

Parallel Programming

Prof. **Paolo Bientinesi**

`pauldj@aices.rwth-aachen.de`

WS 15/16



High Performance and
Automatic Computing

RWTHAACHEN
UNIVERSITY



Back to Broadcast

- Lower bound: $\log_2(p)\alpha + n\beta$
- MST algorithm: $\log_2(p)\alpha + \log_2(p)n\beta$

Can we do better?

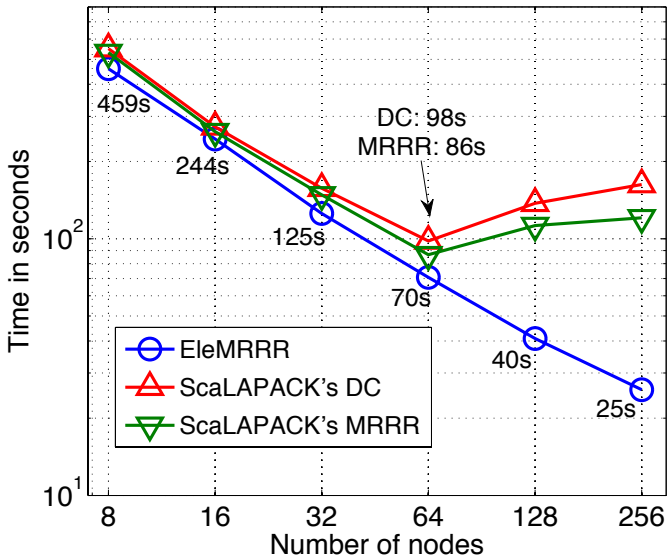
- Idea for large n : first Scatter, then Allgather
- Cost: $\log_2(p)\alpha + \frac{p-1}{p}n\beta + \log_2(p)\alpha + \frac{p-1}{p}n\beta$
 $\approx 2\log_2(p)\alpha + 2n\beta$
- Weight shifted from β to α ; a factor of 2 from optimal
- **In practice:** $p = r \times c \Rightarrow$ 2-stage algorithms

Proposed exercise: Allgather

- Implement an `Allgather` as a sequence of broadcasts.
- Time and compare your implementation with MPI's `Allgather`; test different message lengths and different numbers of ranks.

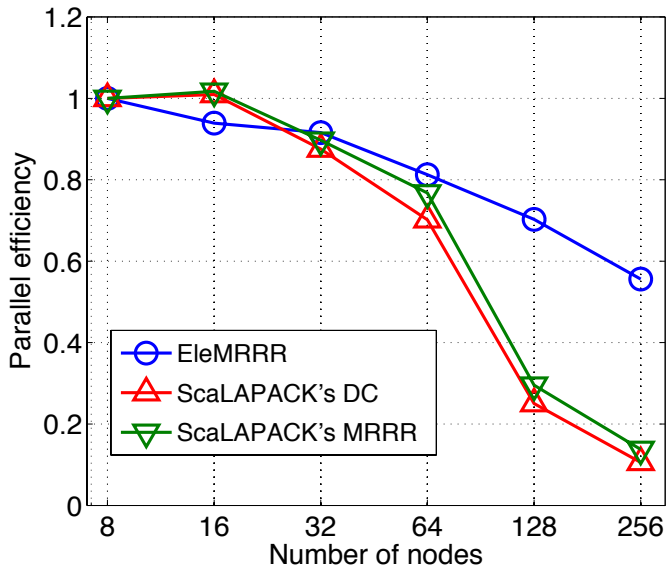
Strong scalability: timings

$n = 20,000$



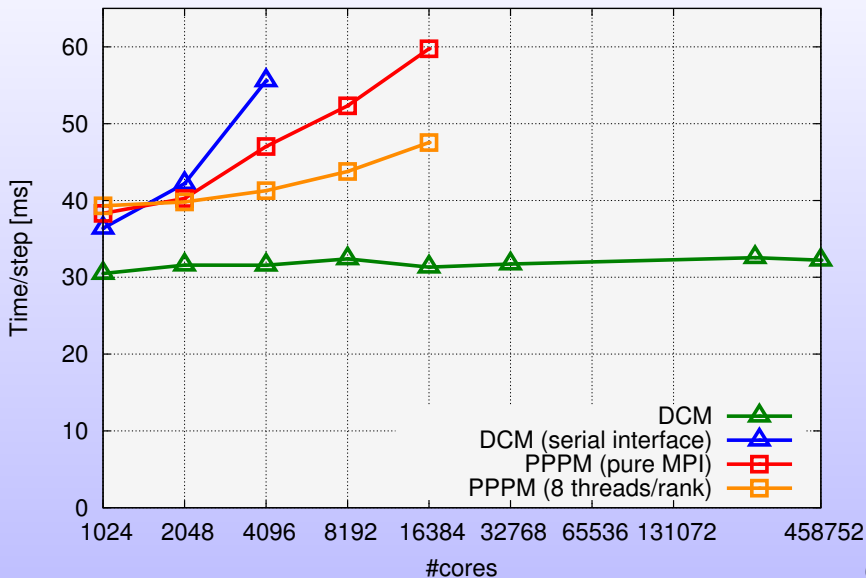
Strong scalability: efficiency

$n = 20,000$



Weak scalability: timings

10,000 particles per rank



Patterns for parallel computing

MPI's perspective

- Pipeline
 - Ok, but what if the different stages differ significantly in cost?
- Fork-join
 - Not really
- Master-slave
 - Yes, absolutely

Proposed exercise: master-slave

- Rank `root` has access to an input device.
`root` performs an infinite loop, reading from the device, sending tasks to all other ranks, and collecting (and printing) the results.
`root` does not execute tasks itself.
- Simulate the input device with a text file, containing a long list of integers (one per row); each integer denotes the “length” (size) of a chunk of input data.
The idea is that `root` either reads or creates such data, and sends it to one worker, for processing.
- The workers execute an infinite loop, waiting for an incoming chunk data, processing it, and sending the result to `root`. (Alternatively, the results are handled by another designated process `dest`).
- Objective: minimize (eliminate) wait times.