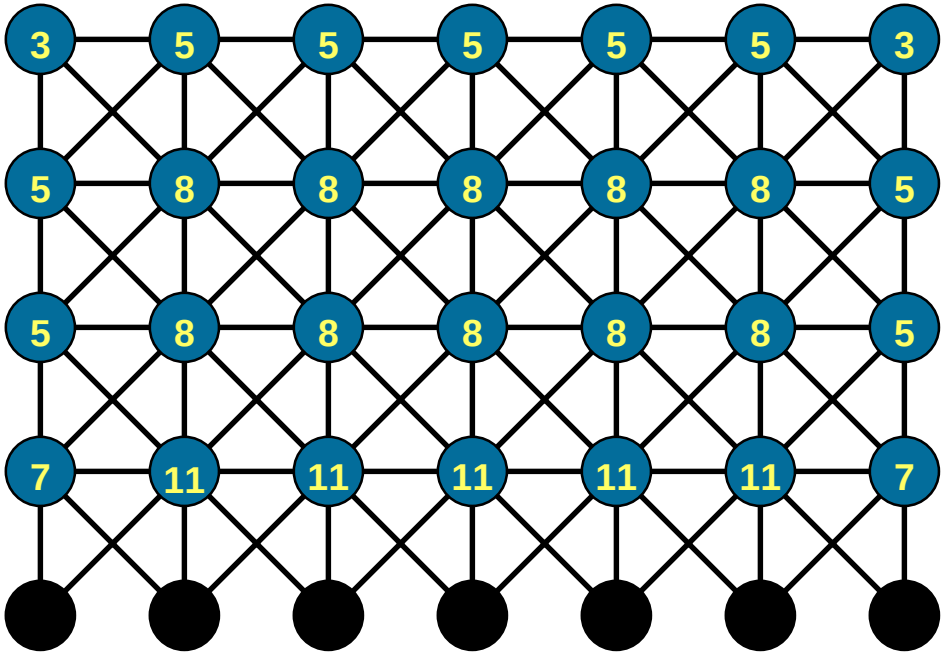
- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: update F-pt neighbors 2



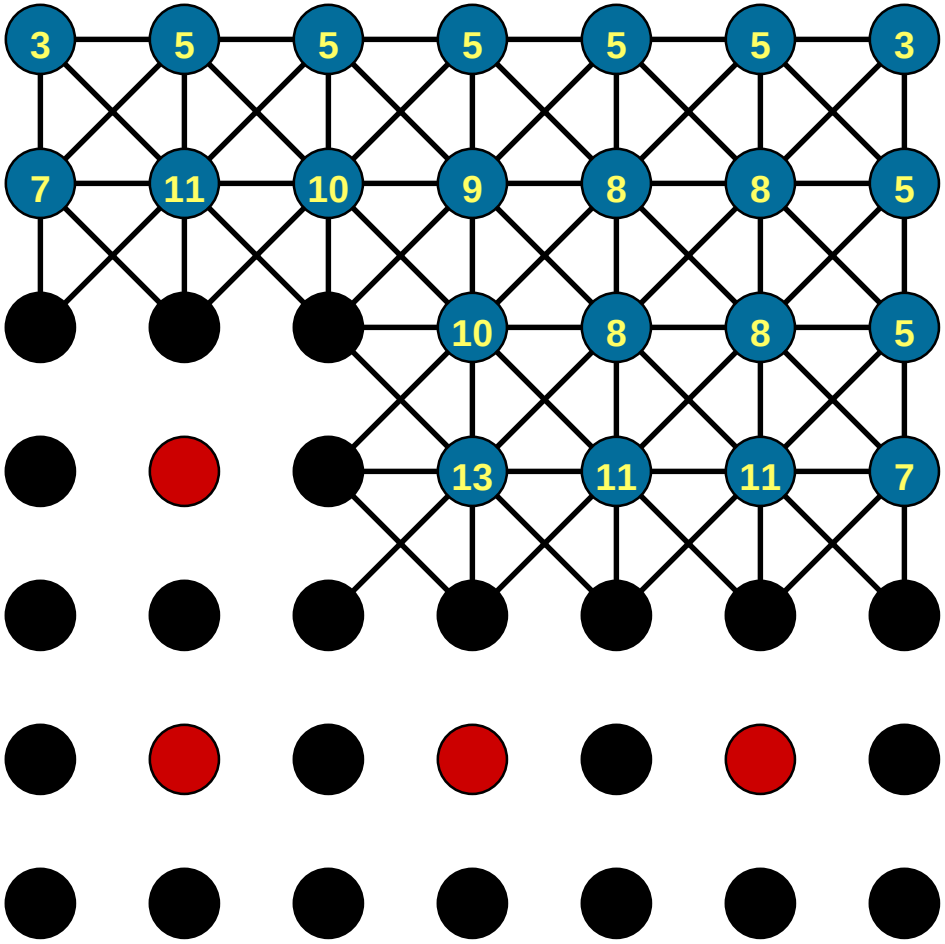
- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select C-pt, F-pts, update neighbors 3



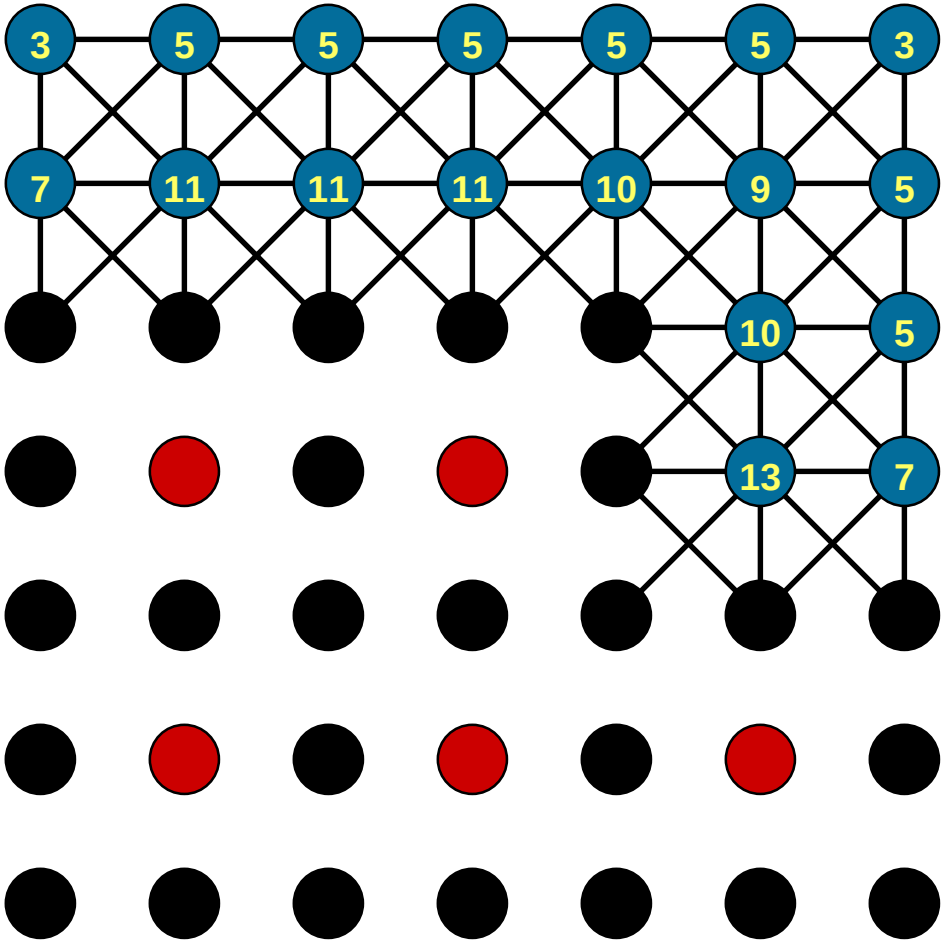
- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select C-pt, F-pts, update neighbors 4



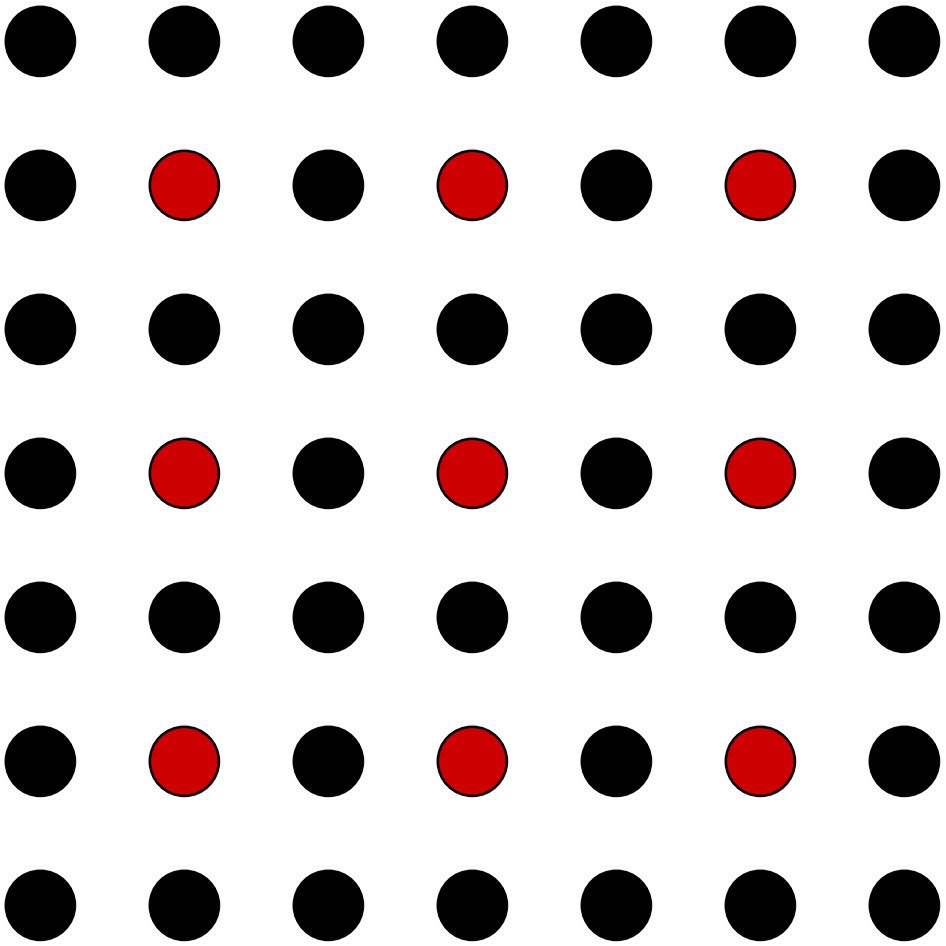
- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

Ruge AMG: select C-pt, F-pts, update neighbors 5



- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

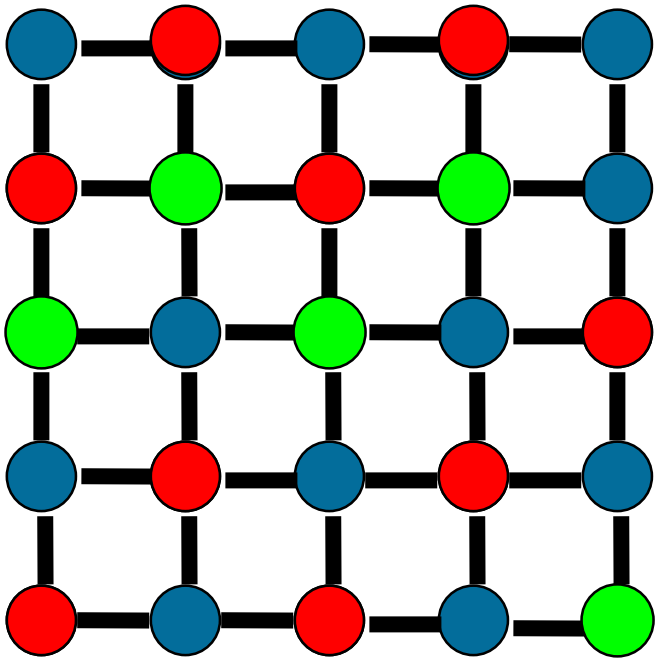
Ruge AMG: select C-pt, F-pts, update neighbors 6,7,8,9



- select next C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

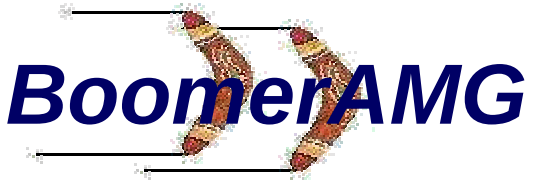
A second pass is needed to enforce (C1)

BoomerAMG



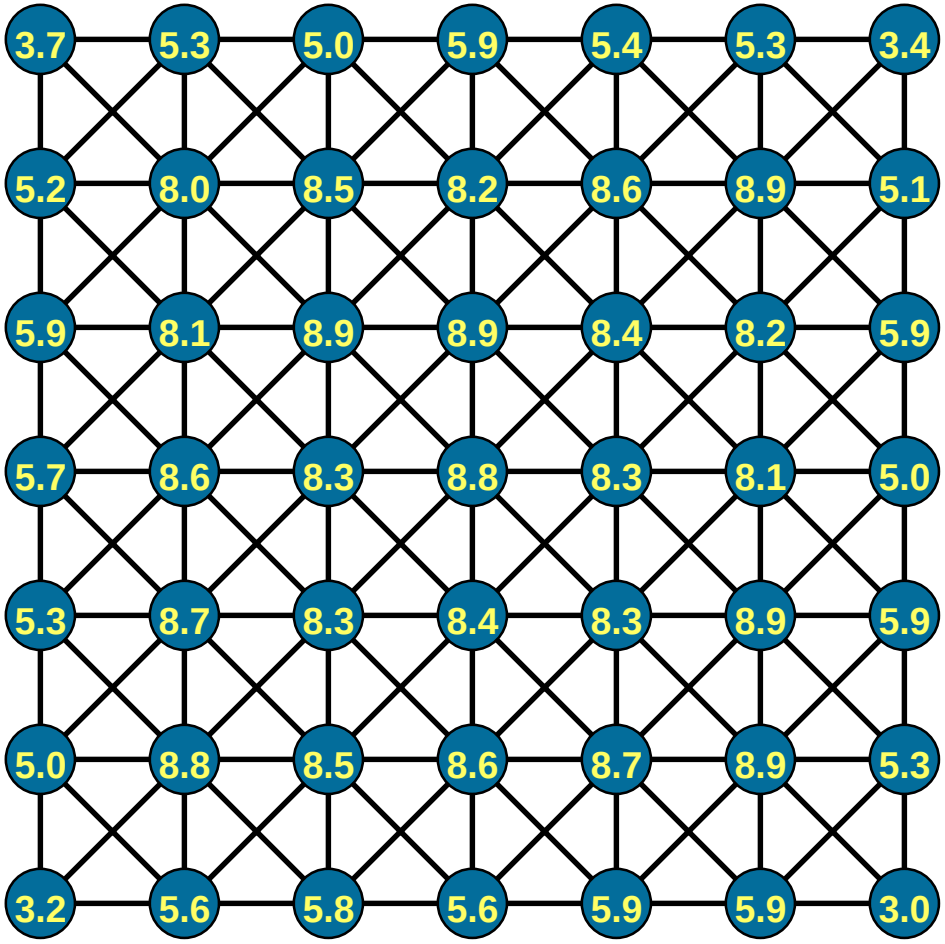
- First-pass coarsening of 5 point Laplacian , periodic boundary conditions
- Numerous *F-F dependencies among points not sharing common C-point*
- A second “coloring” pass is made, changing *F*-points to *C*-points, as needed, to ensure (C1).

A new approach: the Cleary-LJP algorithm



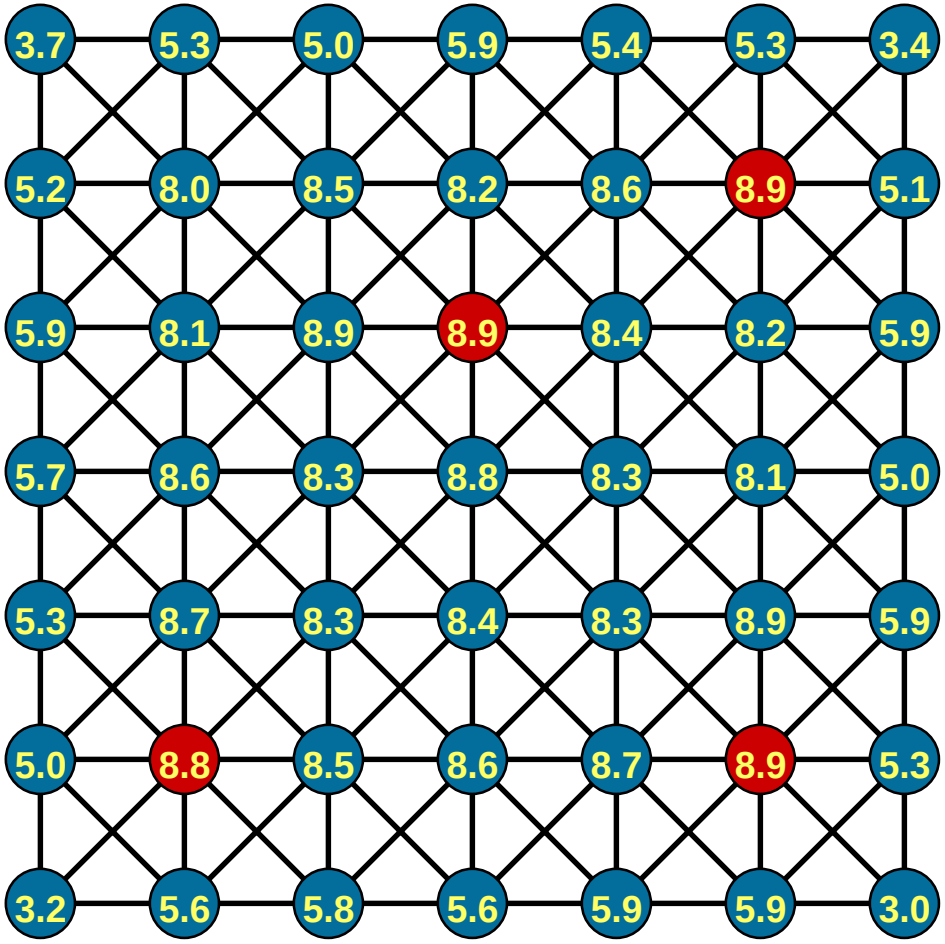
- The Ruge algorithm is inherently sequential.
- A new algorithm was proposed by Andrew Cleary , following parallel-independent-set algorithms developed by Luby and later by Jones & Plasssman
- Resulting coarsening algorithm (Cleary-LJP) is fully parallel, independent of the number of processors or processor topology. Serial prototype early 98, parallel code late 98.

Cleary-LJP start



- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

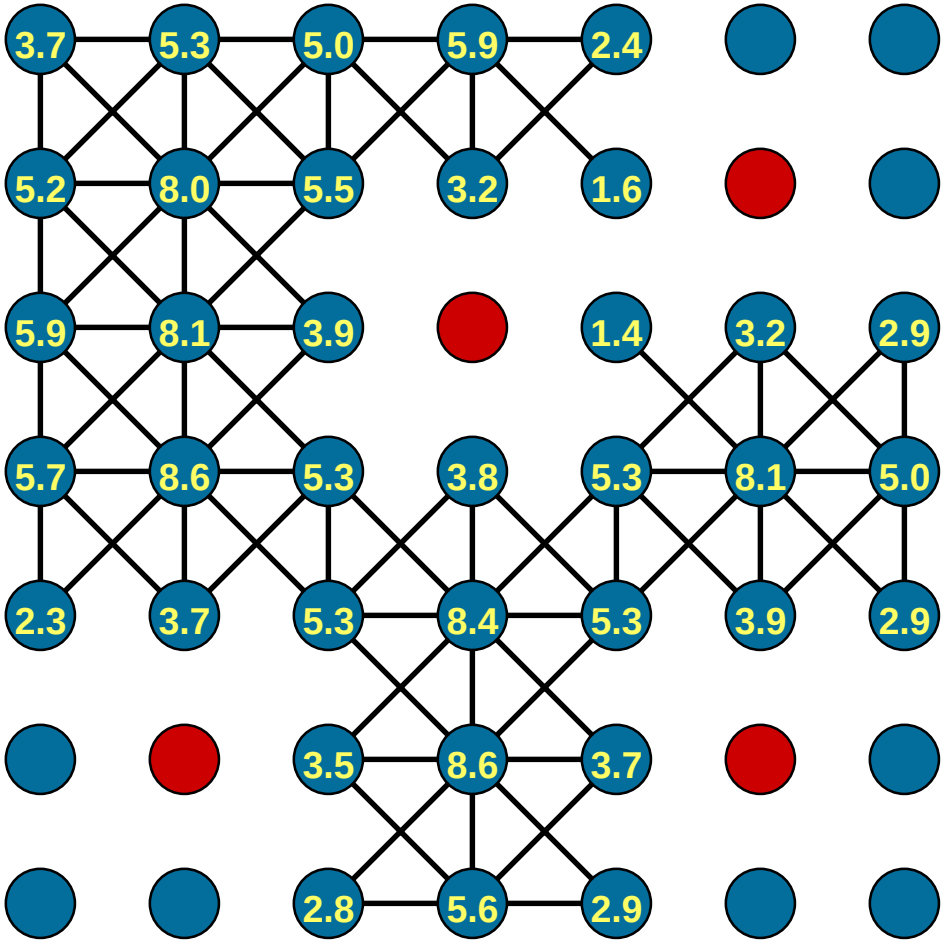
Cleary-LJP select 1



- **select C-pts with maximal measure locally**
- **remove neighbor edges**
- **update neighbor measures**

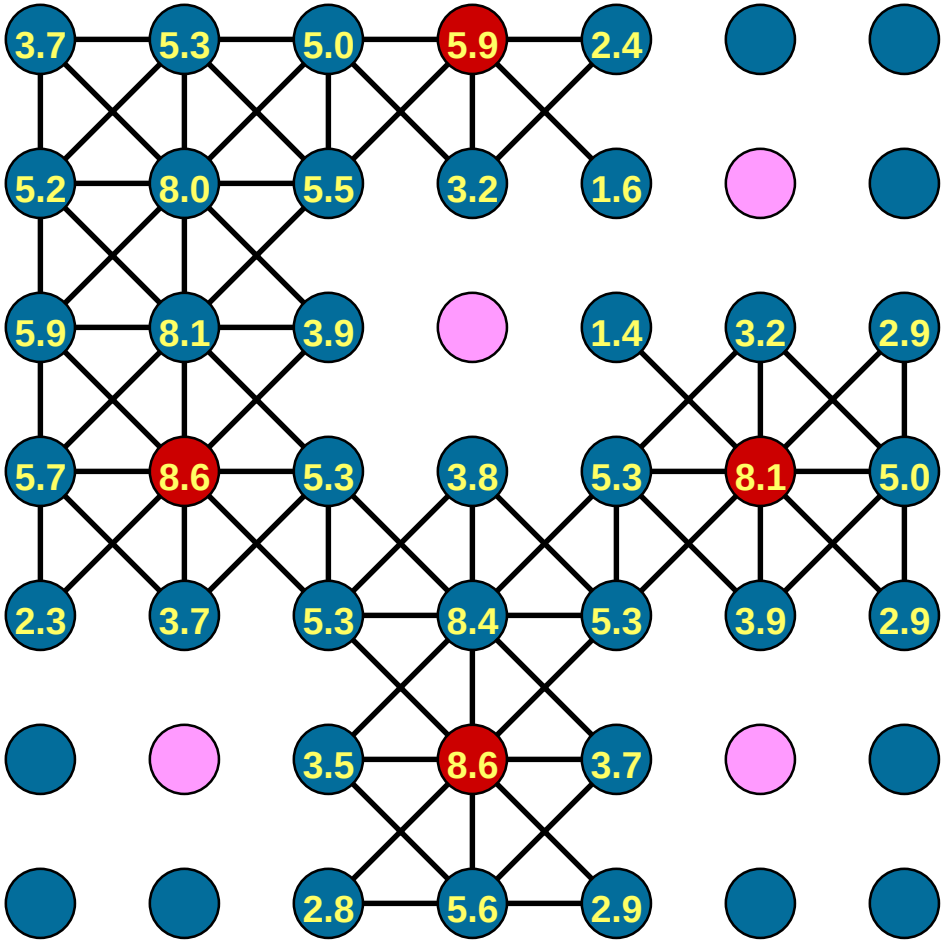
Cleary-LJP: remove and update 1

BoomerAMG



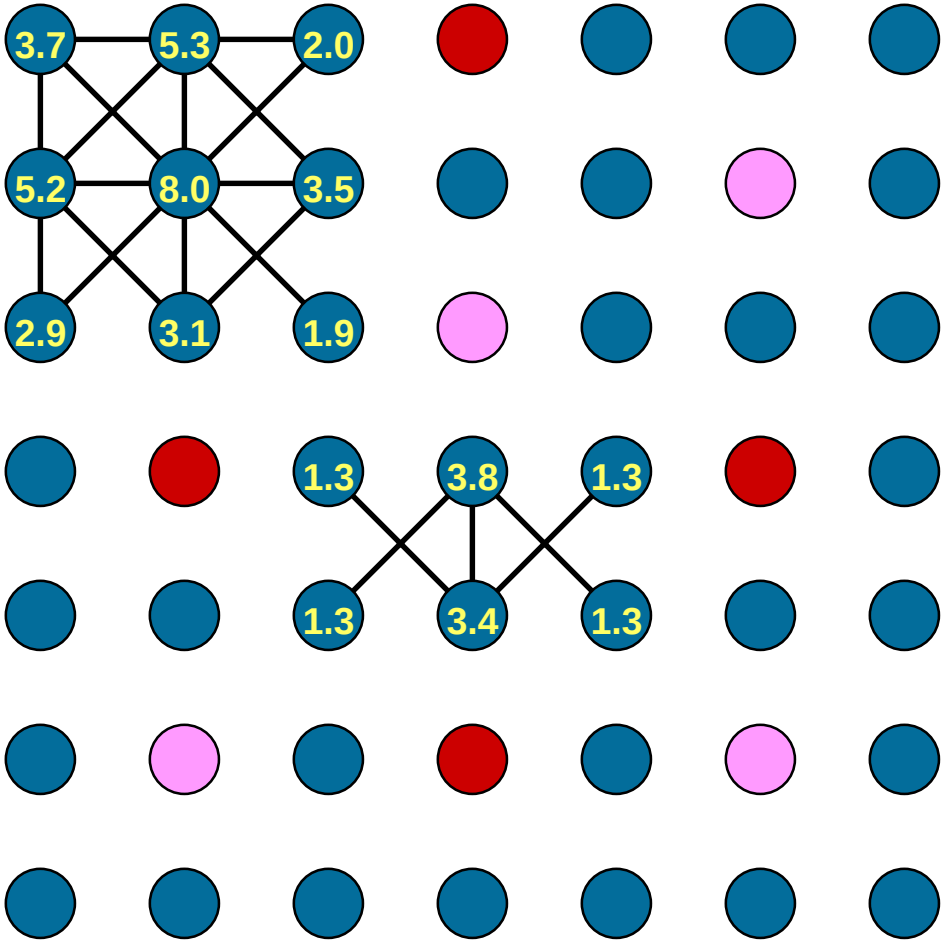
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

Cleary-LJP: select 2



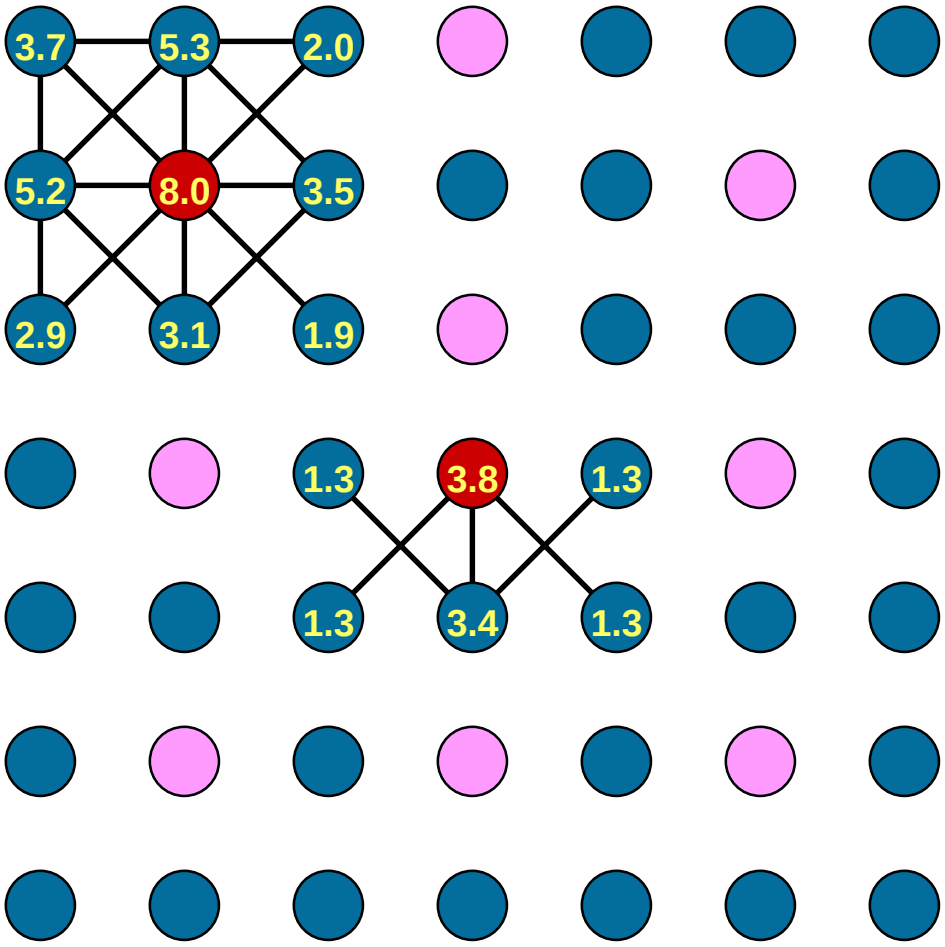
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

Cleary-LJP: remove and update 2



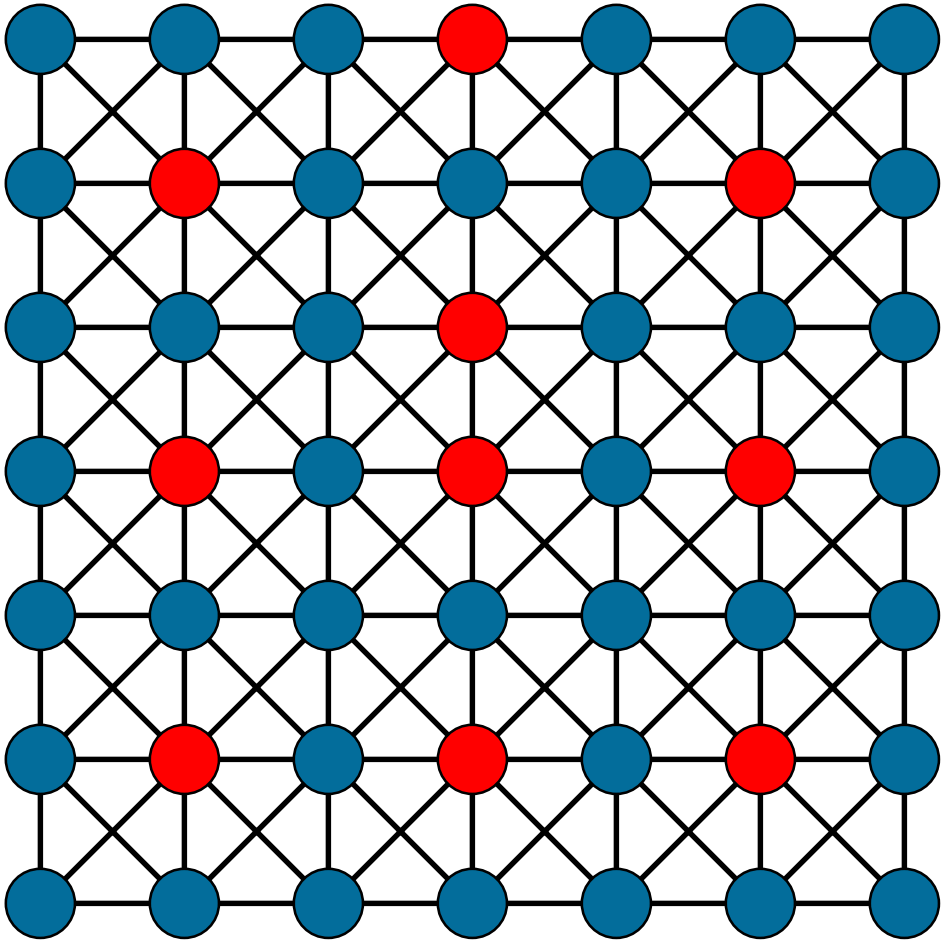
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

Cleary-LJP: select 3



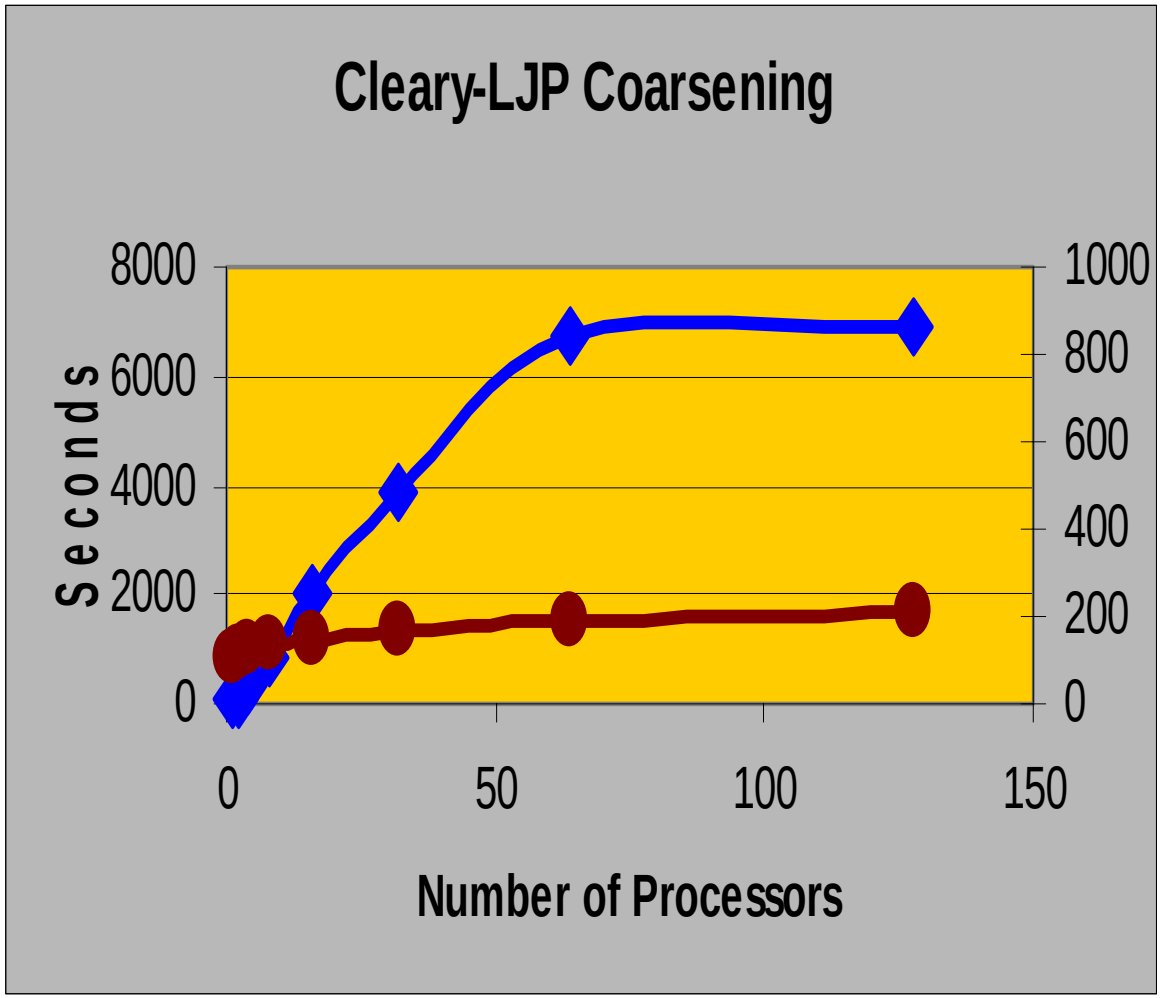
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

Cleary-LJP: final grid



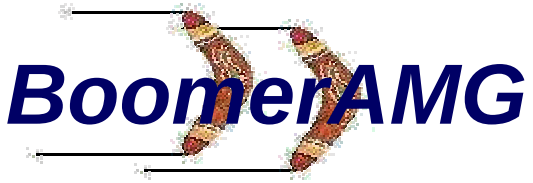
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

Cleary-LJP results



- 3D 7pt Laplacian, 125,000 points/proc.
- Setup phase shows poor scalability
- Solve phase shows relatively good scalability
- Operator complexity (ratio: total matrix nonzeros, all grids, to nonzeros, fine grid) quite high ~20-25

Cleary-LJP results



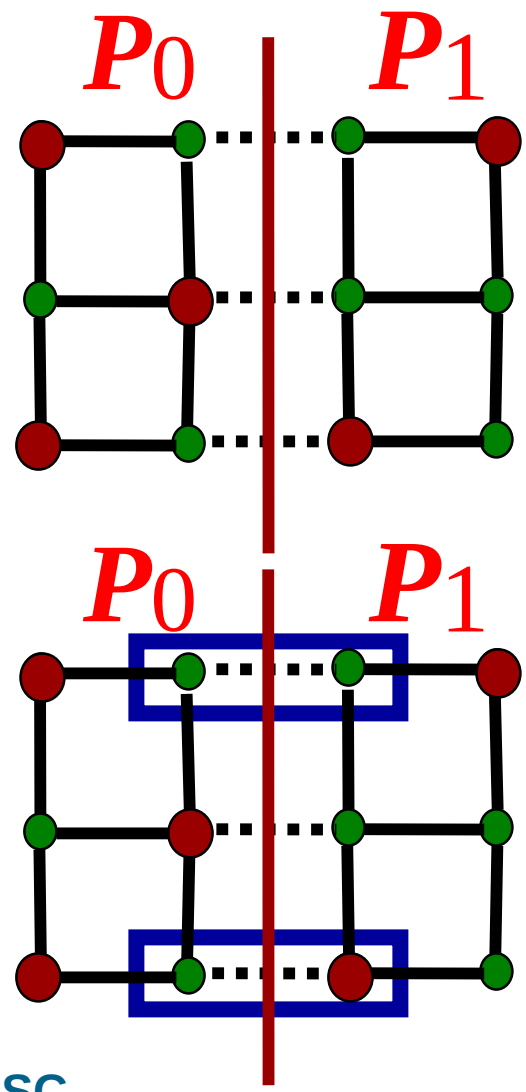
7 pt 3D Laplacian			27 pt 3D Laplacian		9 pt 2D Laplacian	
Procs.	Setup	Op. Cplx	Setup	Op. Cplx	Setup	Op. Cplx
1	19	18.37	16	1.83	6	1.94
2	48	20.01	44	1.89	7	1.94
4	142	21.89	138	2.01	8	1.95
8	354	23.45	318	2.16	11	1.95
16	681	24.41	595	2.19	14	1.96
32	1405	25.39	1009	2.31	14	1.96
64	2992	26.39	1975	2.93	18	1.96
128	3030	27.06	2010	2.43	32	1.96
256					49	1.96
Procs	Solve	C.F.	Solve	C.F.	Solve	C.F.
1	49	0.176	22	0.116	22	0.312
2	55	0.199	24	0.147	23	0.338
4	61	0.217	25	0.167	24	0.391
8	67	0.267	28	0.271	24	0.382
16	75	0.334	30	0.277	24	0.438
32	82	0.381	33	0.307	24	0.436
64	94	0.456	36	DIV	25	0.472
128	97	0.486	41	DIV	29	0.473
256					31	0.551

Parallel Ruge Coarsening

- Another approach to coarsening in parallel: perform the standard Ruge algorithm on each processor. Various treatments possible at processor boundaries.
- Yields processor dependent coarsenings, and will not produce the same reults for different numbers of processors.
- The “measure” of each point should include the number of off-processor connections, even when coarsening within processor.

Parallel Ruge coarsening: boundary treatment: I

BoomerAMG

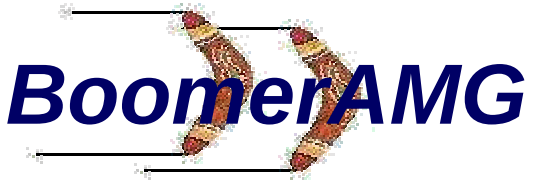


Perform first and second passes on each processor

Method 1: Do nothing. Accept the coarsening provided by the independent processors.

Problem: Leaves $F \Leftrightarrow F$ dependencies without mutual C-points

Parallel Ruge coarsening results



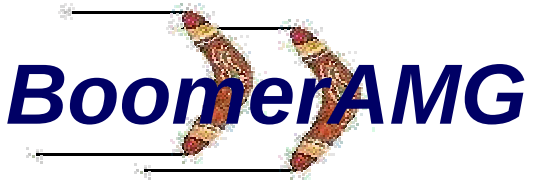
7 pt 3D Laplacian		
Procs.	Setup	Op. Cplx
1	14	4.91
2	26	5.25
4	63	5.71
8	153	6.23
16	328	6.75
32	561	6.98
64	999	7.34
128		
Procs	Solve	C.F.
1	36	0.065
2	40	0.081
4	43	0.111
8	48	0.210
16	389	0.246
32	3433	0.605
64	3352	0.384
128		

Ruge coarsening is much faster and yields much better complexities than Cleary-LJP on the 7-pt Laplacian

Note that the solve times jump by orders of magnitude as problem grows. Parallel Ruge leads to large "coarsest" grids with direct solve.

Solution: hybrid coarsening

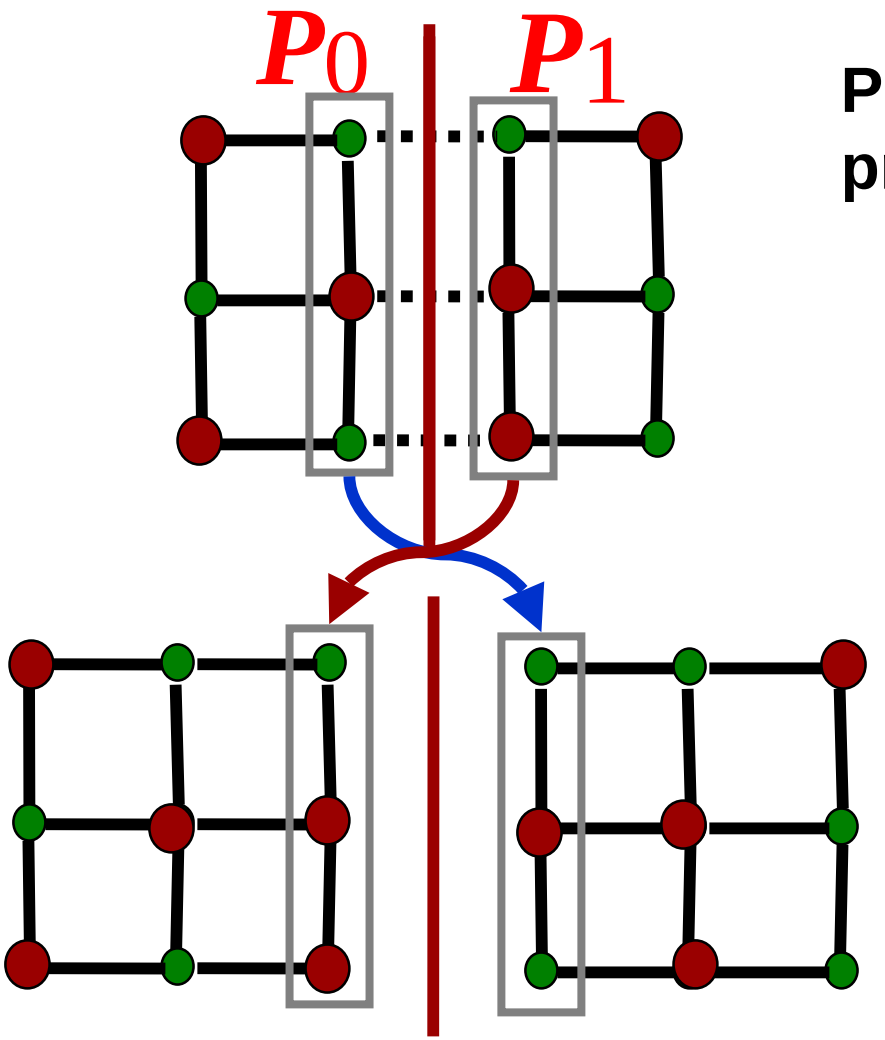
Parallel Ruge-JLP Hybrid: no boundary treatment



7 pt 3D Laplacian			27 pt 3D Laplacia		9 pt 2D Laplacia	
Procs	Setup	Op. Cplx	Setup	Op. Cplx	Setup	Op. Cplx
1	6	4.39			3	
2	11	5.46	21	2.54	4	1.35
4	25	6.79	60	2.76	4	1.35
8	65	8.51	153	3.13	4	1.35
16	148	8.89	271	3.21	5	1.35
32	292	8.92	506	3.45	5	1.35
64	456	8.52	1089	3.81	6	1.35
128	488	8.38	1217	3.93	8	1.35
256					12	1.49
Procs	Solve	C.F.	Solve	C.F.	Solve	C.F.
1					17	0.203
2	20	0.091	29	0.123	18	0.599
4	24	0.138	32	0.145	18	0.612
8	64	0.279	38	0.192	19	0.61
16	147	0.385	41	DIV	19	0.606
32	293	DIV	54	DIV	19	0.607
64	456	DIV	62	DIV	20	0.627
128	488	DIV	69	DIV	20	0.644
256					21	0.472

Parallel Ruge coarsening: boundary treatment (Ruge2b)

BoomerAMG



Perform first pass on each processor

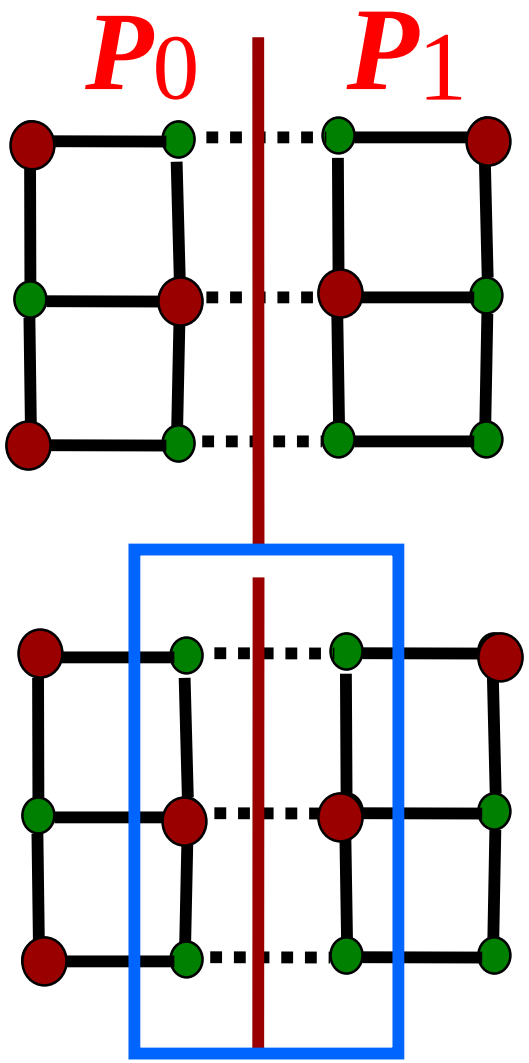
Perform second pass locally on each processor, augmented by boundary points from neighbor

Choices must be made about how to resolve conflicting decisions among processors

Parallel Ruge coarsening: boundary treatment (Ruge2b)

7 pt 3D Laplacian			27 pt 3D Laplacian		9 pt 2D Laplacian	
Procs.	Setup	Op. Cplx	Setup	Op. Cplx	Setup	Op. Cplx
1					3	1.33
2	15	5.31	40	2.72	4	1.33
4	45	6.53	164	3.19	5	1.34
8	121	8.06	460	4.37	6	1.36
16	254	8.09	732	4.74	7	1.37
32	527	8.62	1232	5.51	10	1.38
64					15	1.38
128	1058	9.07			23	1.38
256					39	1.51
Procs	Solve	C.F.	Solve	C.F.	Solve	C.F.
1					17	0.121
2	16	0.122	31	0.111	18	0.120
4	24	0.211	42	0.158	19	0.268
8	29	0.269	54	0.216	19	0.292
16	31	DIV	61	0.257	20	0.347
32	37	0.399	88	DIV	20	0.404
64					20	0.404
128	48	DIV			25	0.388
256					26	0.485

Parallel Ruge coarsening: boundary treatment (Ruge3)



Perform first and second pass on each processor

Perform a third pass, (a second "second pass"), only on those points adjacent to processor boundaries

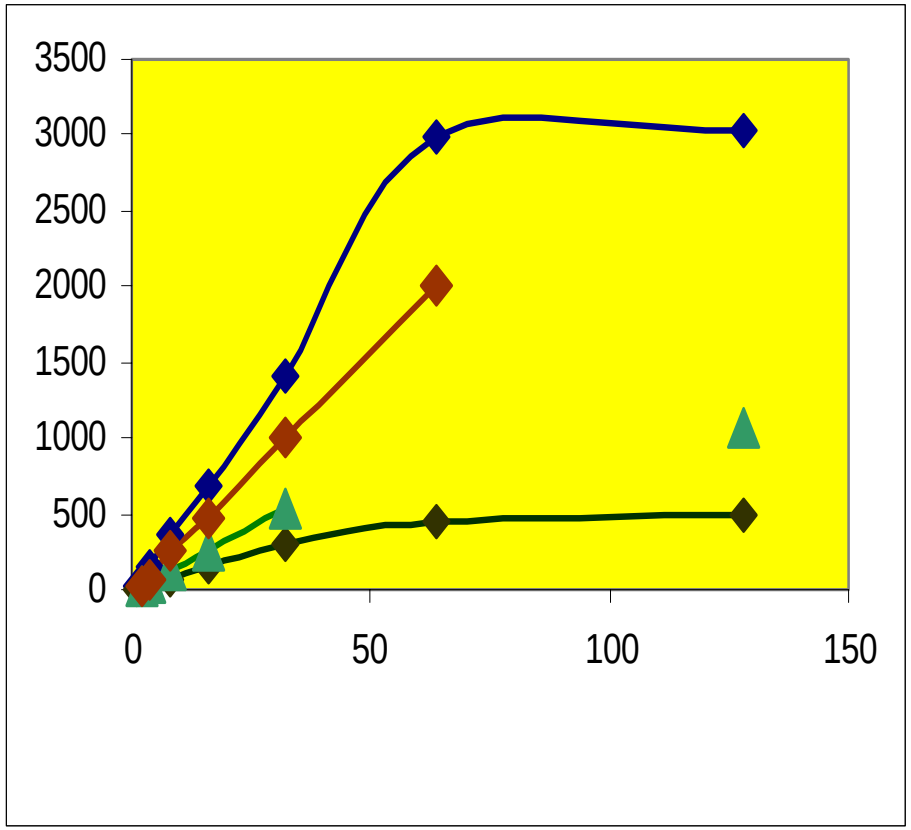
Choices must be made about how to resolve conflicting decisions among processors

Parallel Ruge coarsening: boundary treatment (Ruge3)

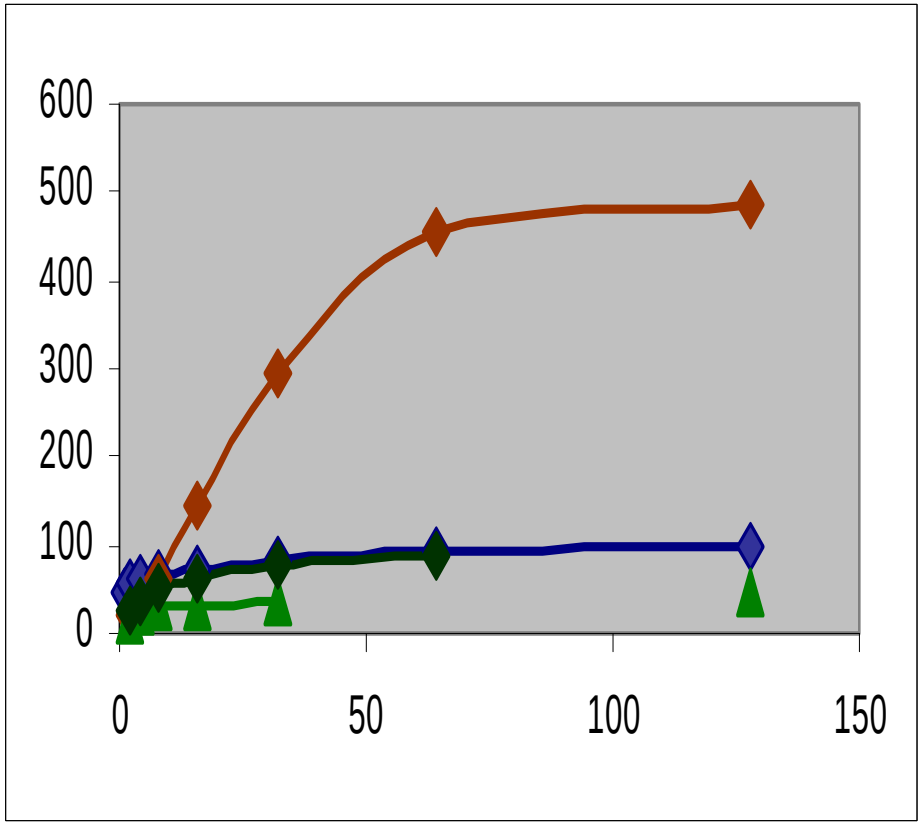
7 pt 3D Laplacian			27 pt 3D Laplacian		9 pt 2D Laplacian	
Procs.	Setup	Op. Cplx	Setup	Op. Cplx	Setup	Op. Cplx
1					3	1.33
2	17	7.62	36	2.35	3	1.33
4	70	12.07	128	3.48	4	1.35
8	249	16.76	365	4.88	5	1.36
16	479	16.69	684	5.82	6	1.37
32	1008	16.13	1423	7.31	8	1.38
64	2008	15.25			17	1.38
128					21	1.39
256					40	1.51
Procs	Solve	C.F.	Solve	C.F.	Solve	C.F.
1					17	0.121
2	26	0.128	27	0.119	18	0.121
4	38	0.158	44	0.163	19	0.141
8	53	0.217	64	0.215	19	0.225
16	62	0.233	78	0.238	20	0.336
32	76	0.348	112	DIV	20	0.312
64	90	DIV			22	0.318
128					23	0.385
256					26	0.474

7 pt 3D Laplacian

BoomerAMG



Setup Times

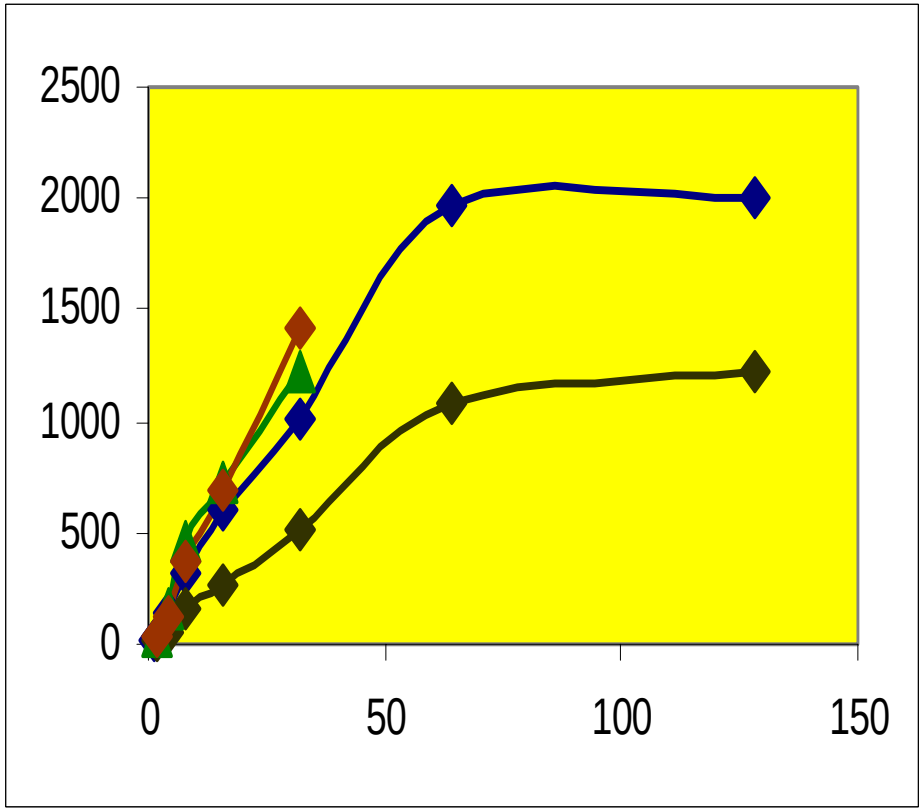


Solve Times

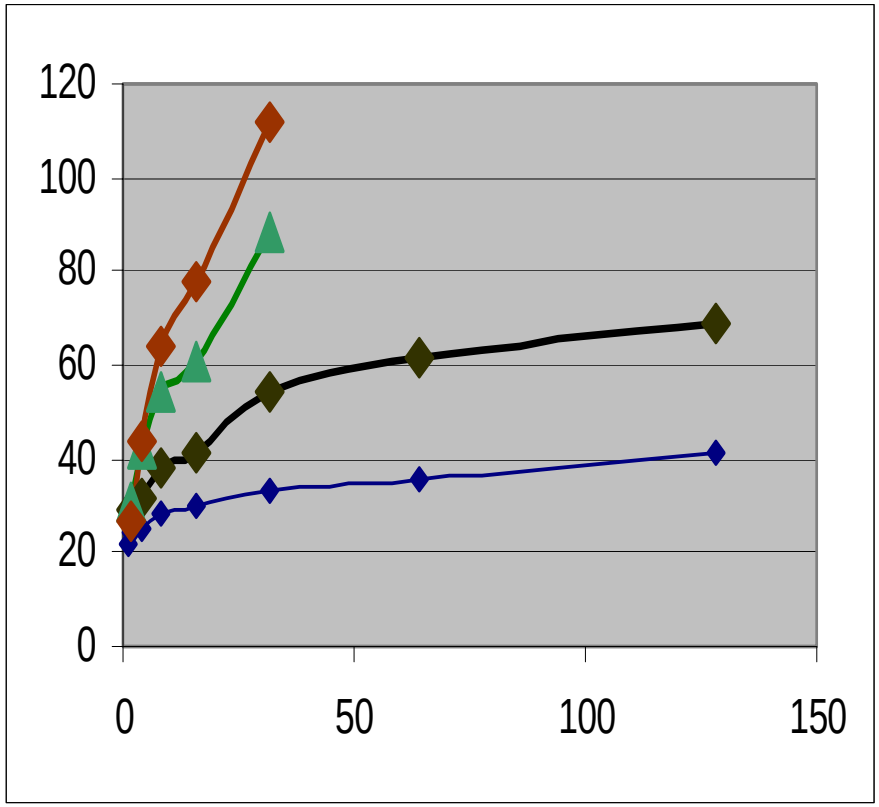
C-LJP Ruge Ruge (2b) Ruge (3)

27 pt 3D Laplacian

BoomerAMG



Setup Times

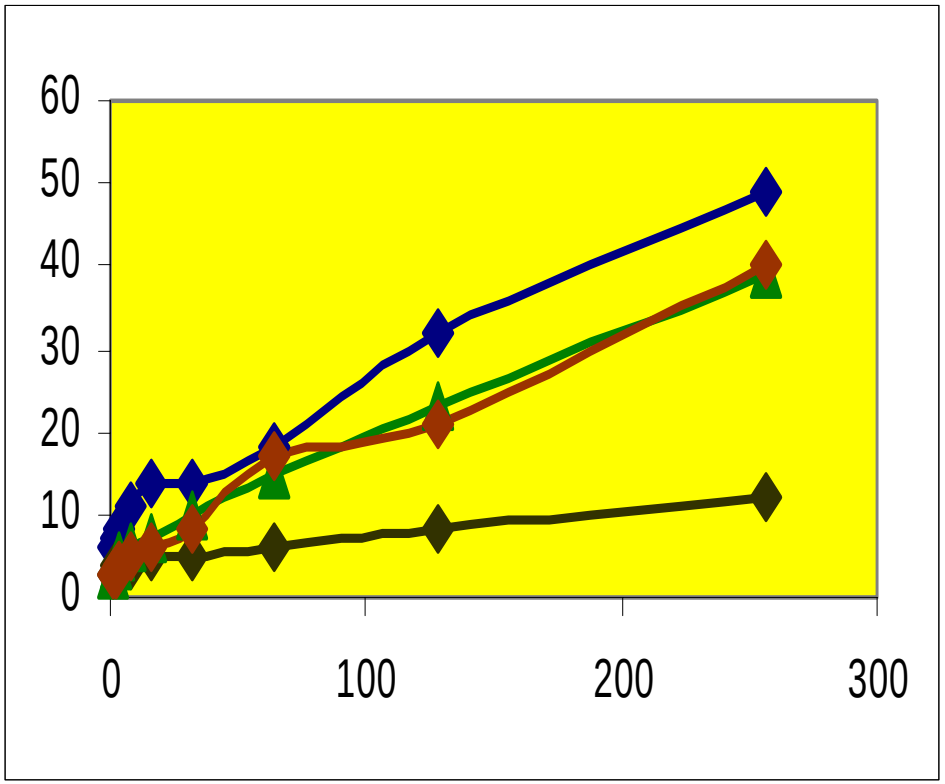


Solve Times

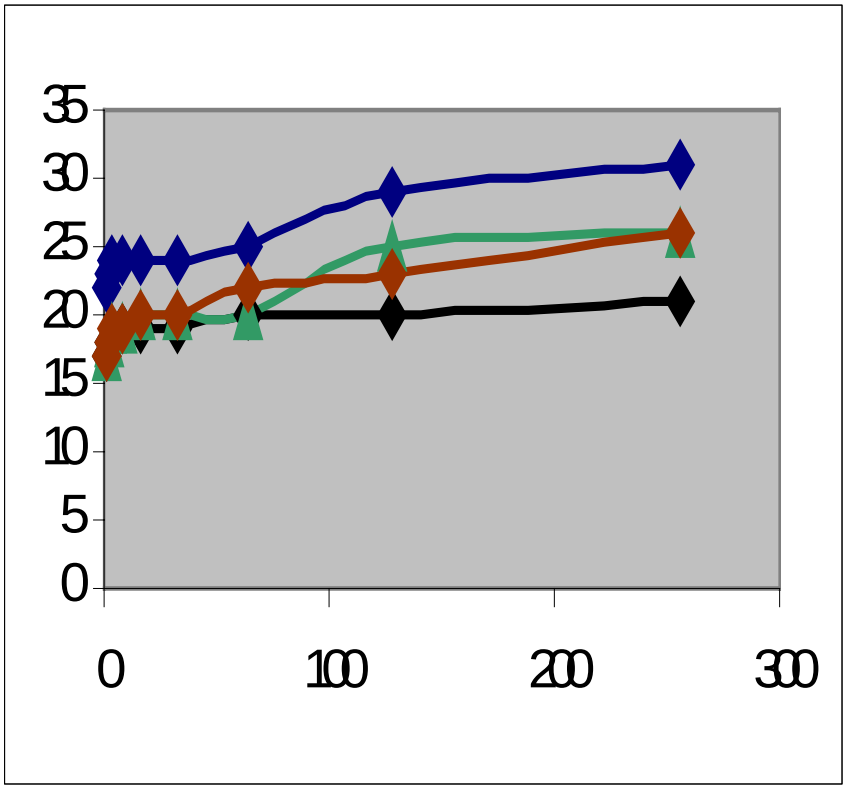
C-LJP Ruge Ruge (2b) Ruge (3)

9 pt 2D Laplacian

BoomerAMG



Setup Times



Solve Times

C-LJP Ruge Ruge (2b) Ruge (3)

Conclusions

- **Testing is still needed to implement the algorithms efficiently; to determine better ways of treating processor boundaries, operator complexities, and growing convergence factors.**
- **Future computer science plans include load balancing and efficient cache useage.**
- **Future algorithmic development centers on implementing "system" solvers and determining MG components using the finite-element stiffness matrices**
- This work was performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under contract number: W-7405-Eng-48. Release number UCRL MI 133583