Reproducibility of Computations and Distributed Data Structures

William Gropp
www.cs.illinois.edu/~wgropp

Reproducibility Issues

- Different order of evaluation can (but remember Jim’s talk) lead to different results – loss of bitwise identical reproducibility
- Two contributors to different ordering
  - ♦ Ordering induced by decomposition across memory domains
  - ♦ Ordering induced to provide maximum parallelism
- Not just an issue of MPI_Allreduce

Reproducibility and Accuracy

- Reproducibility means getting the same result bitwise independent of the number of processors used.
- This is not the same as computing an accurate solution
- This talk is concerned *only* with reproducibility
  - ♦ No claims about accuracy are made ☺

What Kind of Reproducibility?

- “The same result I got with my serial code”
  - ♦ Always possible, but may not be effectively parallel or efficient
- “The same result regardless of the number of processes”
  - ♦ This is the one I’m targeting, with an additional caveat:
    - For the different number of processes in which I’m interested
- Note: Reproducibility applies to the *entire* program
- Also assuming the same hardware and code choices by compiler

Example: Data Decomposition

- A typical computation starts with an expression of the serial computation:

  -
    ```
    Do i=1,n
       sum = sum + a(i)*b(i)
    ```
- Parallelizing to two processes gives

  -
    ```
    Do i=1,n/2
       sum = sum + a(i)*b(i)
    MPI_Allreduce(MPI_IN_PLACE,sum,…,
                               MPI_SUM,comm)
    ```

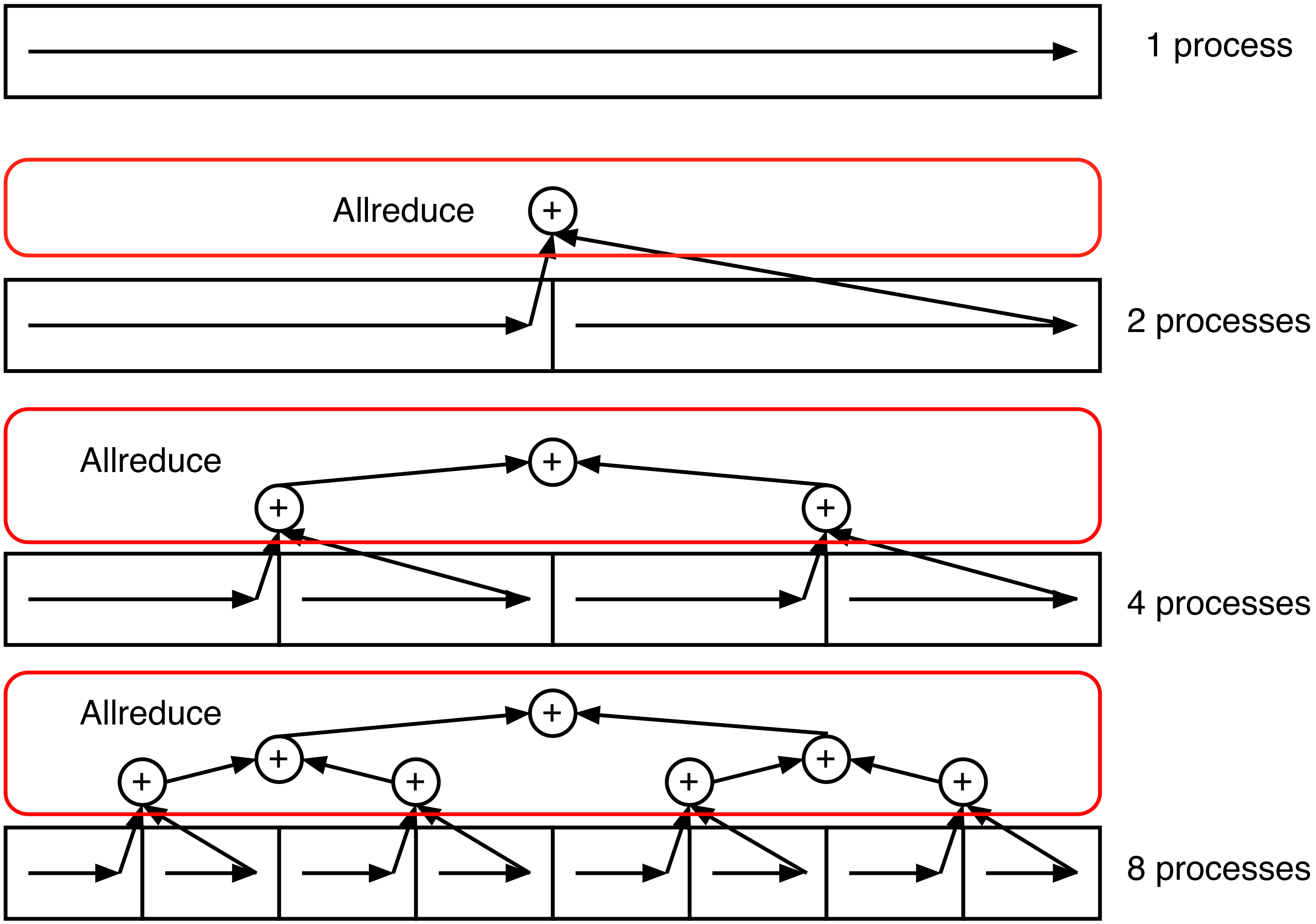
Simple Data Decomposition

- This follows the common practice of decomposing the data from a single global object (the vectors) to a collection of single local objects (the vector elements belonging to the process)
- This practice changes the order of evaluation, leading to the loss of bitwise reproducibility

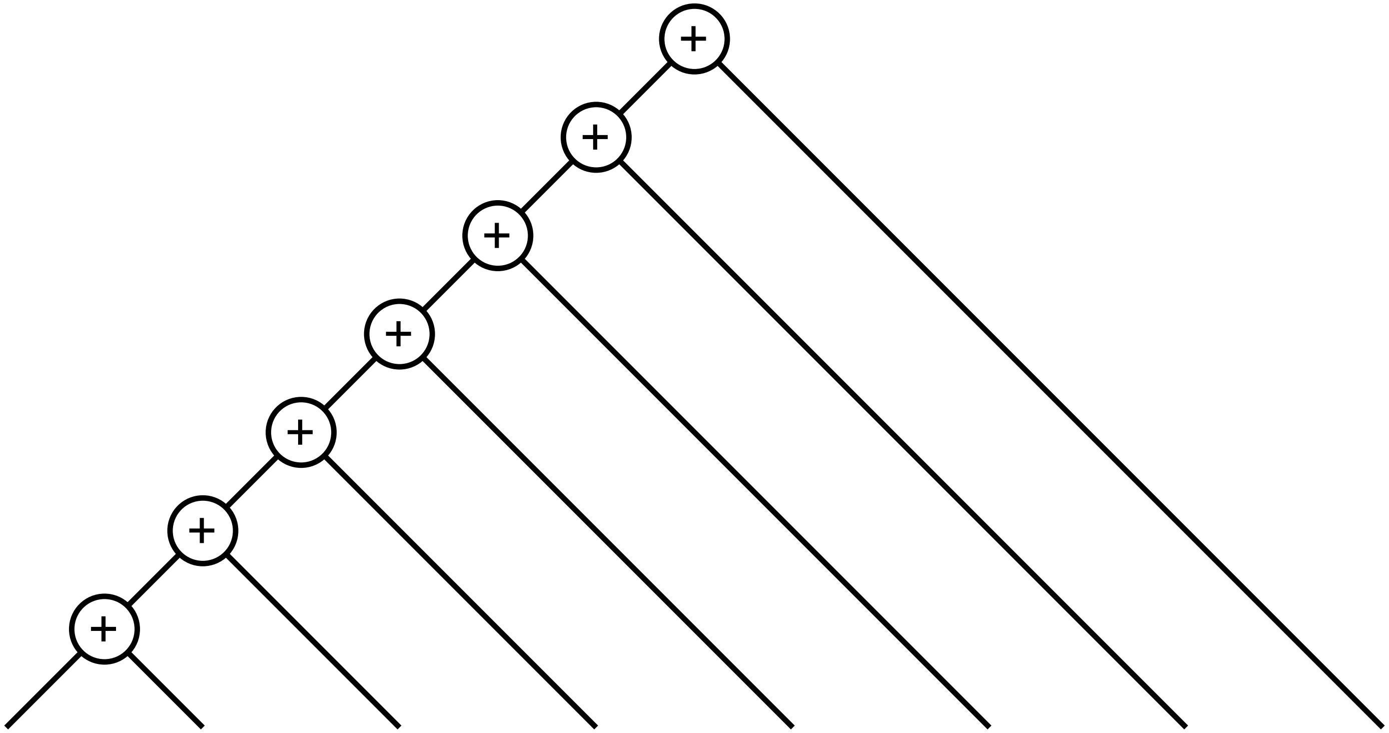
Simple Data Decomposition

- Assumptions:
  - ♦ Data divided into one block per process
  - ♦ Data processed first locally, then globally
    - E.g., first form local dot product, then use MPI_Allreduce to get global sum
- Neither of these is necessary or even a good idea…
  - ♦ Lets look at the sum reduction again

Reduction With Different Process Counts

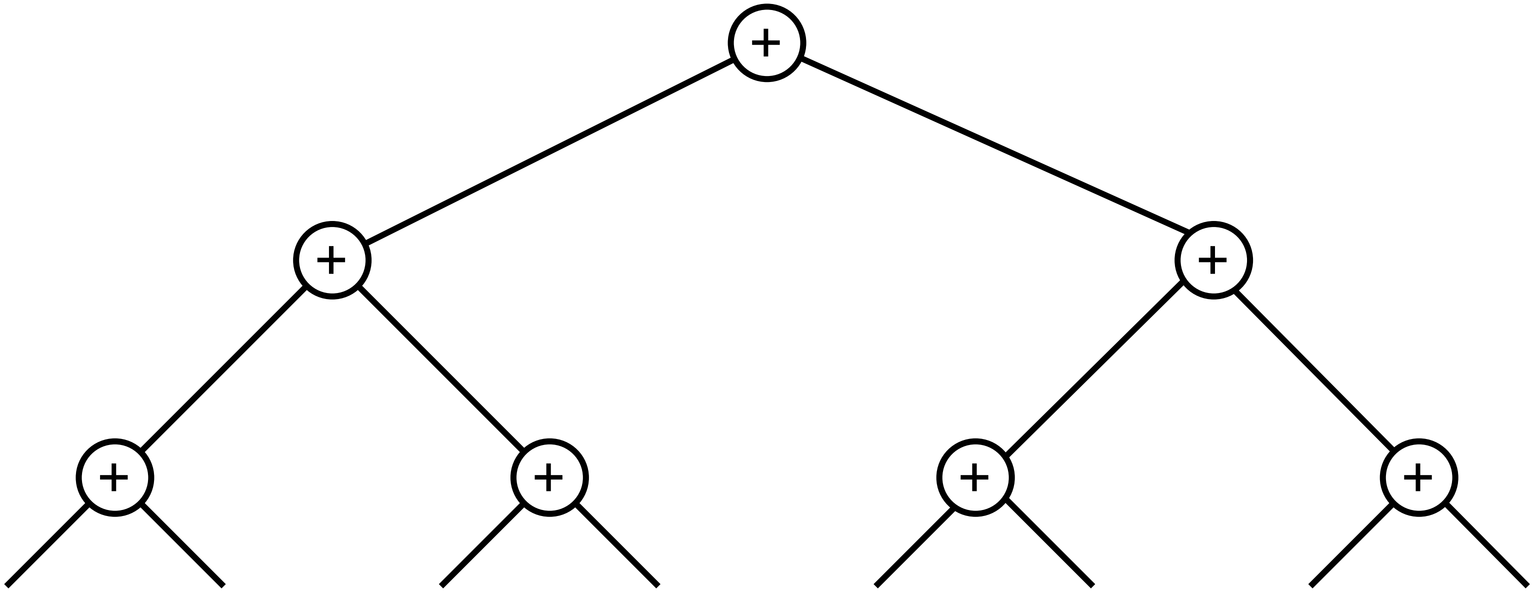


Typical Reduction Tree

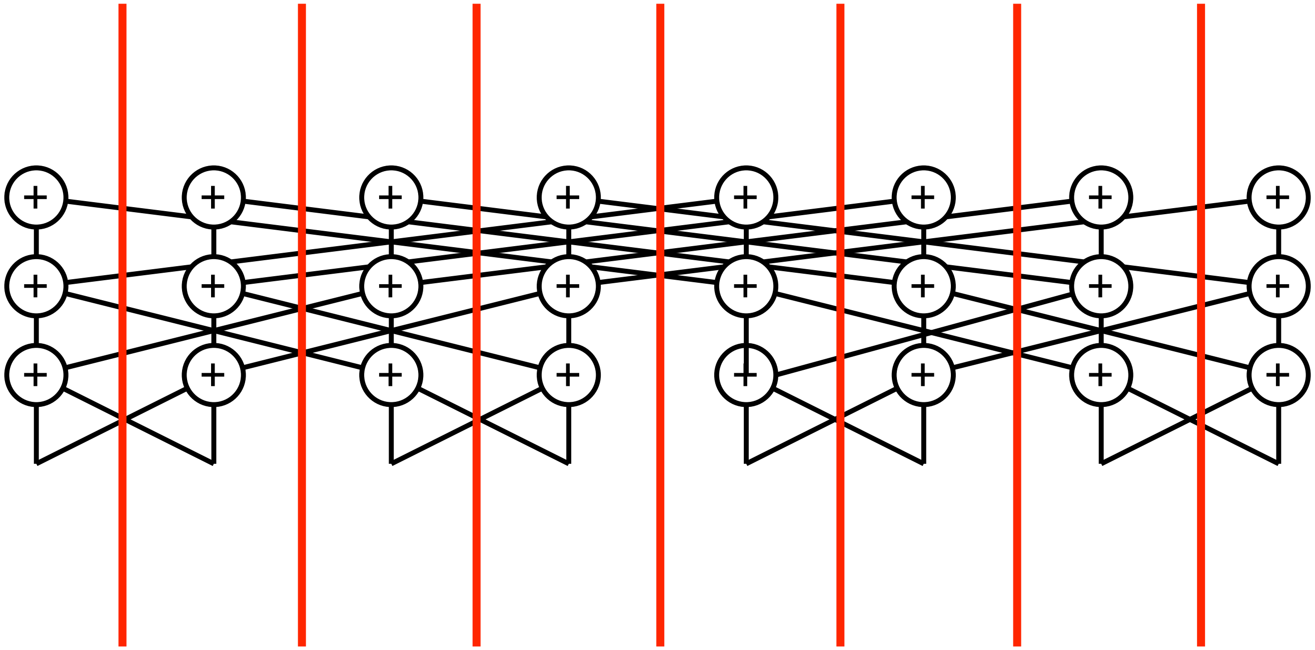


No parallelism, since every operation depends on results of a previous sum
"Centipede Tree"

Balanced Reduction Tree



Recursive Doubling Exchange

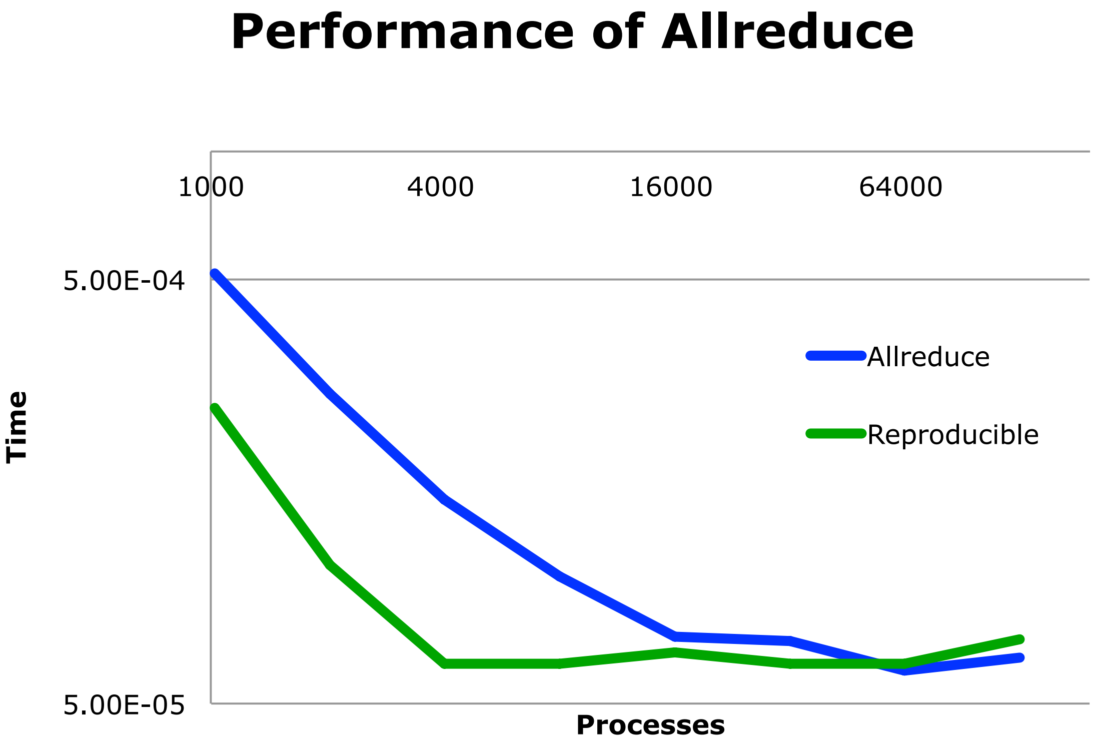


Offers parallelism, bitwise identical result independent of number of processes

One Approach to Reproducibility

- Define a single schedule for computing results *independent* of the number of processes.
  - ♦ Can ***always*** do this
    - How will determine efficiency, parallelism

A Reproducible Dot Product Can Be as Fast as a Simple Dot Product



- Strong scaling result to 128k ranks
- $N=2^{27}=134217728$

Notes on Reproducible Dot Product Experiment

- Example for 2^k processes for k=0,…,17
- Vector length 2^j for j>=k+10
  - Smallest block is 2^{10} elements
- Reproducible version faster because uses a more parallel local sum, giving better performance
  - Could do for the “Allreduce” one, but used simplest code
- Both become communication bound (vector rather short at a mere 128M)

An Alternate Design Approach

- Pick a single decomposition, independent of p
  - Have a maximum number of processes
  - May have a set of processes, e.g., 2^k
- Pick a schedule for computation on the decomposition, independent of p
  - But choose to maximize available parallelism
- With care, computation is now reproducible for all p (within set)

Relaxing the Schedule

- Using a different schedule may give better performance
  - ♦ Dynamic, adapt to different computation speeds, especially on SMP nodes
  - ♦ Some schedules produce bitwise identical results
    - Order of evaluation of blocks does not affect final result
- If (mostly) the same code, fewer places for bugs to reside

Comments for (Batched) BLAS

- Can’t fix reproducibility by *only* looking at parallel vector operations
  - ♦ Having a “reproducible allreduce” is not sufficient
- Data decomposition critical
  - ♦ One block per core/thread/process may not be the best choice
    - Offers other advantages, such as dynamic load balancing on SMPs, memory hierarchy optimizations, …
  - ♦ Good fit to using a small-tile approach
  - ♦ Choices span many (often all) routines
    - May make sense to use inspector/executor approaches
    - Requires an API with separate setup and execute routines

Conclusion

- Reproducibility (in terms of “independent of parallelism”) should be defined in terms of a set of # of processes and data decomposition
  - ♦ General case possible but (needlessly?) hard
- Overdecomposition combined with a deterministic, parallel-friendly schedule, provides a way to achieve the same operations, in the same order
  - ♦ Can relax the schedule requirements to trade performance for bit-wise reproducibility
- Overhead can be low
  - ♦ Demonstrated with dot product of distributed vectors

Thanks!

- Funding from:
  - ♦ NSF
  - ♦ Blue Waters
- Chris Gropp
  - ♦ For earlier (and more general) work