Institute for Advanced Computing
Applications and Technologies

Finding the Happy Medium: Tradeoffs in Communication, Algorithms, Architectures and Programming Models

William Gropp

www.cs.illinois.edu/~wgropp

NCSA

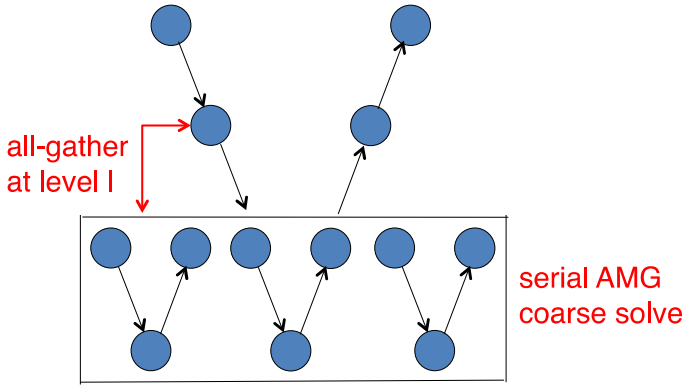
The Institute combines research in a host of disciplines at the University of Illinois with the advanced technology capabilities of the National Center for Supercomputing Applications.

Message

- Algorithms and software must acknowledge realities of architecture
- Message:
  - ***Appropriate*** performance models can guide development
  - Avoid ***unnecessary*** synchronization
    - Often encouraged by the programming model
  - ***Don't*** (only) ***optimize components individually***
    - Interactions between parts matter

Using Redundant Solvers

- AMG requires a solve on the coarse grid



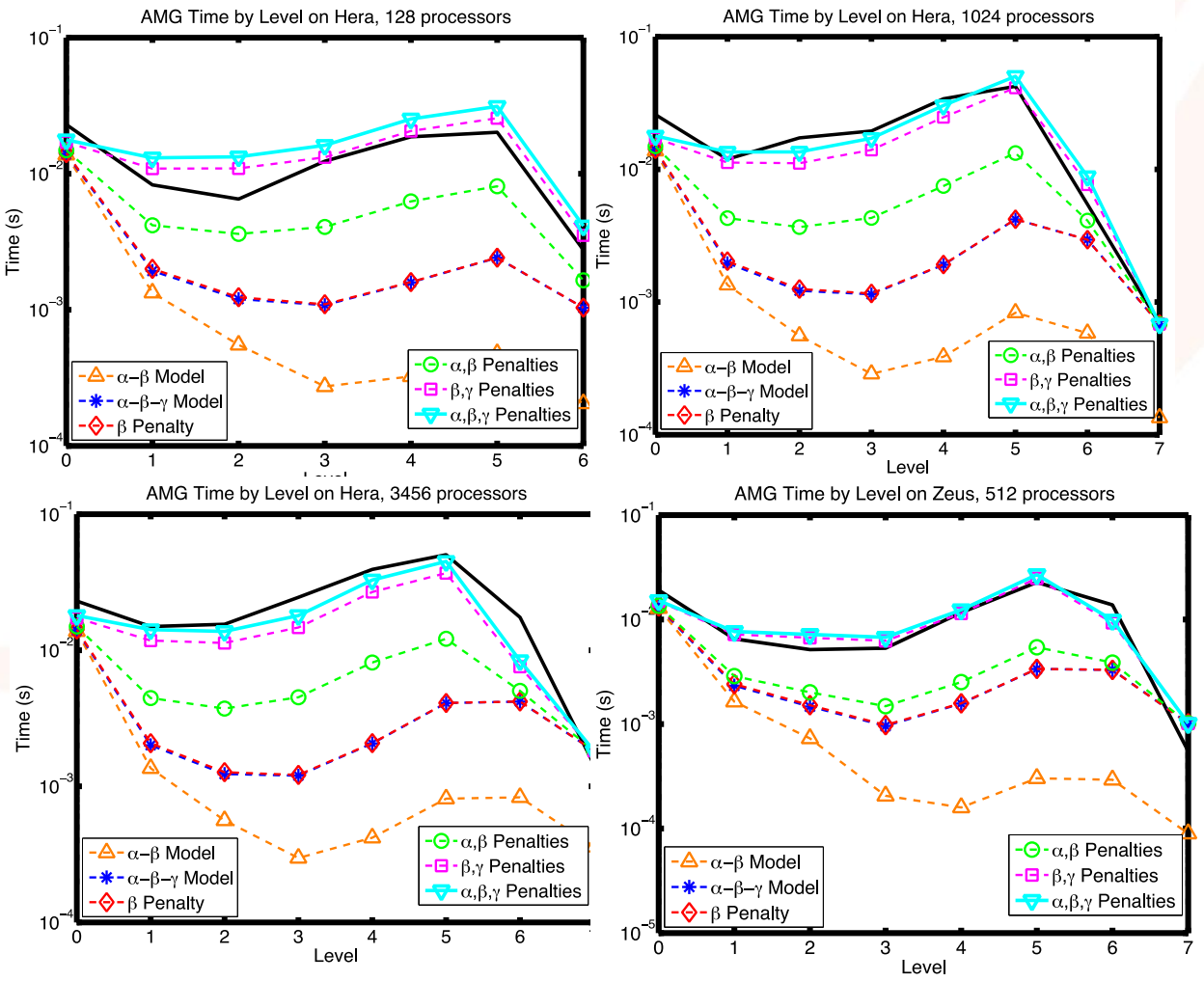
- Rather than either solve in parallel (too little work for the communication) or solve in serial and distribute solution, solve redundantly (either in smaller parallel groups or serial, as in this illustration)

Redundant Solution

- Replace communication at levels ≥lred with Allgather
- Every process now has complete information; no further communication needed
- Performance analysis (based on Gropp & Keyes 1989) can guide selection of lred

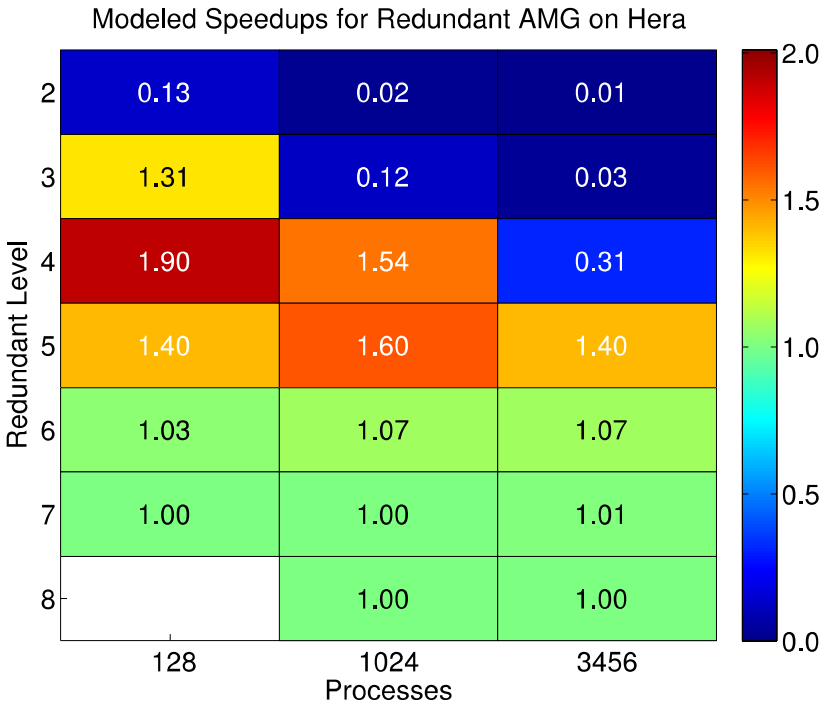
AMG Performance Model

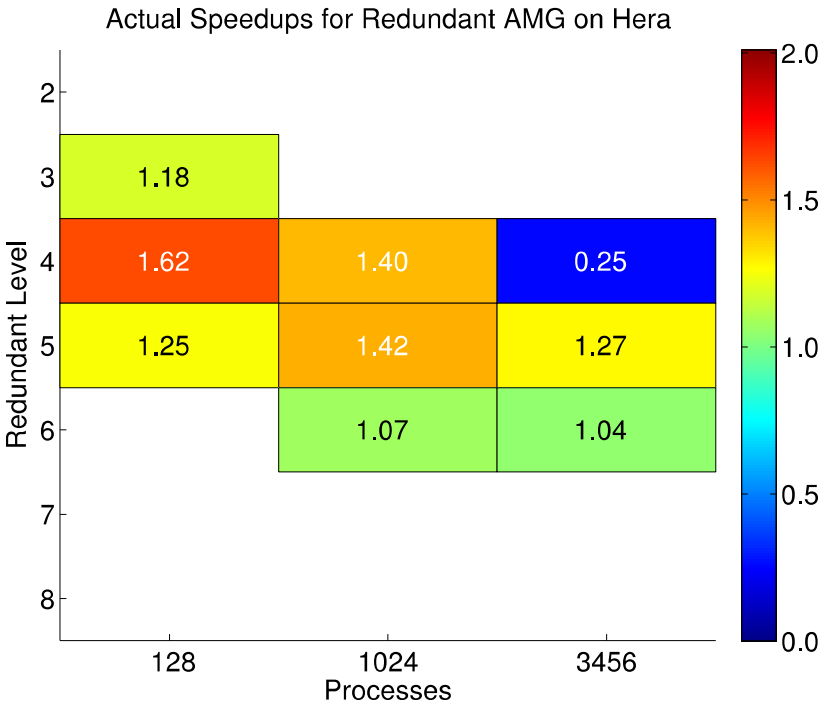
- We can establish upper and lower bounds and compare performance
- Includes contention, bandwidth, multicore penalties
- 82% accuracy on Hera, 98% on Zeus
- Gahvari, Baker, Schulz, Yang, Jordan, Gropp (ICS'11)



Redundant Solves

- Applied to Hera at LLNL, provides significant speedup





- Thanks to Hormozd Gahvari

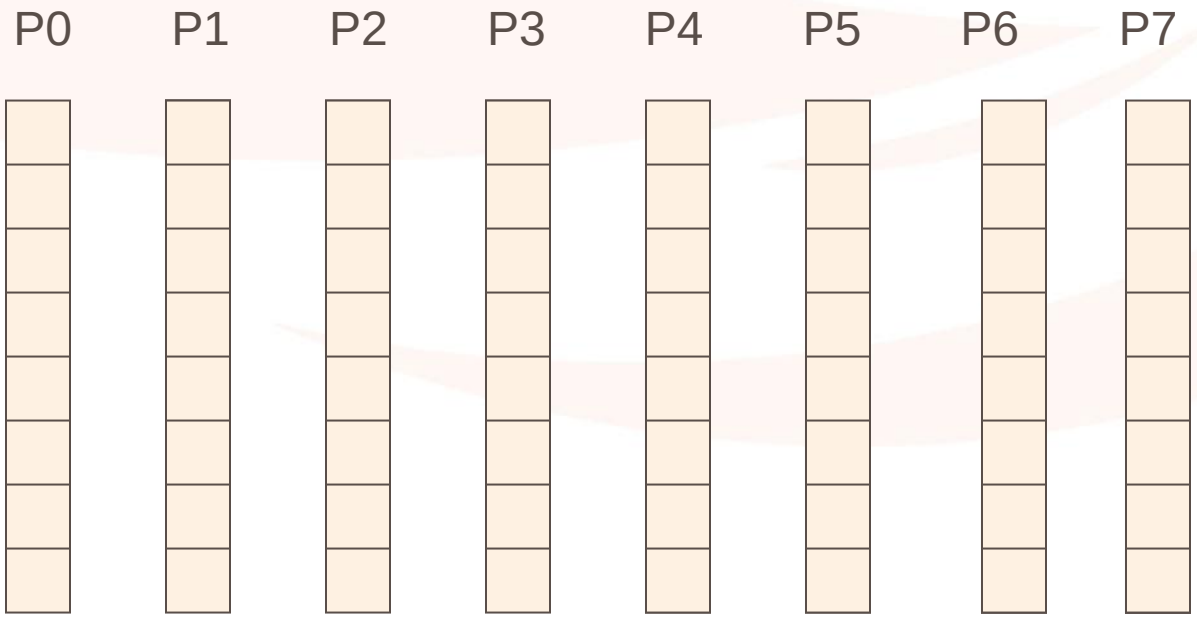
Thinking about Broadcasts

- MPI_Bcast(buf, 100000, MPI_DOUBLE, …);
- Use a tree-based distribution:

- Use a *pipeline*: send the message in b byte pieces. This allows each subtree to begin communication after b bytes sent
- Improves total performance:
  - Root process takes same time (asymptotically)
  - Other processes wait less
    - Time to reach leaf is *b log p + (n-b)*, rather than *n log p*
- Special hardware and other algorithms can be used …

Make Full Use of the Network

- Implement MPI_Bcast(buf,n,…) as
  MPI_Scatter(buf, n/p,…, buf+rank*n/p,…)
  MPI_Allgather(buf+rank*n/p, n/p,…,buf,…)



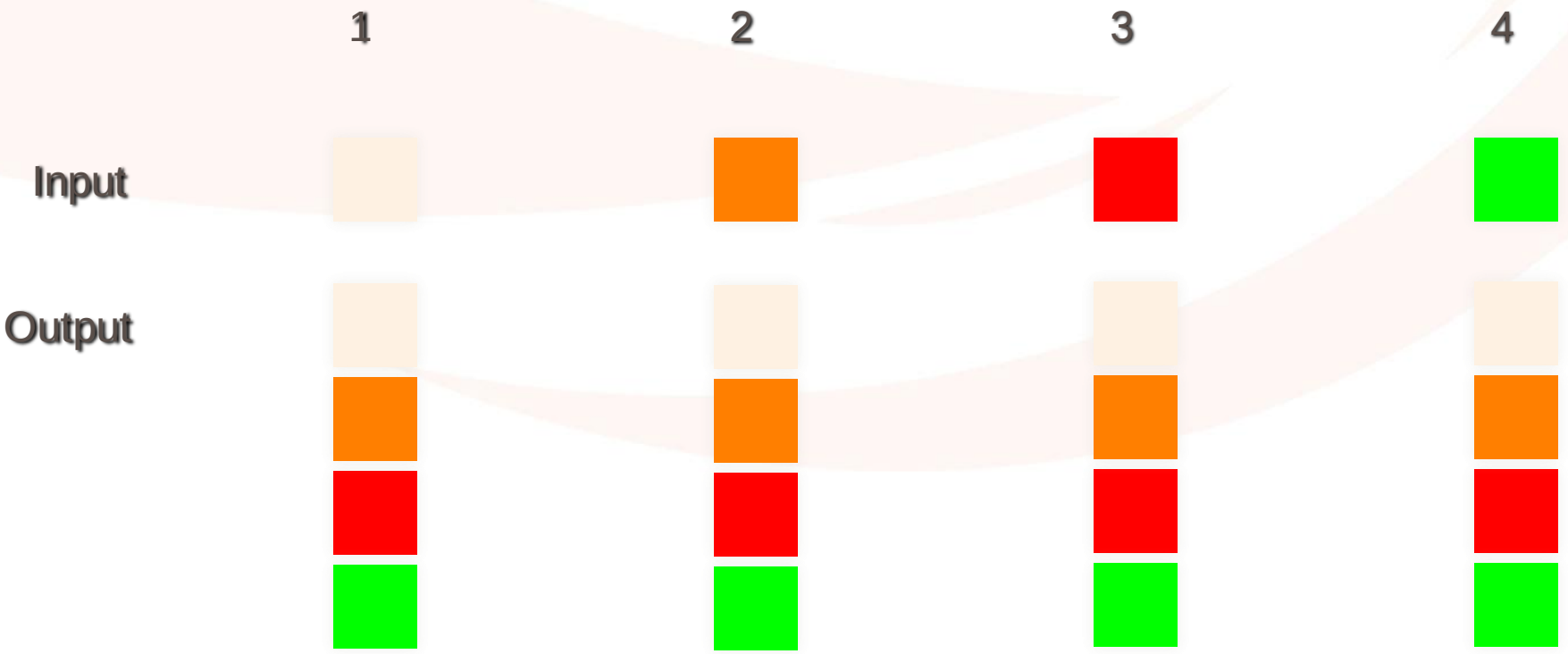
Optimal Algorithm Costs

- Optimal cost is O(n) (O(p) terms don't involve n) since scatter moves n data, and allgather also moves only n per process; these can use pipelining to move data as well
  - Scatter by recursive bisection uses log p steps to move n(p-1)/p data
  - Scatter by direct send uses p-1 steps to move n(p-1)/p data
  - Recursive doubling allgather uses log p steps to move
    - N/p + 2n/p + 4n/p + … (p/2)/p = n(p-1)/p
  - Bucket brigade allgather moves
    - N/p (p-1) times or (p-1)n/p
- See, e.g., van de Geijn for more details

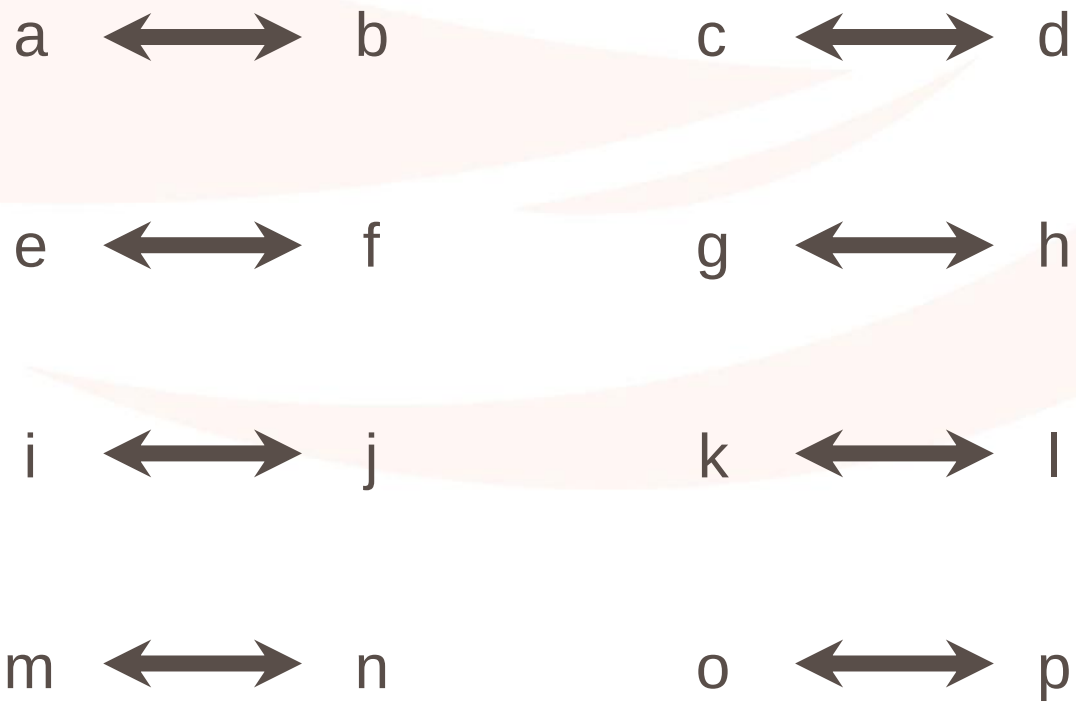
Is it communication avoiding or minimum solution time?

- Example: non minimum collective algorithms
- Work of Paul Sack; see “Faster topology-aware collective algorithms through non-minimal communication”, PPoPP 2012
- Lesson: ***minimum communication need not be optimal***

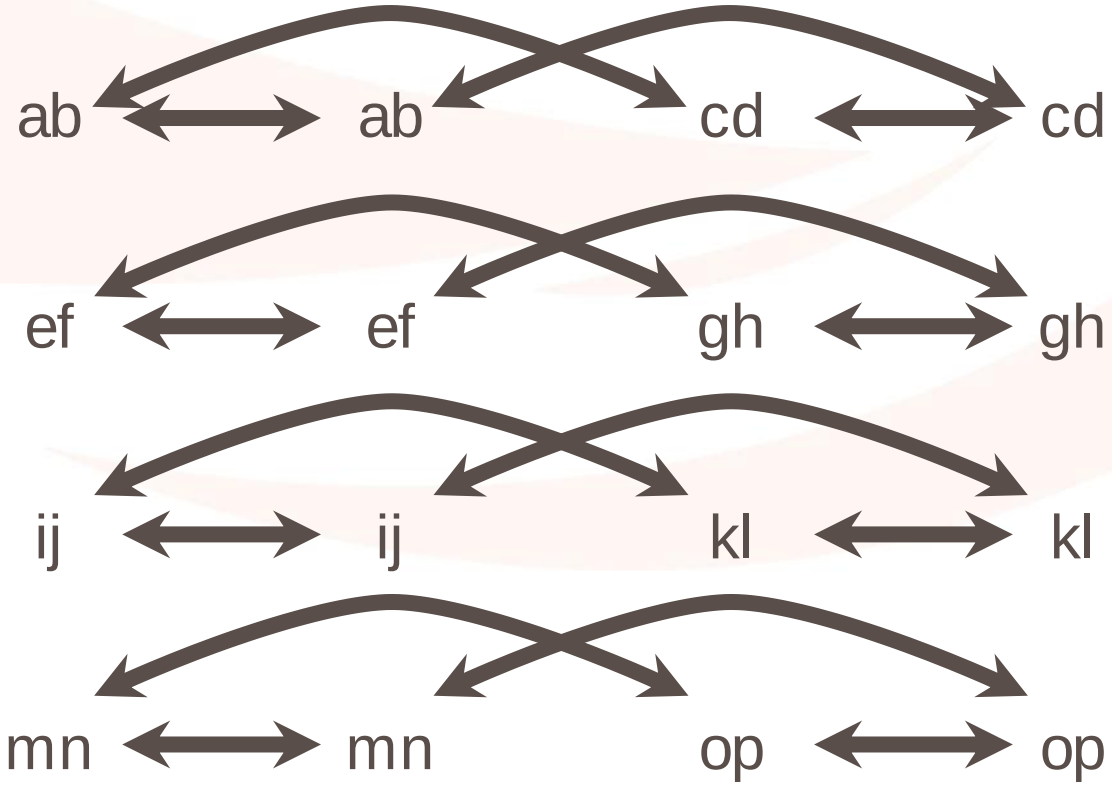
Allgather



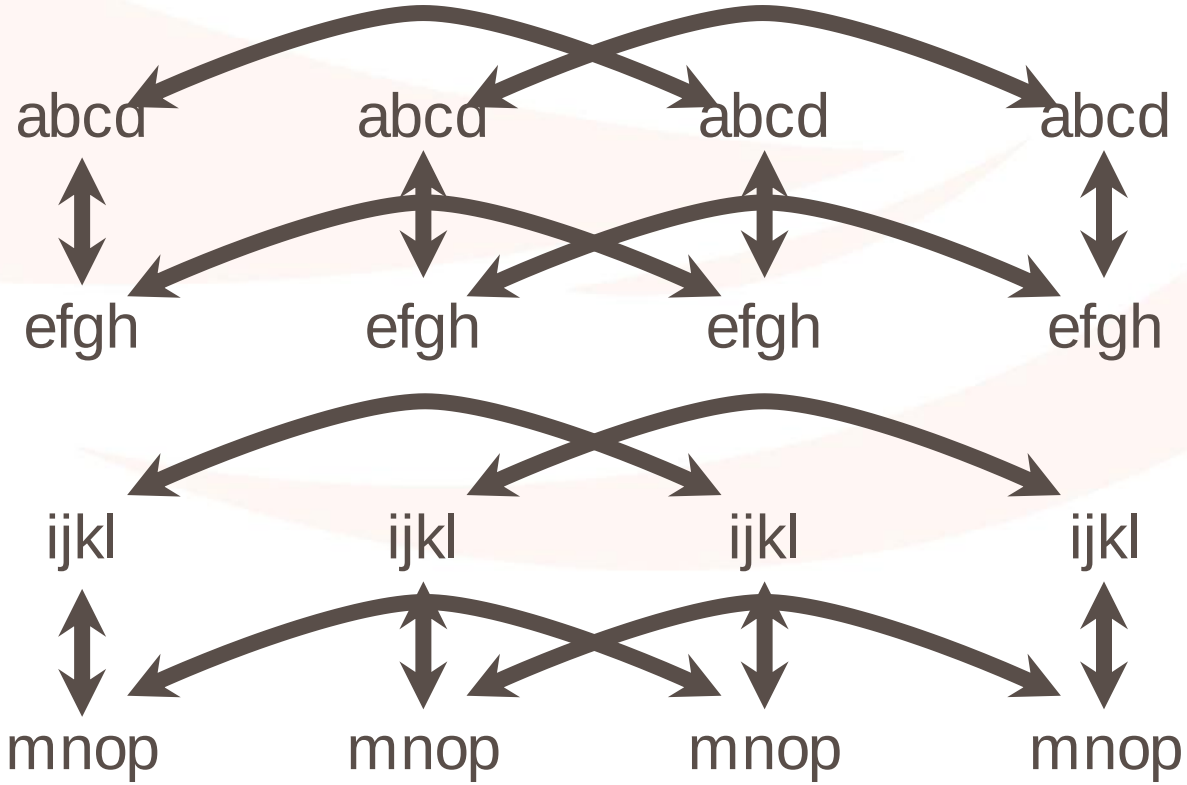
Allgather: recursive doubling



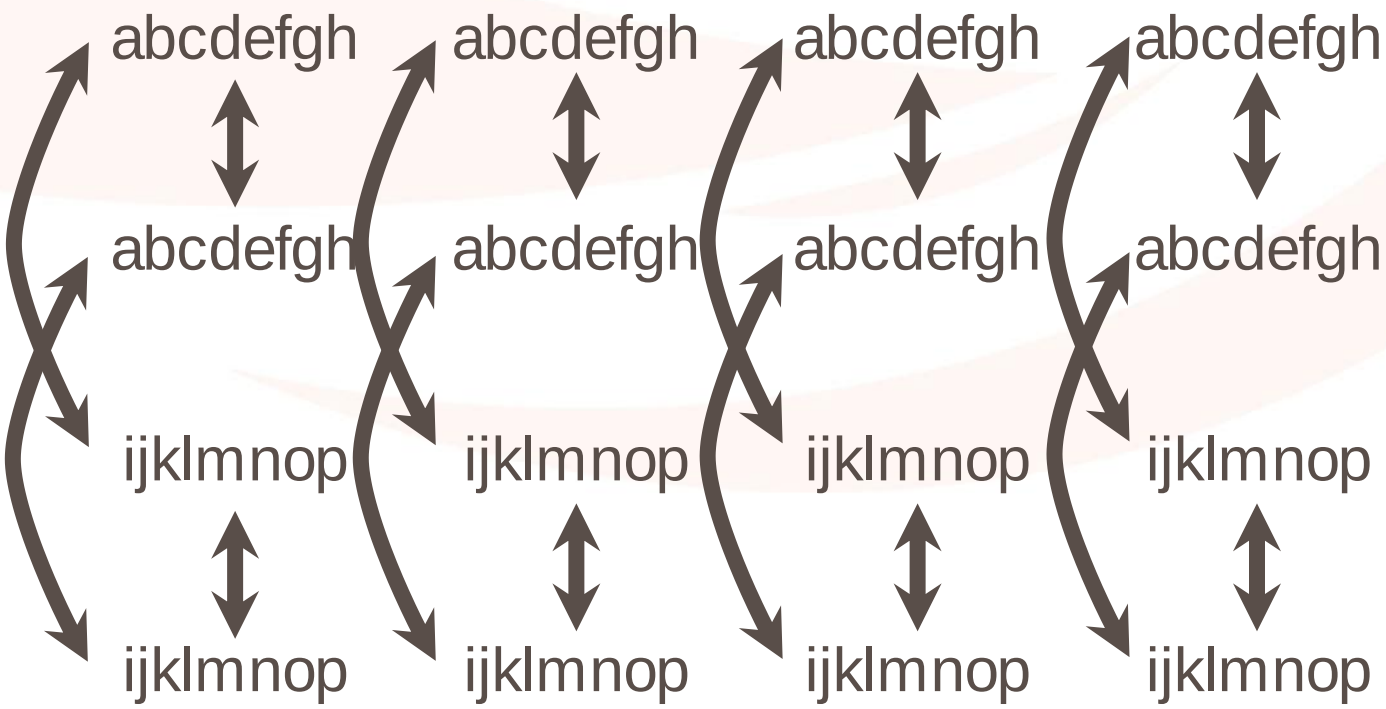
Allgather: recursive doubling



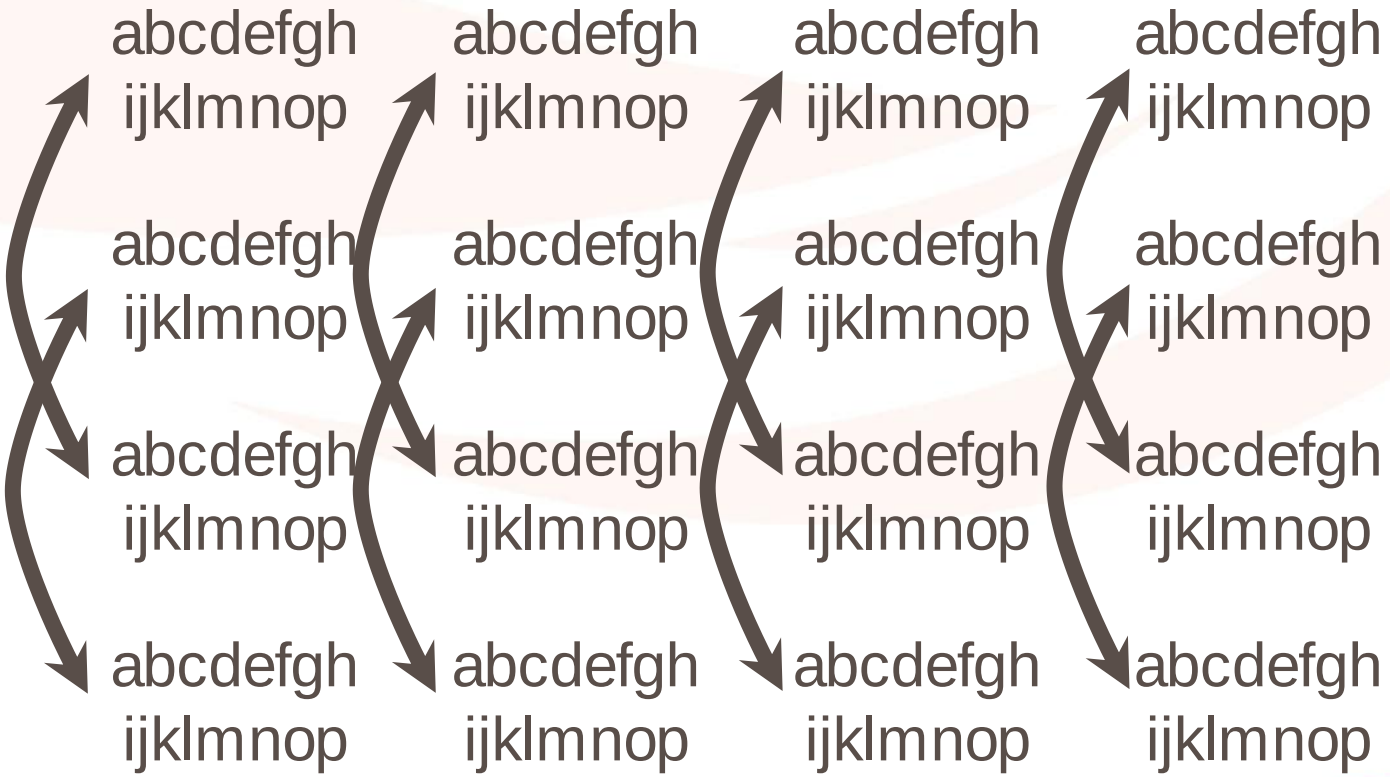
Allgather: recursive doubling



Allgather: recursive doubling



Allgather: recursive doubling



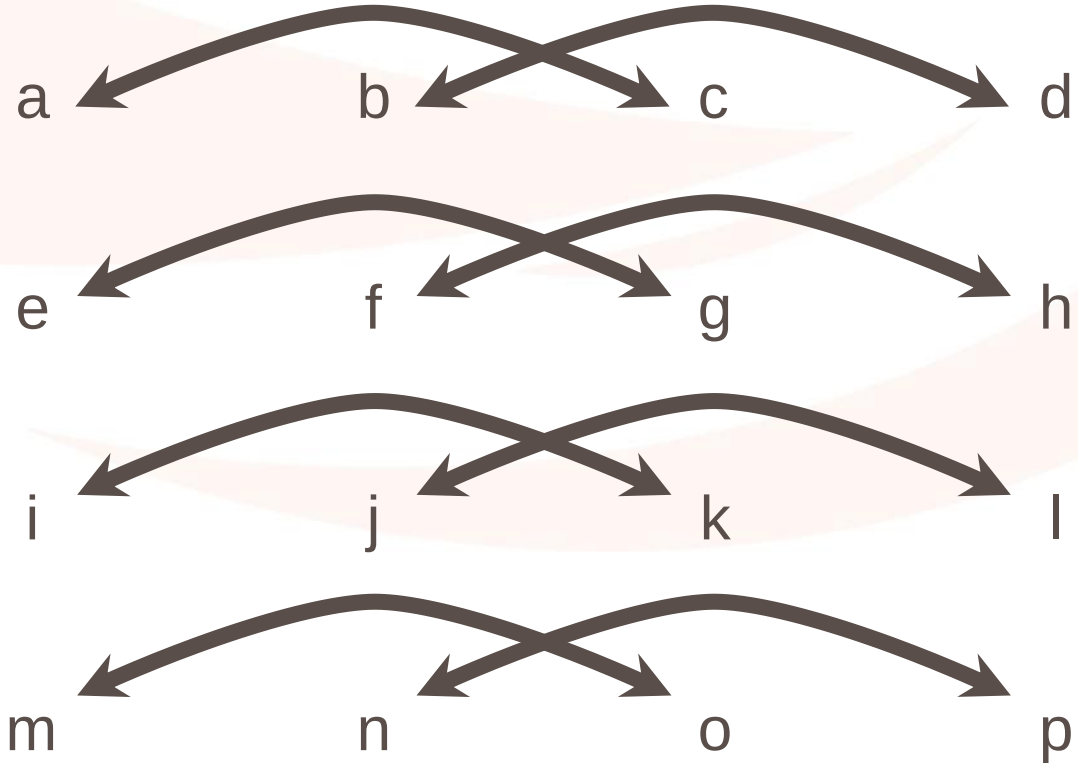
$T=(lg\ P)\ \alpha + n(P-1)\beta$

Problem: Recursive-doubling

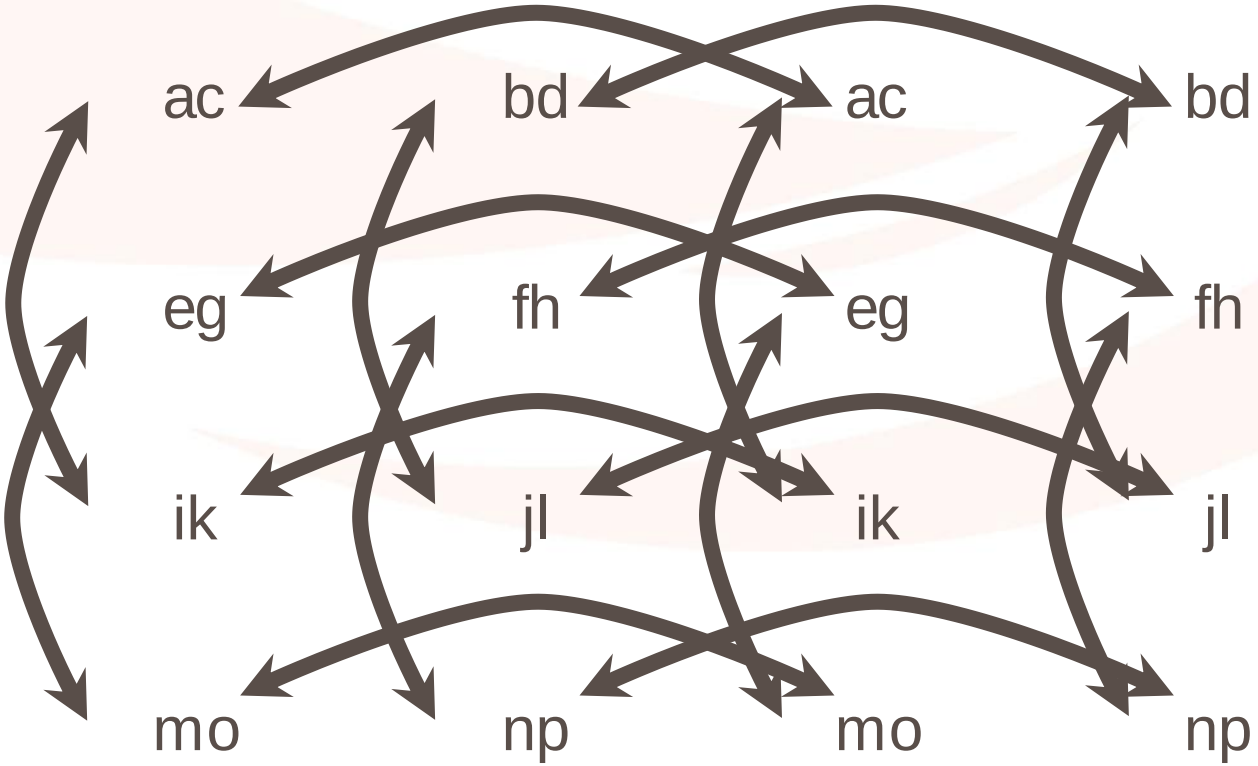
- No congestion model:
  - $T=(\lg P)\alpha + n(P-1)\beta$
- Congestion on torus:
  - $T\approx(\lg P)\alpha + (5/24)nP^{4/3}\beta$
- Congestion on Clos network:
  - $T\approx(\lg P)\alpha + (nP/\mu)\beta$

- Solution approach: move smallest amounts of data the longest distance

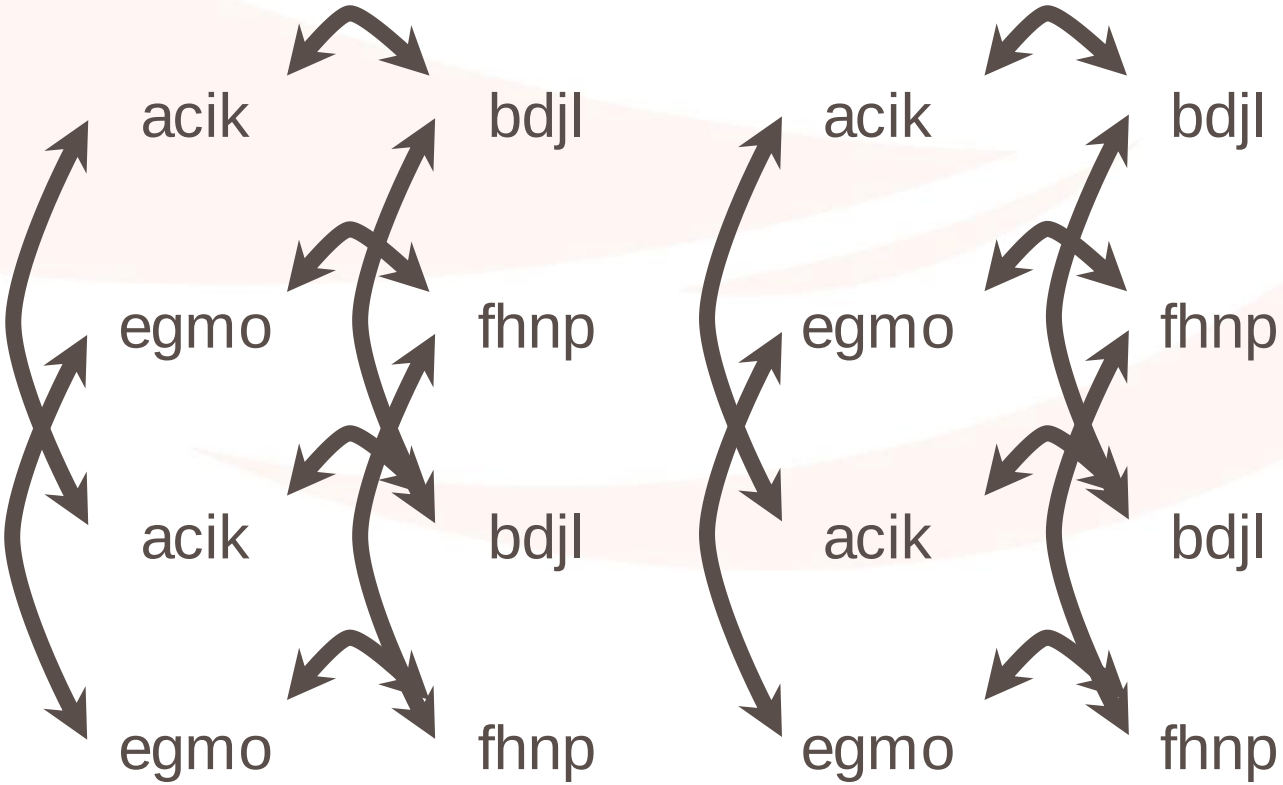
Allgather: recursive halving



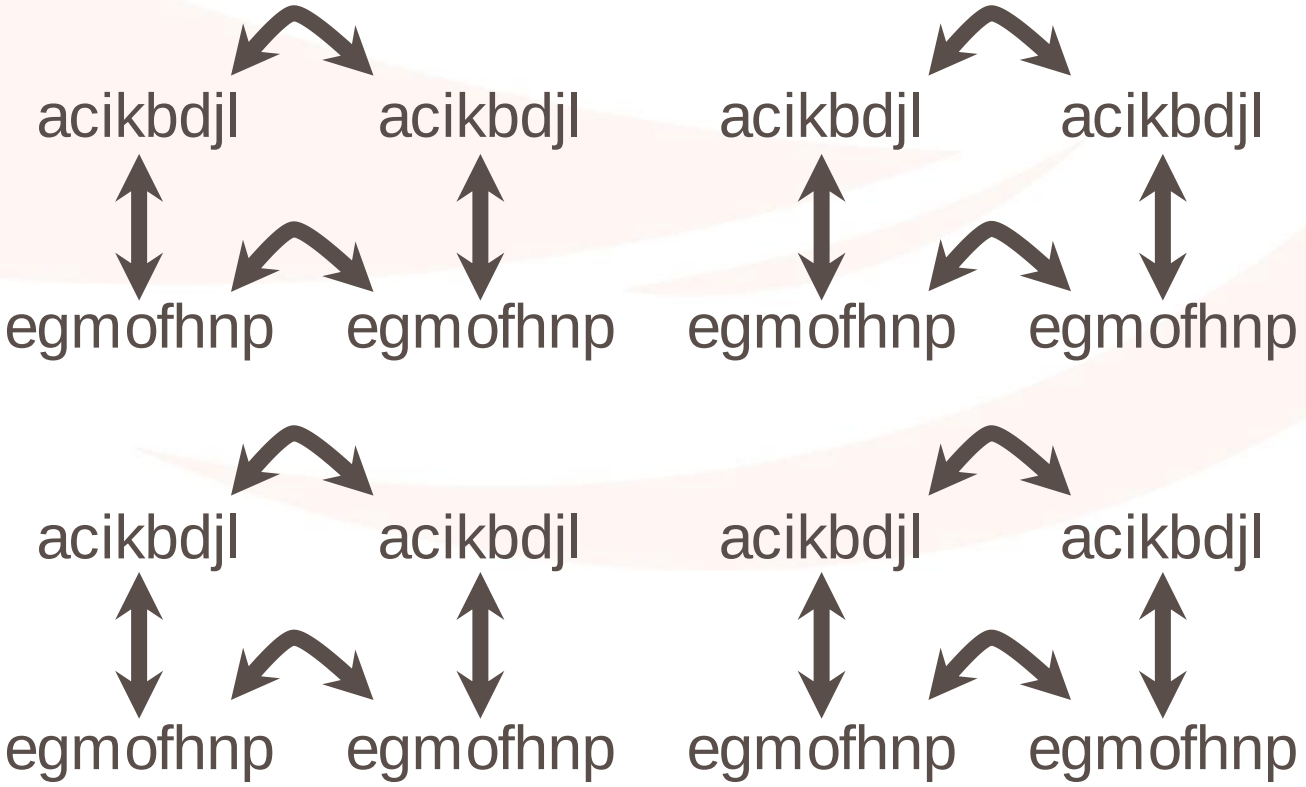
Allgather: recursive halving



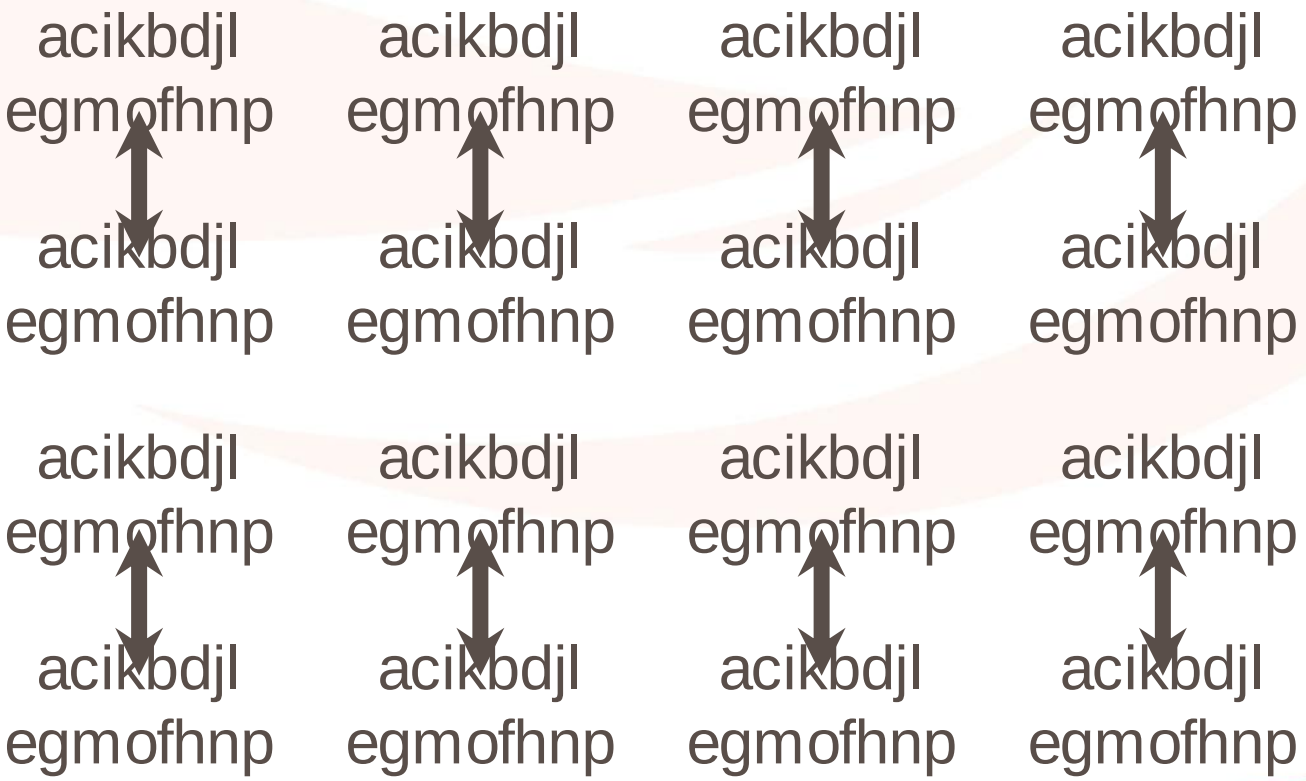
Allgather: recursive halving



Allgather: recursive halving



Allgather: recursive halving

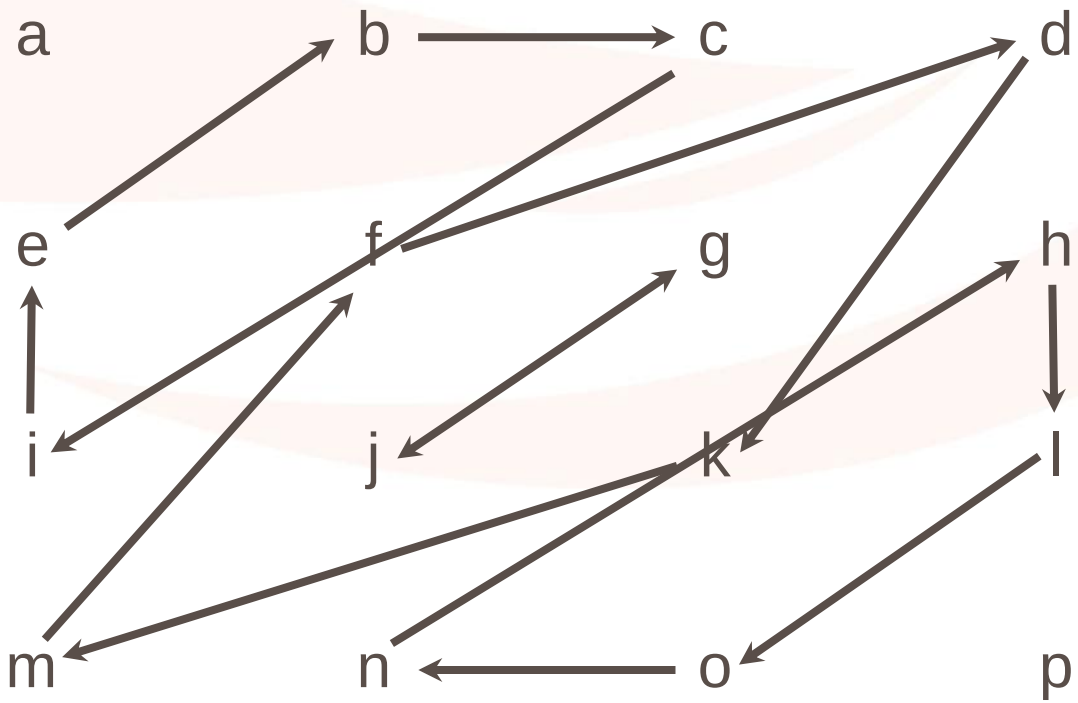


$T=(lg\ P)\alpha + (7/6)nP\beta$

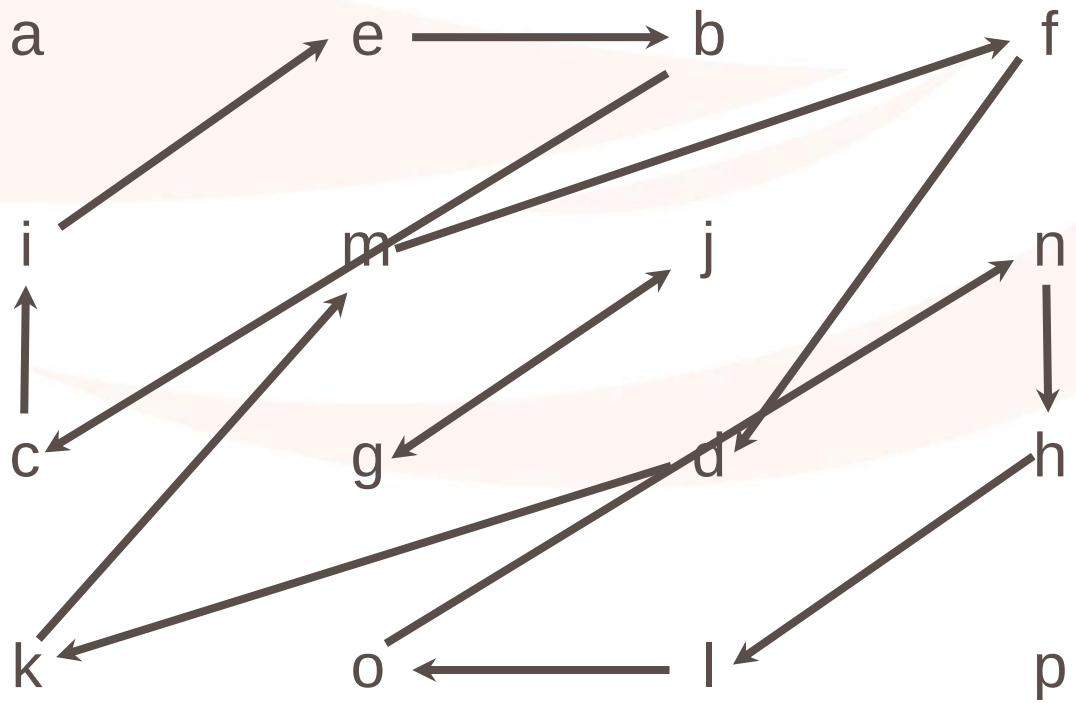
New problem: data misordered

- Solution: shuffle input data
  - Could shuffle at end (redundant work; all processes shuffle)
  - Could use non-contiguous data moves
  - Shuffle data on network…

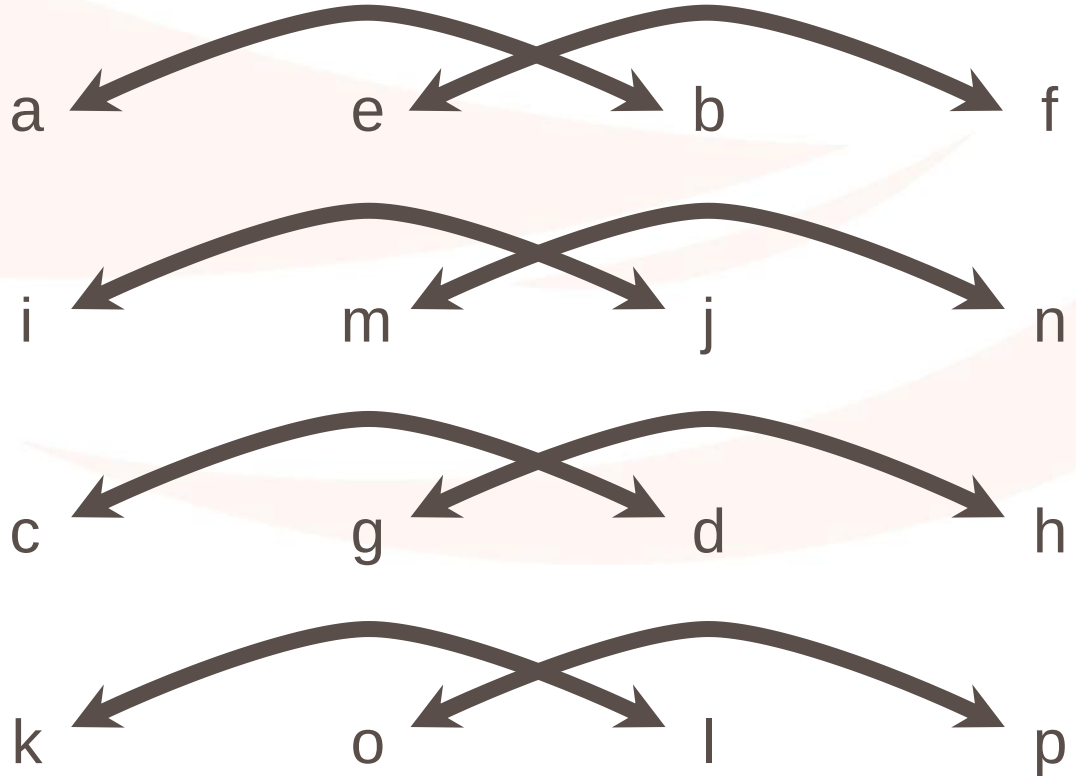
Solution: Input shuffle



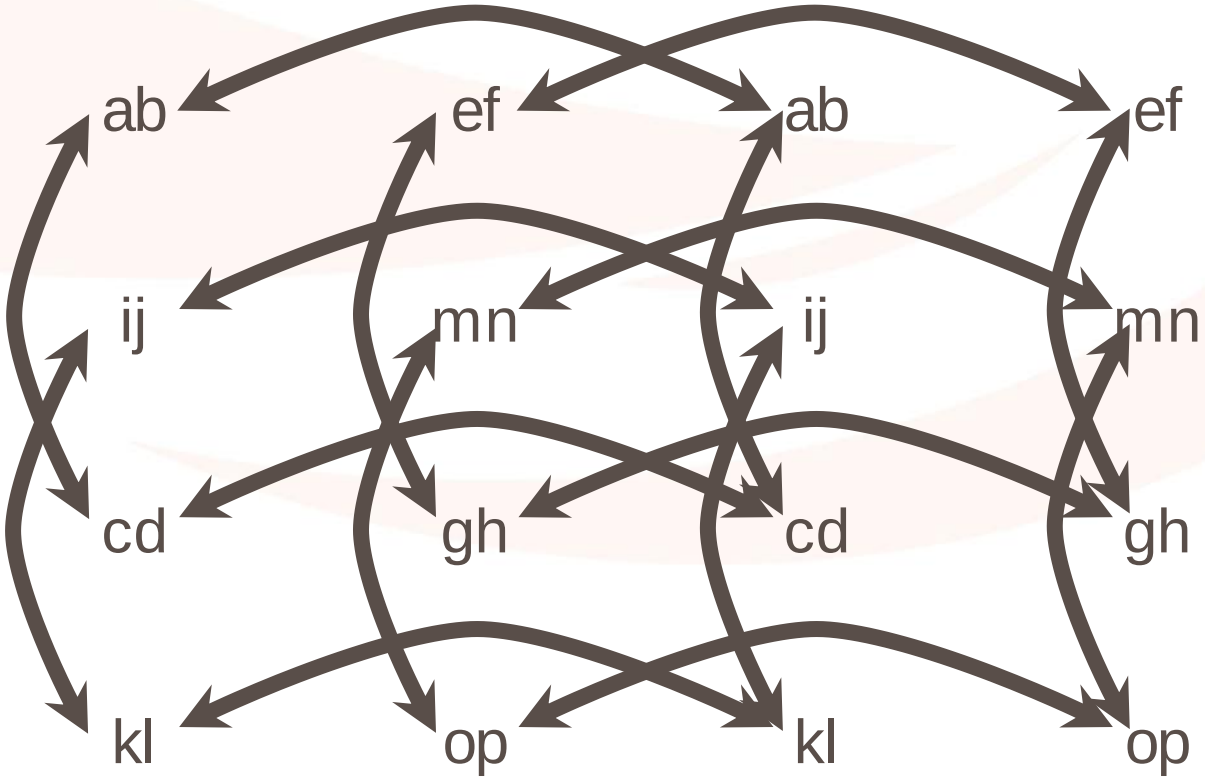
Solution: Input shuffle



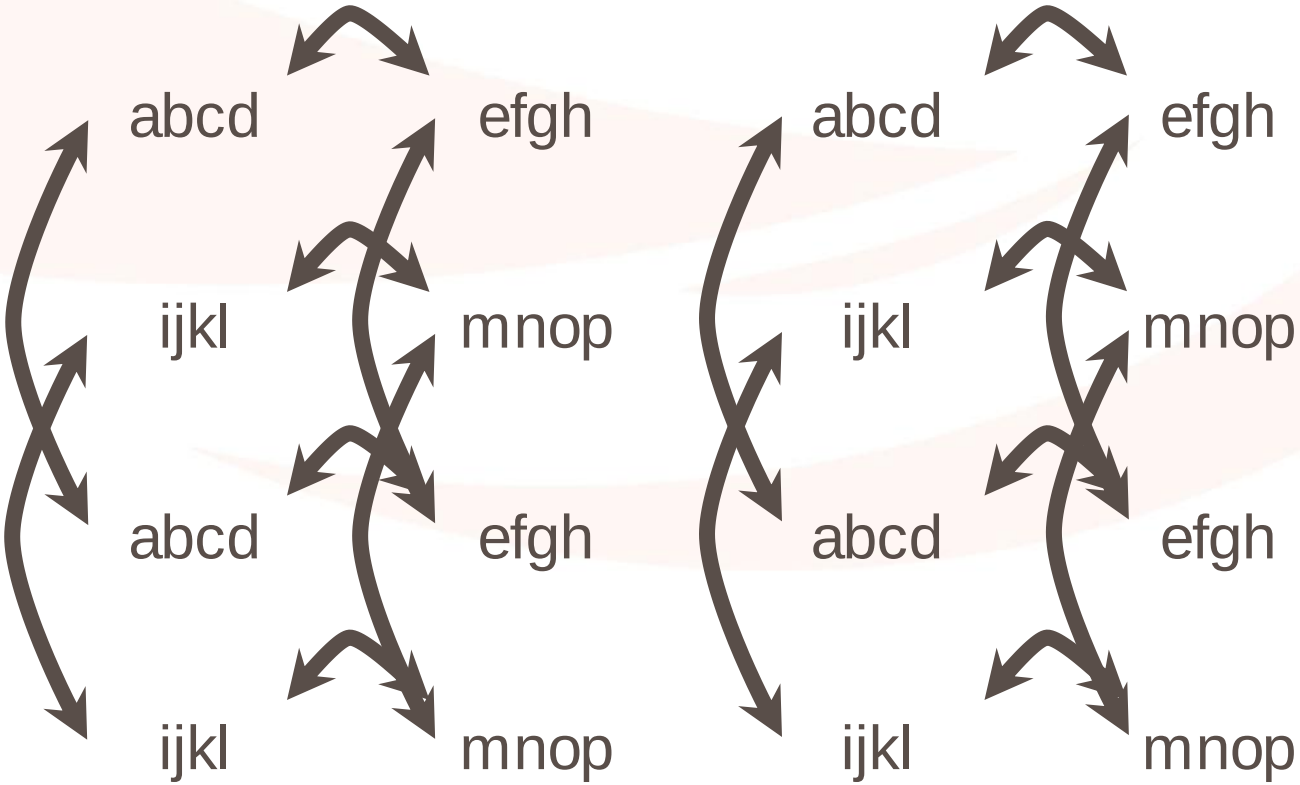
Solution: Input shuffle



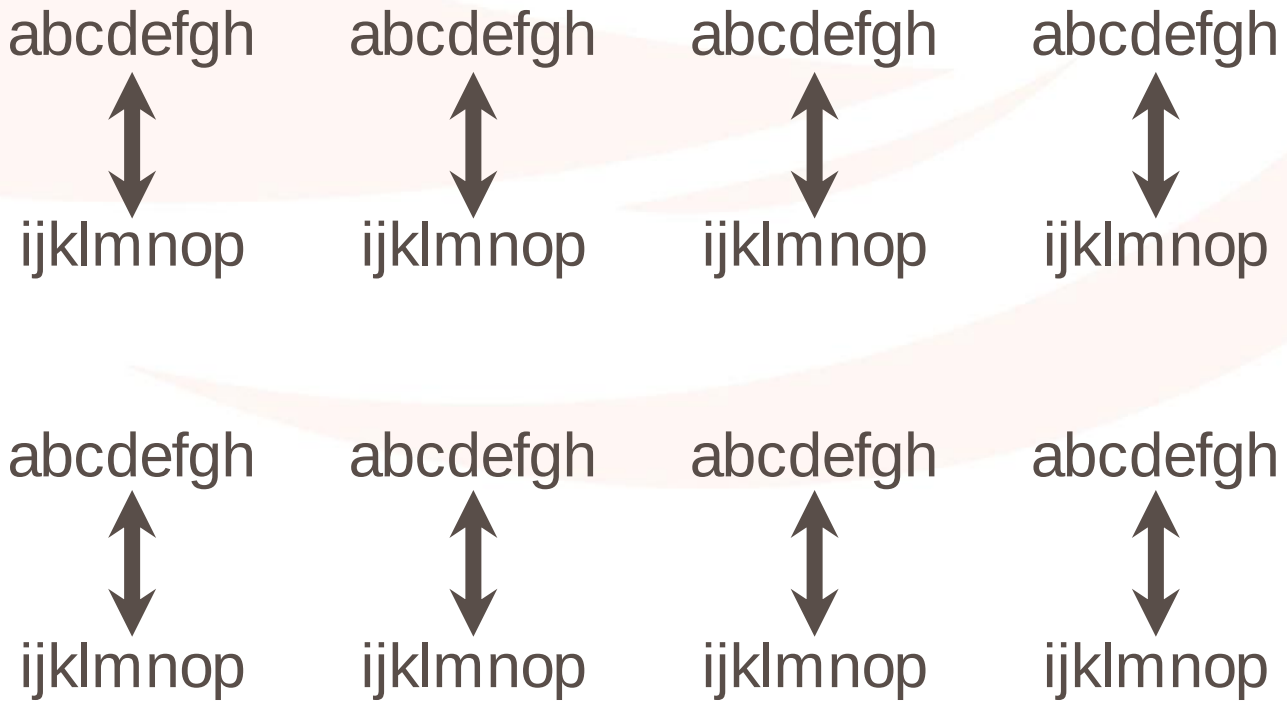
Solution: Input shuffle



Solution: Input shuffle



Solution: Input shuffle



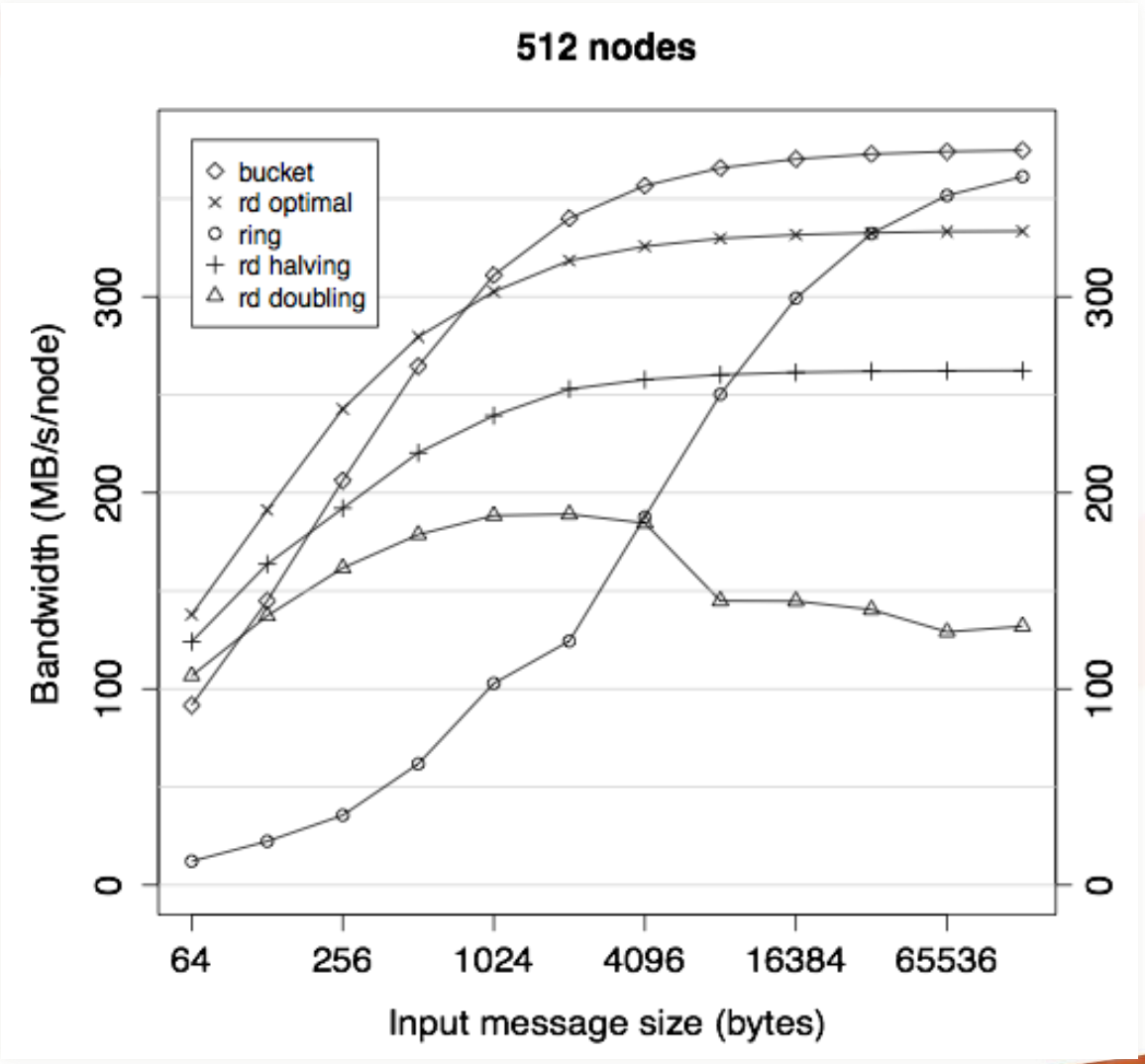
$$T=(1+lgP)\,\alpha + (7/6)nP\beta$$

$$T\approx(lgP)\alpha + (7/6)nP\beta$$

Evaluation: Intrepid BlueGene/P at ANL

- 40k-node system
  - Each is 4 x 850 MHz PowerPC 450
- 512+ nodes is 3d torus; fewer is 3d mesh
- XLC -O4
- 375 MB/s delivered per link
  - 7% penalty using all 6 links both ways

Allgather performance



Notes on Allgather

- Bucket algorithm (not described here) exploits multiple communication engines on BG
- *Analysis shows performance near optimal*
- Alternative to reorder data step is in memory move; analysis shows similar performance and measurements show reorder step faster on tested systems

Synchronization and OS Noise

- Load imbalance due to many factors:
  - “OS noise”
  - Programming model runtime
  - Memory waits (cache misses, refresh cycles)
  - Compounded by timing effects
- Synchronization delays due to communication (e.g., collective, halo exchange) shows up as slow communication; scalability issues

Saving Allreduce

- One common suggestion is to avoid using Allreduce
  - But algorithms with dot products are among the best known
  - Can sometimes aggregate the ate to reduce the number of separate Allreduce operations
  - But better is to reduce the impact of the synchronization by hiding the Allreduce behind other operations (in MPI, using MPI_Iallreduce)
- We can adapt CG to nonblocking Allreduce with some added floating point (but perhaps little time cost)

The Conjugate Gradient Algorithm

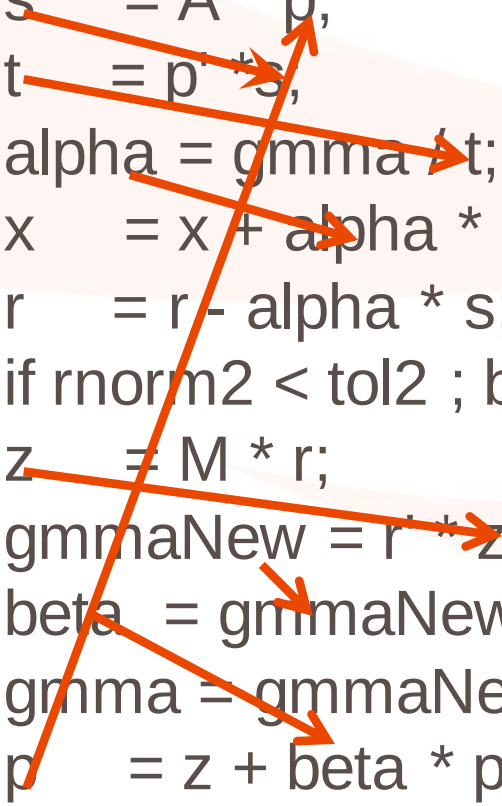
- While (not converged)

```
  niters += 1;
  s     = A * p;
  t     = p' *s;
  alpha = gmma / t;
  x     = x + alpha * p;
  r     = r - alpha * s;
  if rnorm2 < tol2 ; break ; end
  z     = M * r;
  gmmaNew = r' * z;
  beta  = gmmaNew / gmma;
  gmma = gmmaNew;
  p     = z + beta * p;
end
```

The Conjugate Gradient Algorithm

- While (not converged)

```
  niters += 1;
  s     = A * p;
  t     = p' * s;
  alpha = gmma / t;
  x     = x + alpha * p;
  r     = r - alpha * s;
  if rnorm2 < tol2 ; break ; end
  z     = M * r;
  gmmaNew = r' * z;
  beta  = gmmaNew / gmma;
  gmma = gmmaNew;
  p     = z + beta * p;
end
```



A Nonblocking Version of CG

- While (not converged)

```
 niters += 1;
 s     = Z + beta * s;
 % Can begin p'*s
 S     = M * s;
 t     = p' *s;
 alpha = gmma / t;
 x     = x + alpha * p;
 r     = r - alpha * s;
 % Can move this into the subsequent dot product
 if rnorm2 < tol2 ; break ; end
 z     = z - alpha * S;
 % Can begin r'*z here (also begin r'*r for convergence test)
 Z     = A * z;
 gmmaNew = r' * z;
 beta  = gmmaNew / gmma;
 gmma  = gmmaNew;
 % Could move x = x + alpha p here to minimize p moves.
 p     = z + beta * p;
end
```

CG Reconsidered

- By reordering operations, nonblocking dot products (MPI_Iallreduce in MPI-3) can be overlapped with other operations
- Trades extra local work for overlapped communication
  - On a pure floating point basis, the nonblocking version requires 2 more DAXPY operations
  - A closer analysis shows that some operations can be merged
- *More work does not imply more time*

Processes and SMP nodes

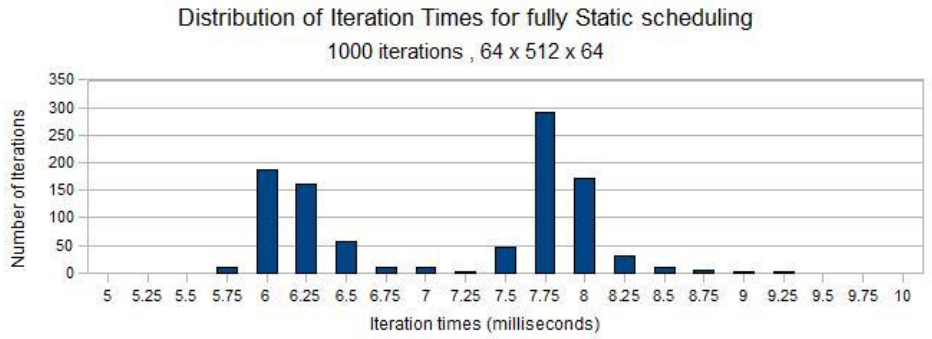
- HPC users typically believe that their code “owns” all of the cores all of the time
  - The reality is that was never true, but they did have all of the cores the same fraction of time when there was one core /node
- We can use a simple performance model to check the assertion and then use measurements to identify the problem and suggest fixes.
- Consider a simple Jacobi sweep on a regular mesh, with every core having the same amount of work. How are run times distributed?

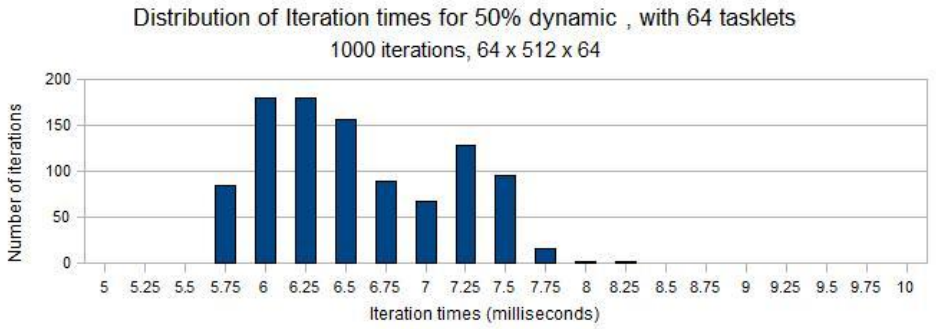
New (?) Wrinkle – Avoiding Jitter

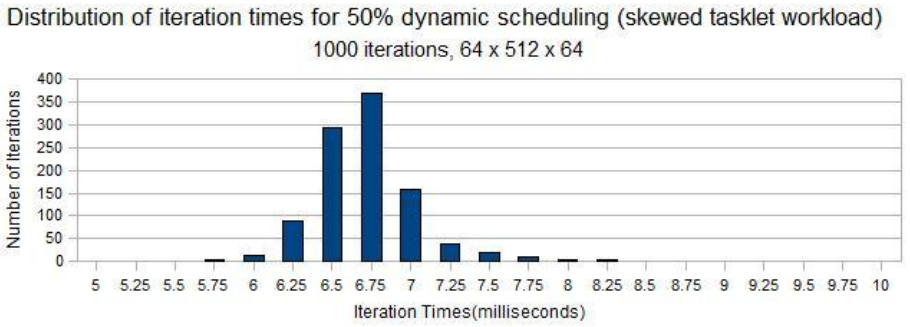
- Jitter here means the variation in time measured when running identical computations
  - Caused by other computations, e.g., an OS interrupt to handle a network event or runtime library servicing a communication or I/O request
- This problem is in some ways less serious on HPC platform, as the OS and runtime services are tuned to minimize impact
  - However, cannot be eliminated entirely

Sharing an SMP

- Having many cores available makes everyone think that they can use them to solve other problems (“no one would use all of them all of the time”)
- However, compute-bound scientific calculations are often *written* as if all compute resources are owned by the application
- Such *static* scheduling leads to performance loss
- Pure dynamic scheduling adds overhead, but is better
- Careful mixed strategies are even better
- Thanks to Vivek Kale

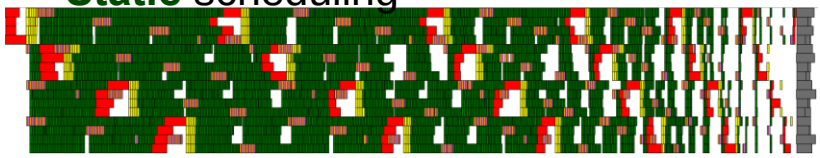




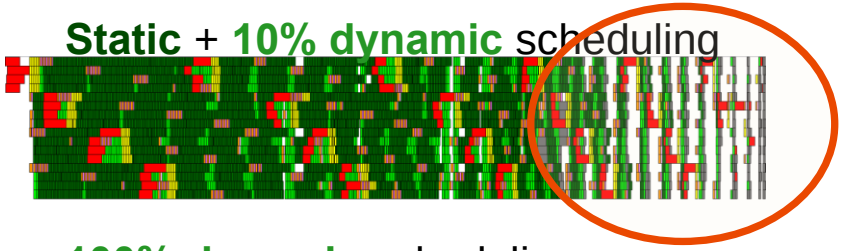


Happy Medium Scheduling

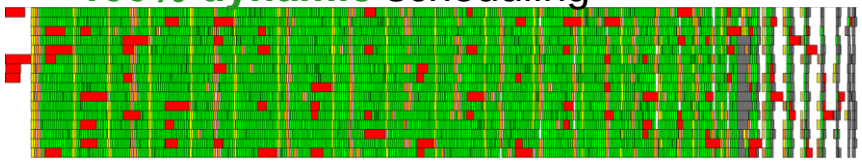
Static scheduling



Static + **10% dynamic** scheduling



100% dynamic scheduling



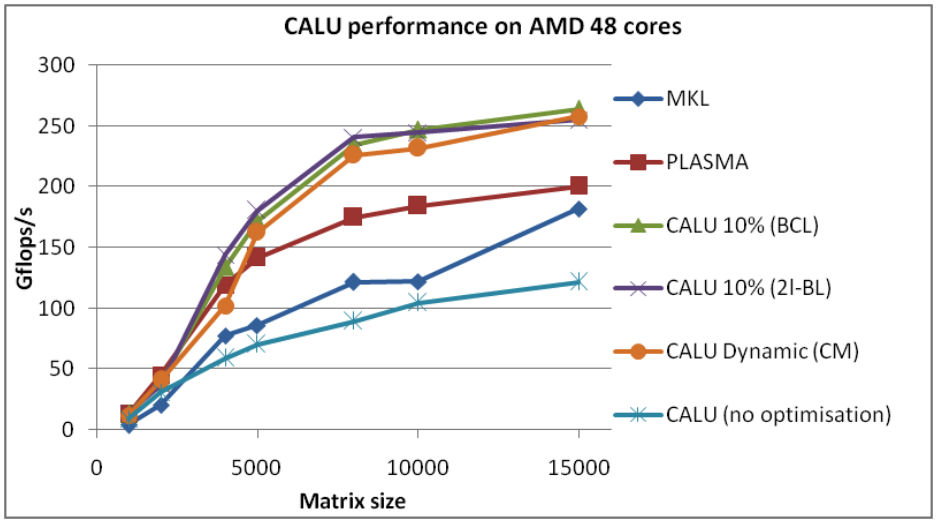
time

Scary Consequence: Static data decompositions *will not work at scale.*

Corollary: programming models with static task models *will not work at scale*

Performance irregularities introduce load-imbalance.
Pure dynamic has significant overhead; pure static too much imbalance.
Solution: combined static and dynamic scheduling

Communication Avoiding LU factorization (CALU) algorithm, S. Donfack, L .Grigori, V. Kale, WG, IPDPS '12



Experiences

- Paraphrasing either Lincoln or PT Barnum:

  You own some of the cores all of the time and all of the cores some of the time, but you don't own all of the cores all of the time

- Translation: a priori data decompositions that were effective on single core processors are no longer effective on multicore processors
- We see this in recommendations to "leave one core to the OS"
  - What about other users of cores, like … the runtime system?

Observations

- Details of architecture impact performance
  - Performance models can guide choices but must have enough (and only enough) detail
  - These models need only enough accuracy to guide decisions, they do not need to be predicitive
- Synchronization is the enemy
- Many techniques have been known for decades
  - We should be asking why they aren't used, and what role development environments should have

Some Final Questions

- Is it communication avoiding or minimum solution time?
- Is it communication avoiding or latency/communication hiding?
- Is it synchronization reducing or better load balancing?
- Is it the programming model, its implementation, or its use?
- How do we answer these questions?