Lecture 11

Memory hierarchy performance
Advanced Collective Communication

Announcements

- No class on 11/17 and 11/19
- Makeup lectures
  - Friday 11/20. 3:00 to 4:20
  - Weds 12/2   5-7PM
- CSE 260 Symposium
  - Week 10: Tues, Weds, Thurs
- Assignment return and feedback

A3 – GPU Technology Track

- Use single precision
- Baseline can be a single CPU core, but in that case I have raised the bar
  - $\times 10$ performance improvement for groups of 2
  - $\times$ 20 for groups of 3
  - Use a problem of size 512^3
- Report timing both with and without transfer between device and host

Measuring performance

- Two ways
- Use Cuda events/elapsed time
- Use an ordinary timer, e.g. gettimeofday()
- See $CUDA_Examples/withTimer
- Note that kernel invocation is asynchronous

```
cudaThreadSynchronize();
double t_device_compute = -getTime();
      COMPUTE
cudaThreadSynchronize();
 t_device_compute +=getTime();
```

CUDA Error Handling

- Cuda can silently fail, you can observe misleading performance
- E.g. if you specify an invalid grid / thread block dimensions
- Beware that the last error can be cleared by sucessive kernel calls, so check frequently

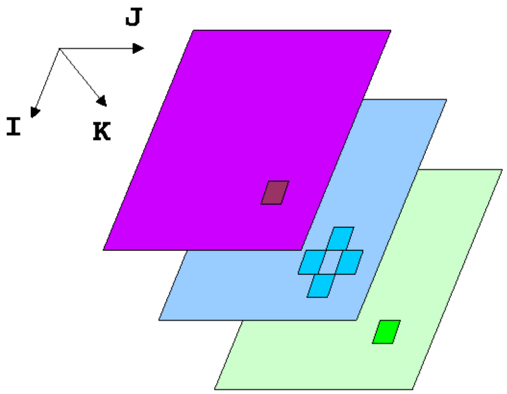
```
assert(cudaSuccess == cudaMalloc((void **) &a_d, size));
printf("Cuda error: %s\n", cudaGetErrorString(cudaGetLastError()));
```

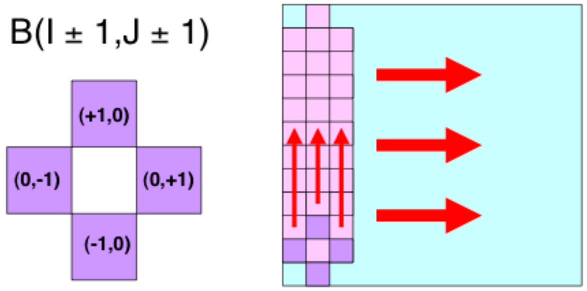
- What about asynchronous calls?
- cf CUDA Programming Guide, “Error Handling”

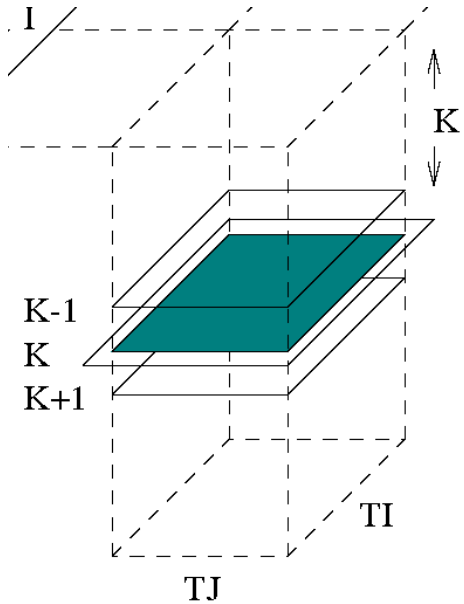
Memory Performance Issues in Stencil Methods

- Processor Geometry
- Cache misses – conflicts
- Sharing among threads
- False sharing

Stencil Memory access patterns

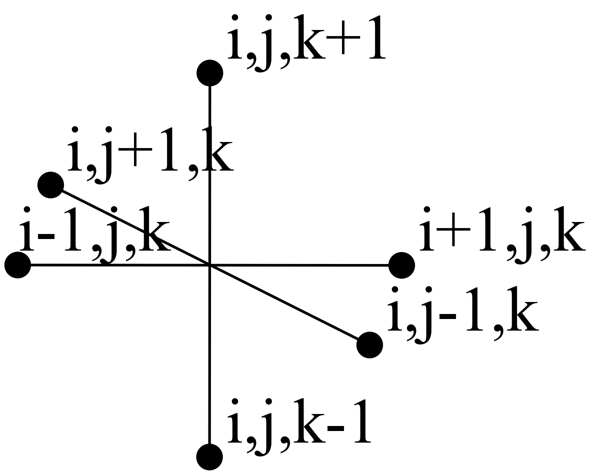


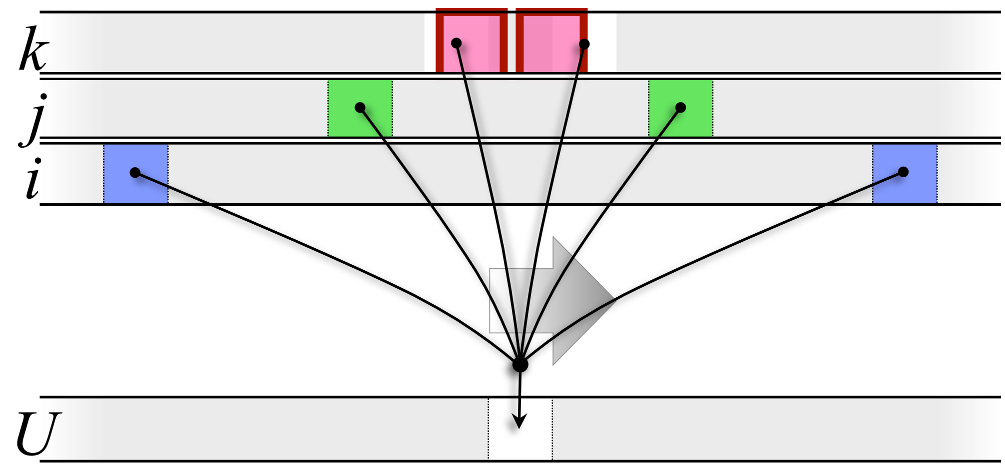




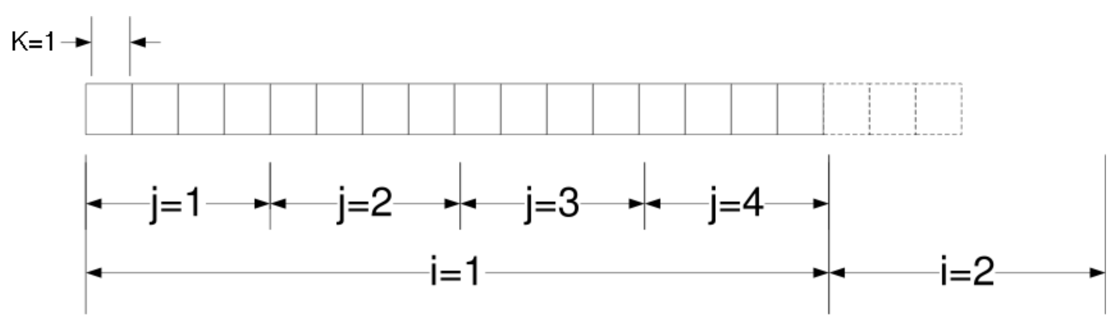
Rivera and Tseng

Memory Strides



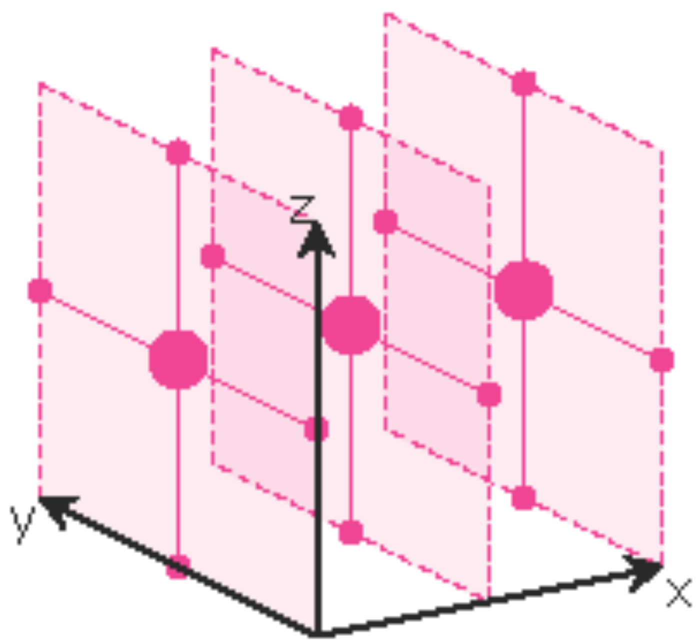


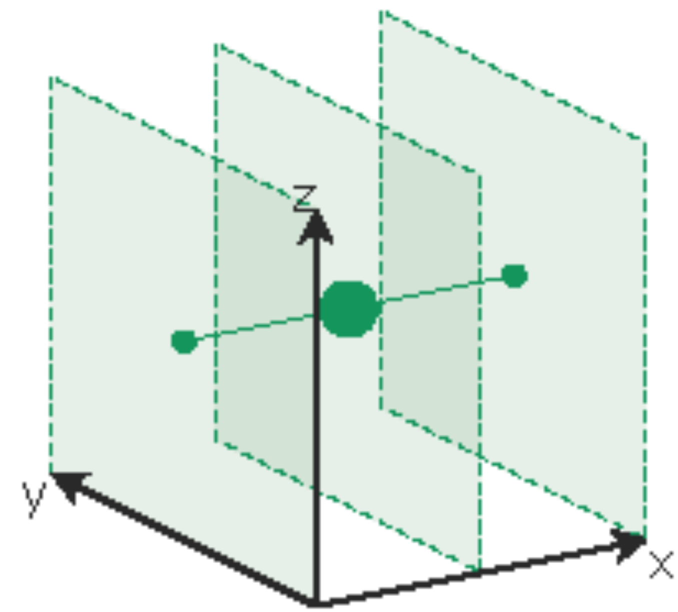
Sam Williams et al.



H. Das, S. Pan, L. Chen

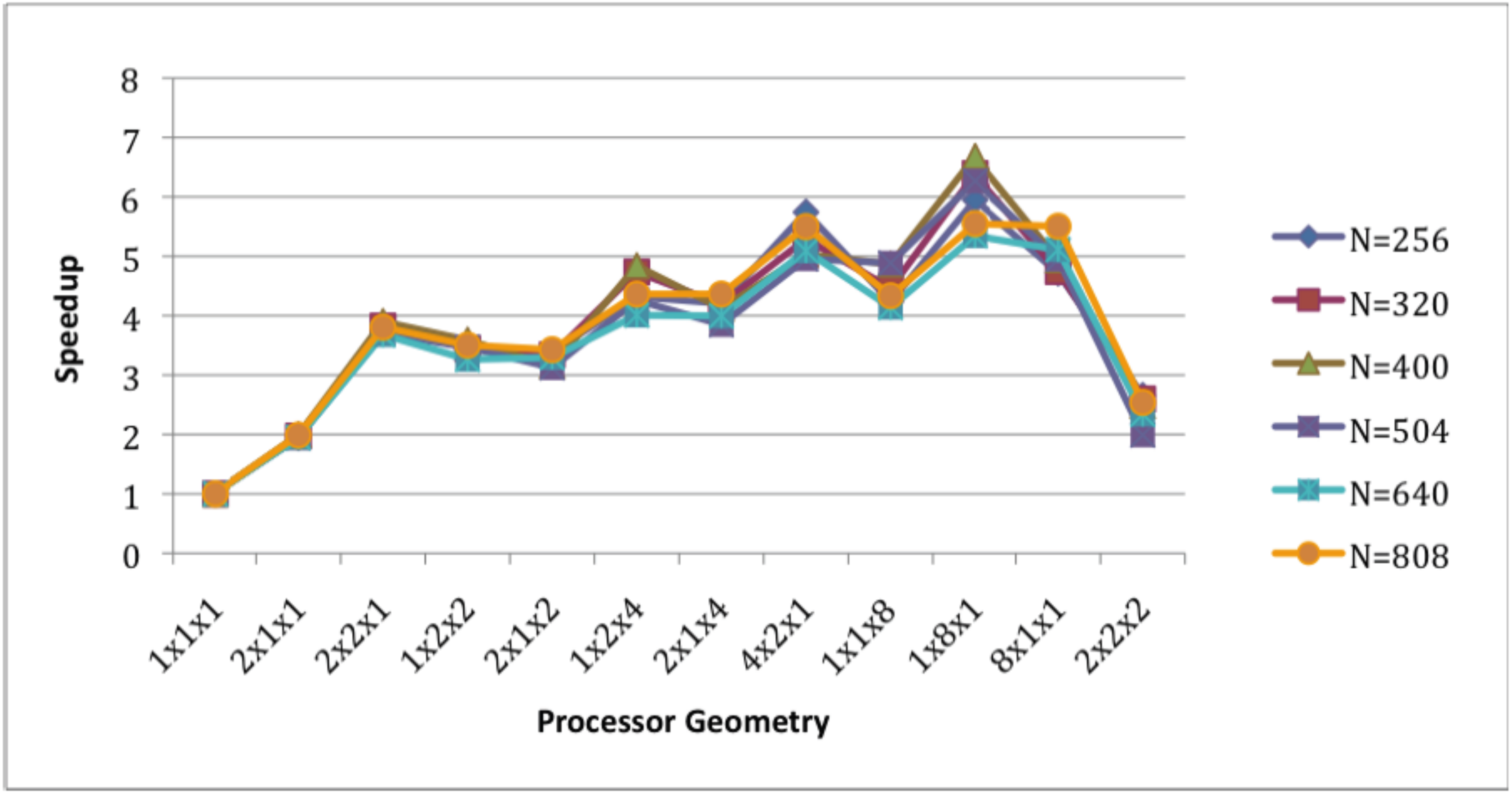
Stencil Patterns in memory



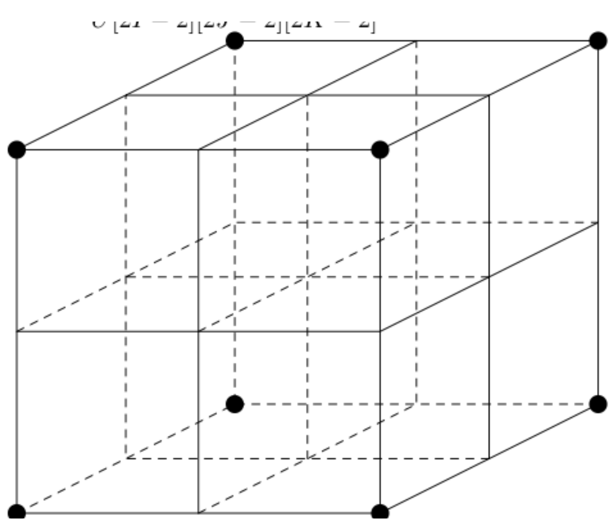


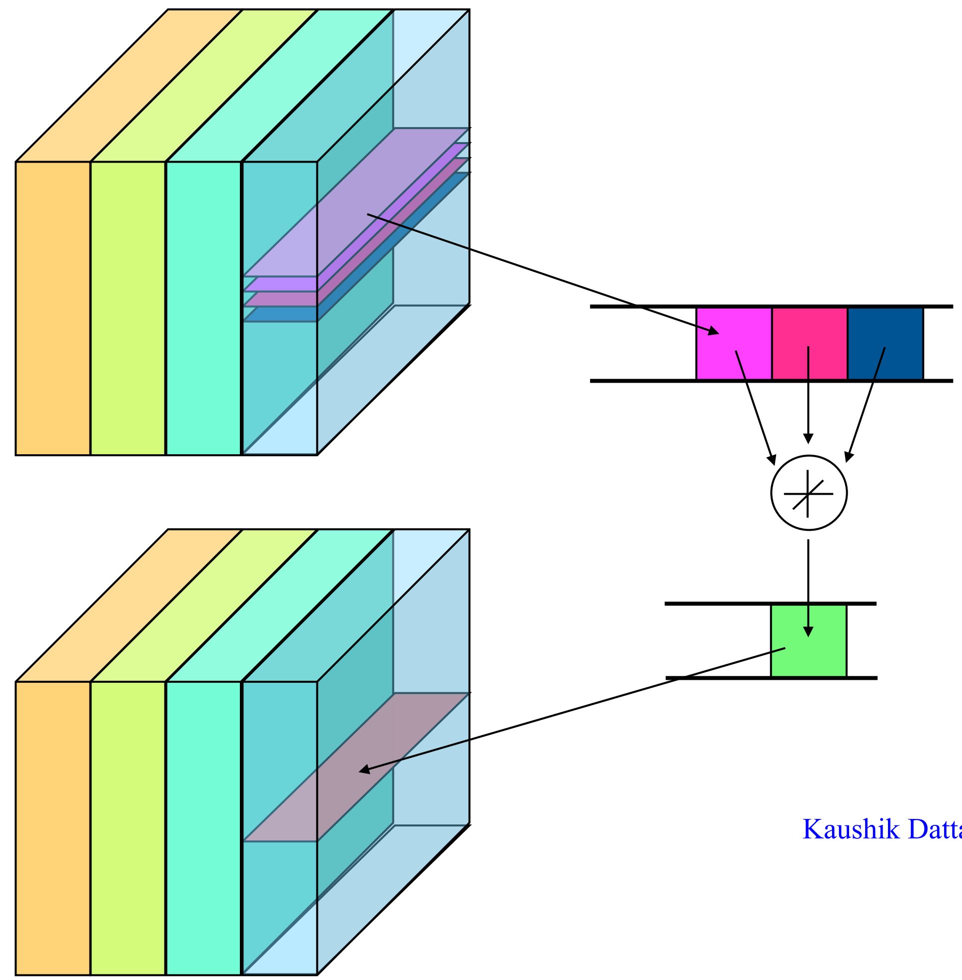
Sam Williams

Processor Geometry



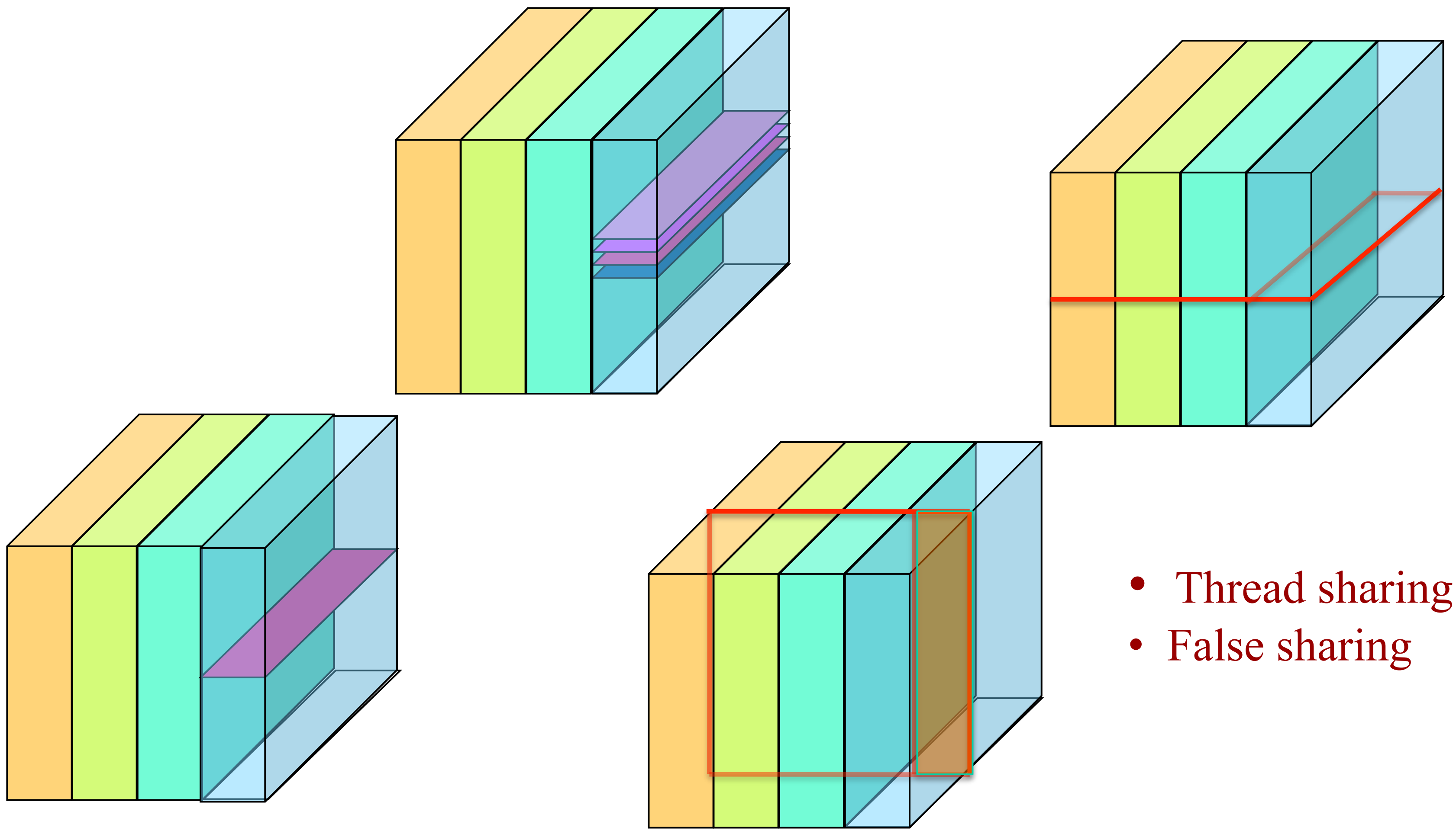
Interactions between threads





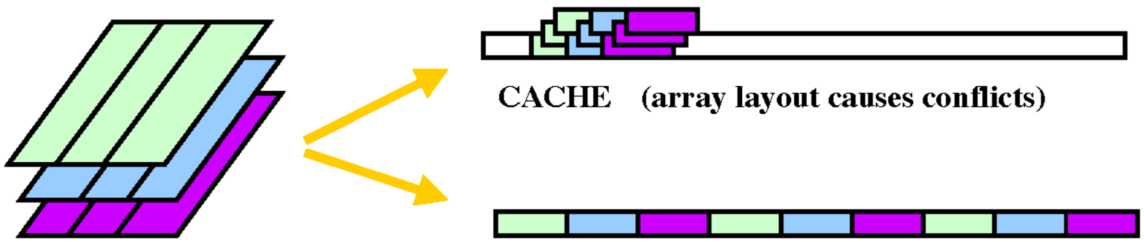
Kaushik Datta

Interactions between threads



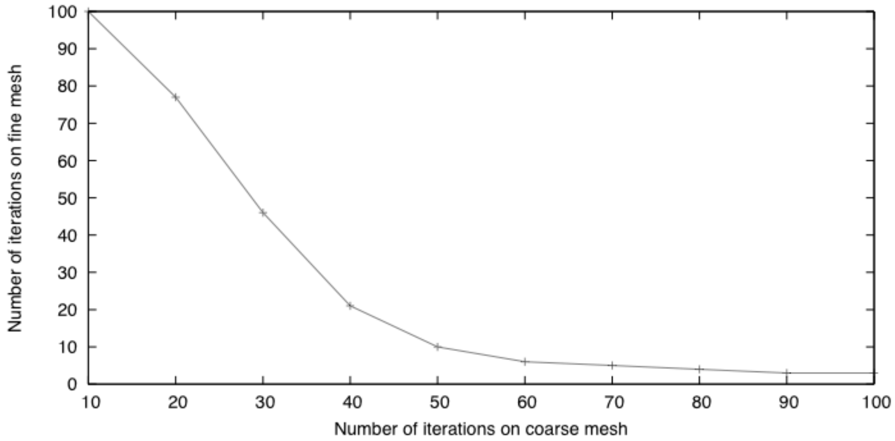
- Thread sharing
- False sharing

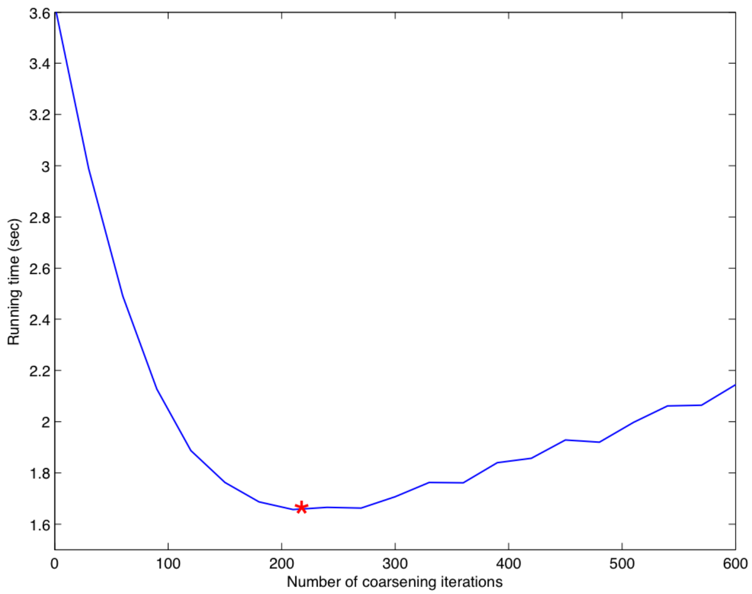
Cache Conflicts



Rivera and Tseng

Acceleration





Performance

- For a problem that fits in L3, but not in L2 (or L1)

  $56^3 = 56*56*56*8*3 = 4{,}214{,}784$

- ./jac3d -N 56 -i 1000 -px X -py Y -pz Z
  - {1,1,1}: 0.813 Gflops
  - {1,2,1}: 1.581 Gflops
  - {1,4,1}: 3.084 Gflops
  - {1,4,2}: 4.711 Gflops

Collective Communication

Collective communication

- Arises in many applications
  - Fast Fourier Transform
  - Sorting
- Collective operations are called by **all** processes within a communicator
- Simplest collectives
  - Broadcast: distribute data from a designated *root* process to all the others
  - Reduce: combine data from all processes returning the result to the root process
- Other Useful collectives
  - Scatter/gather
  - All to all
  - Allgather

Underlying assumptions

- Fast interconnect structure
  - All nodes are equidistant
  - Single-ported, bidirectional links
- Communication time is $\alpha + \beta n$ in the absence of contention
  - Determined by bandwidth β^{-1} for long messages
  - Dominated by latency α for short messages

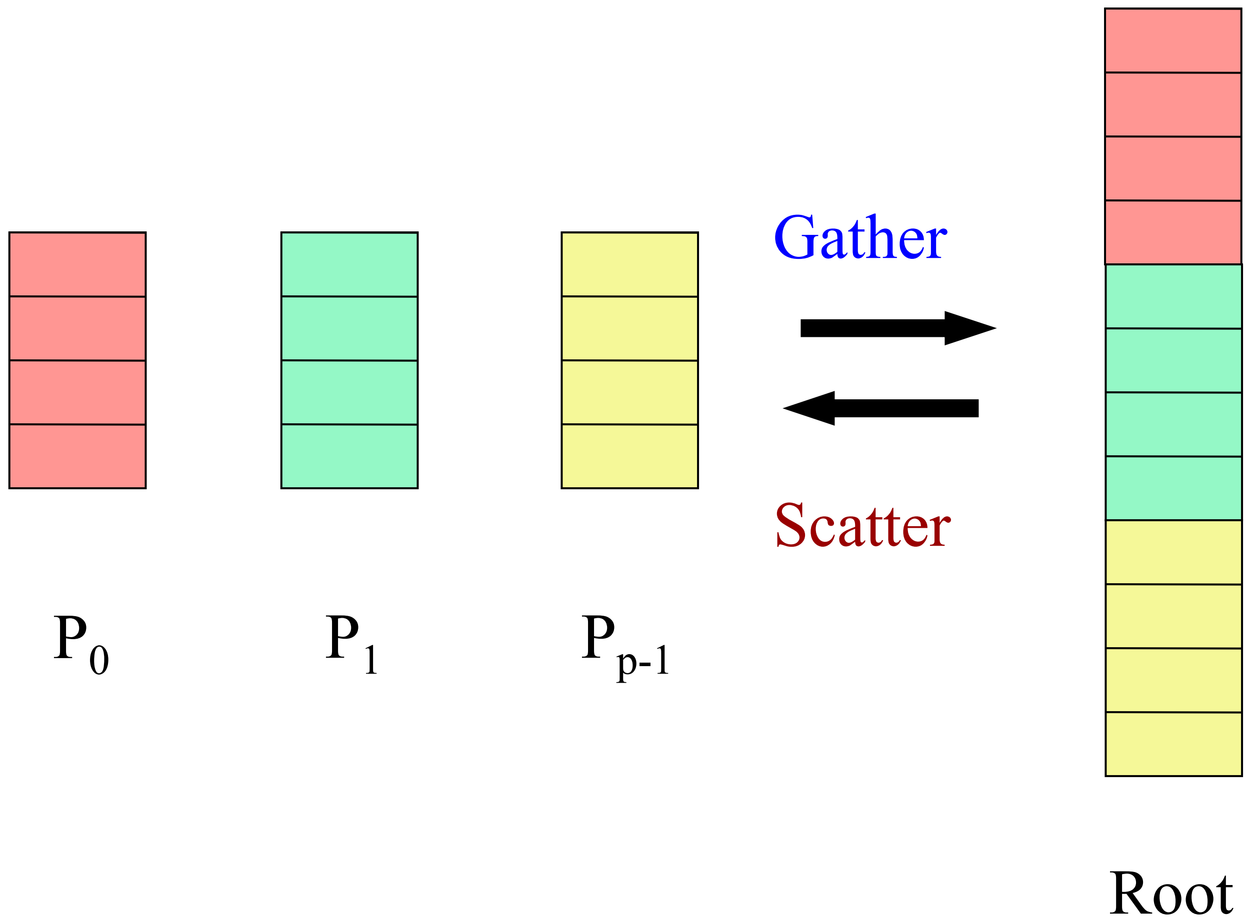
Inside MPI-CH

- Tree like algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
- Block size may be configured at installation time
- If there is hardware support, then it is given responsibility to carry out the broadcast
- Polyalgorithms apply different algorithms to different cases, i.e. long vs. short messages, different machine configurations
- We'll use hypercube algorithms to simplify the special cases when $P=2^k$, k an integer

Details of the algorithms

- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast

Scatter/Gather family



Scatter

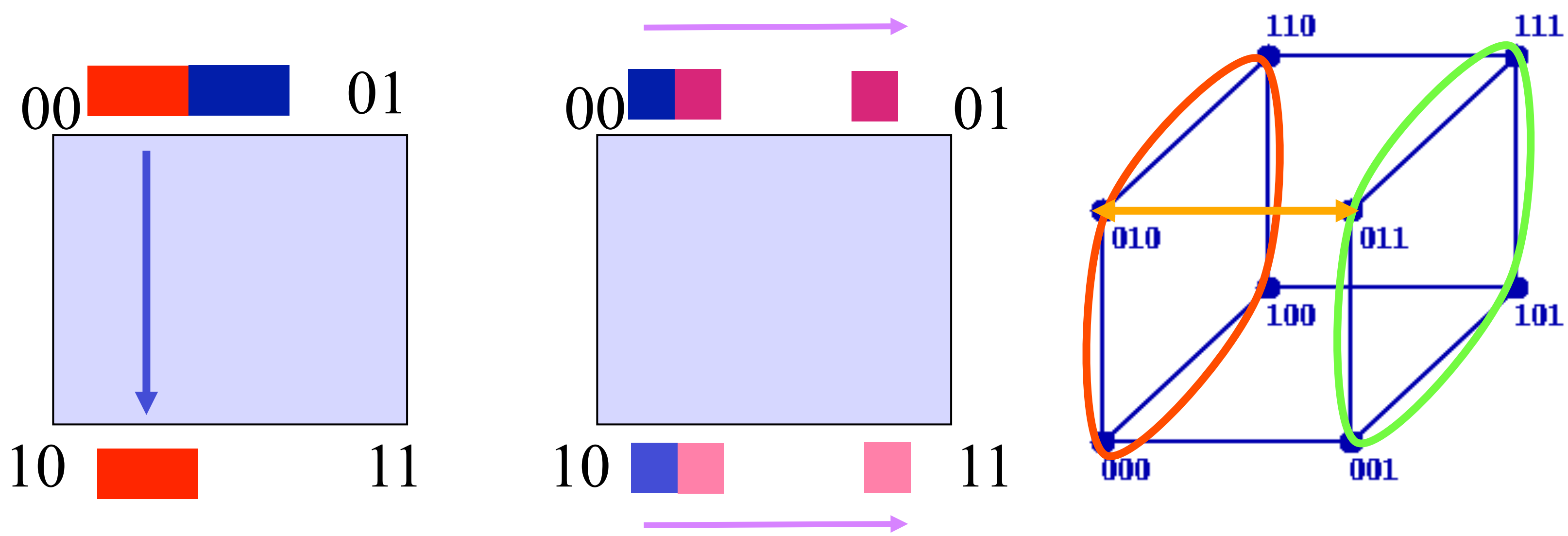
- Simple linear algorithm
  - Each processor sends a chunk of data to all others
  - Reasonable for long messages

$$(p-1)\alpha + \frac{p-1}{p} n\beta$$

- Similar approach taken for Reduce and Gather
- For short messages, we need to reduce the complexity of the latency (α) term

Minimal spanning tree algorithm

- Recursive hypercube-like algorithm with ⌈log P⌉ steps
  - Root sends half its data to process (root + p/2) **mod** p
  - Each receiver acts as a root for corresponding half of the processes
- Running time: $\lceil \lg P \rceil \alpha + \frac{p-1}{p} n\beta$
- Requires O(n/2) buffer space

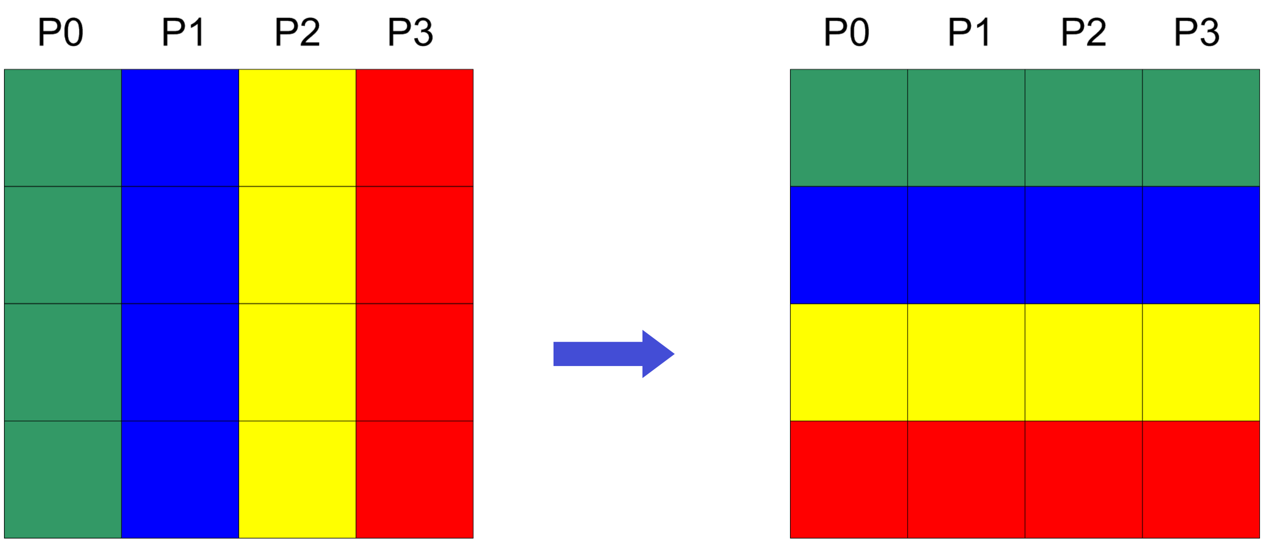


Details of the algorithms

- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast

All to all

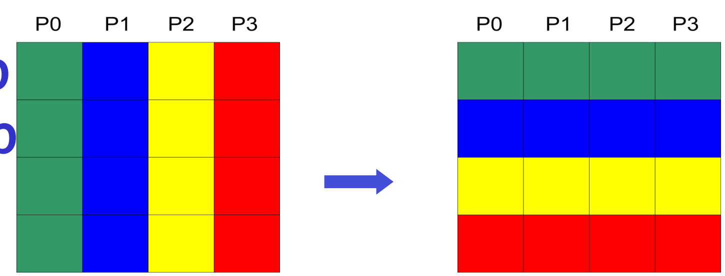
- Also called *total exchange* or *personalized communication:* a transpose
- Each process sends a different chunk of data to each of the other processes
- Used in sorting and the Fast Fourier Transform



Exchange algorithm

- n elements / processor (n total elements)
- p - 1 step algorithm
  - Each processor exchanges n/p elements with each of the others
  - In step i, process k exchanges with processes $k \pm$ i

```
for i = 1 to p-1
   src  =  (rank - i + p)   mod p
   dest =  (rank + i     )   mod p
   sendrecv( from src to dest )
end for
```



- Good algorithm for long messages
- Running time:

$$(p-1)\alpha \; + \; (p-1)\frac{n}{p}\beta \;\; \approx \;\; n\beta$$

Recursive doubling for short messages

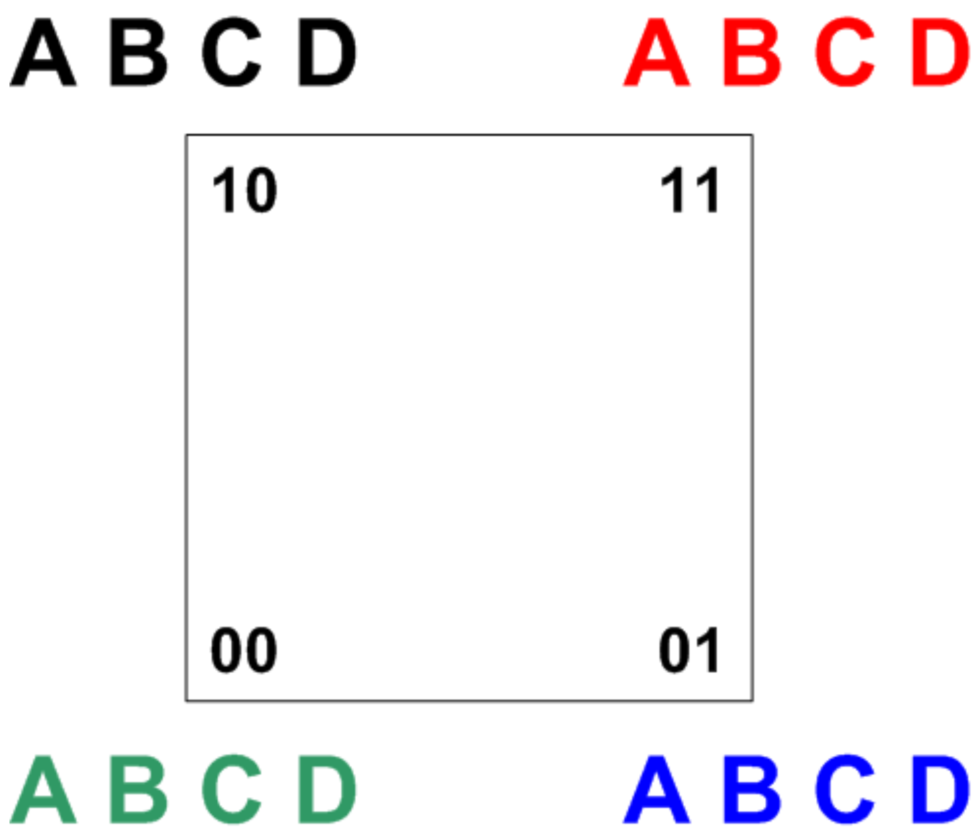
- In each of $\lceil \log p \rceil$ phases all nodes exchange ½ their accumulated data with the others
- Only P/2 messages are sent at any one time

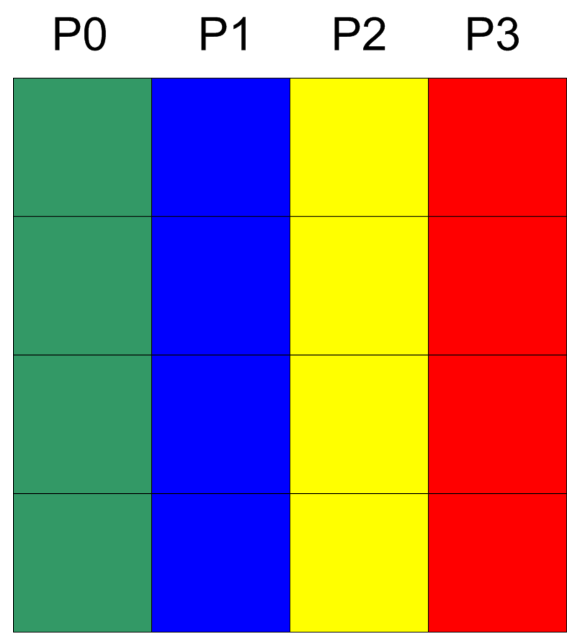
```
D = 1
while (D < p)
   Exchange & accumulate data with rank ⊗ D
   Left shift D by 1
end while
```

- Optimal running time for short messages

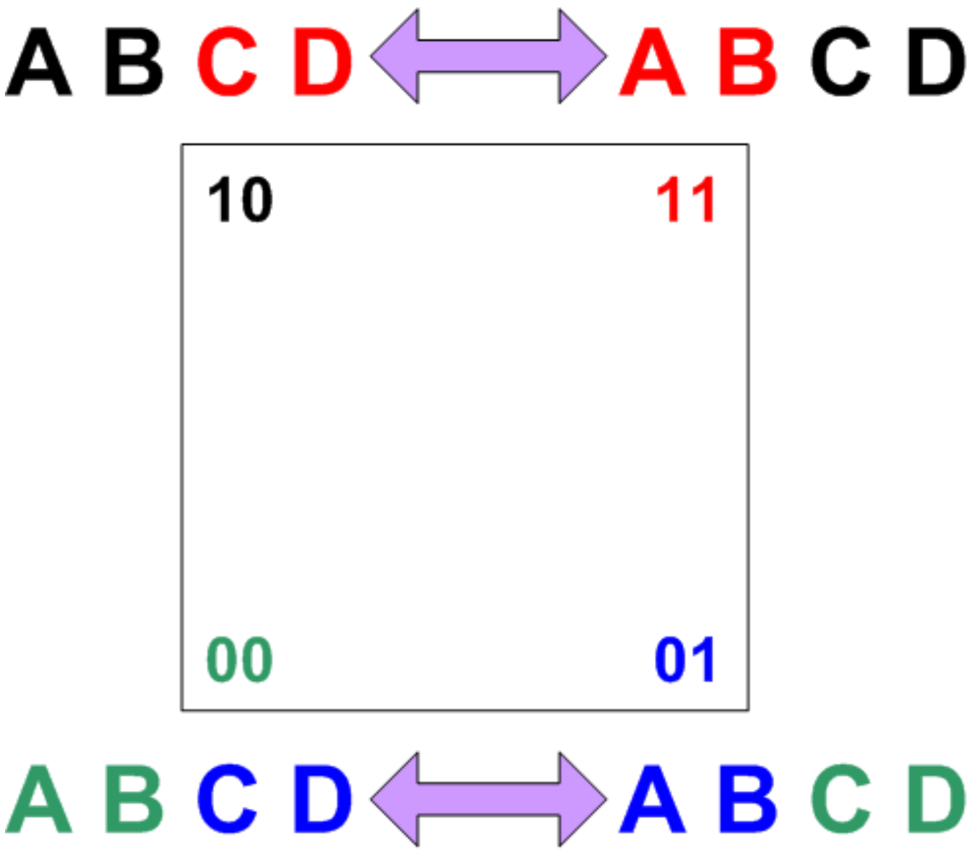
$$\lceil \lg P \rceil \alpha + nP\beta \approx \lceil \lg P \rceil \alpha$$

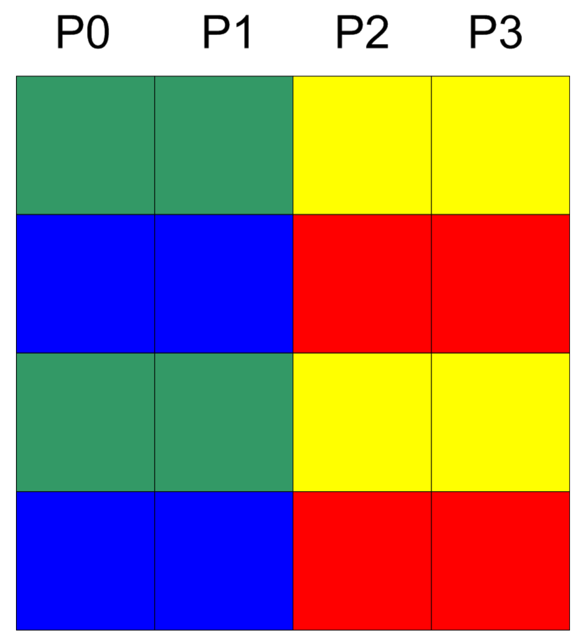
Flow of information



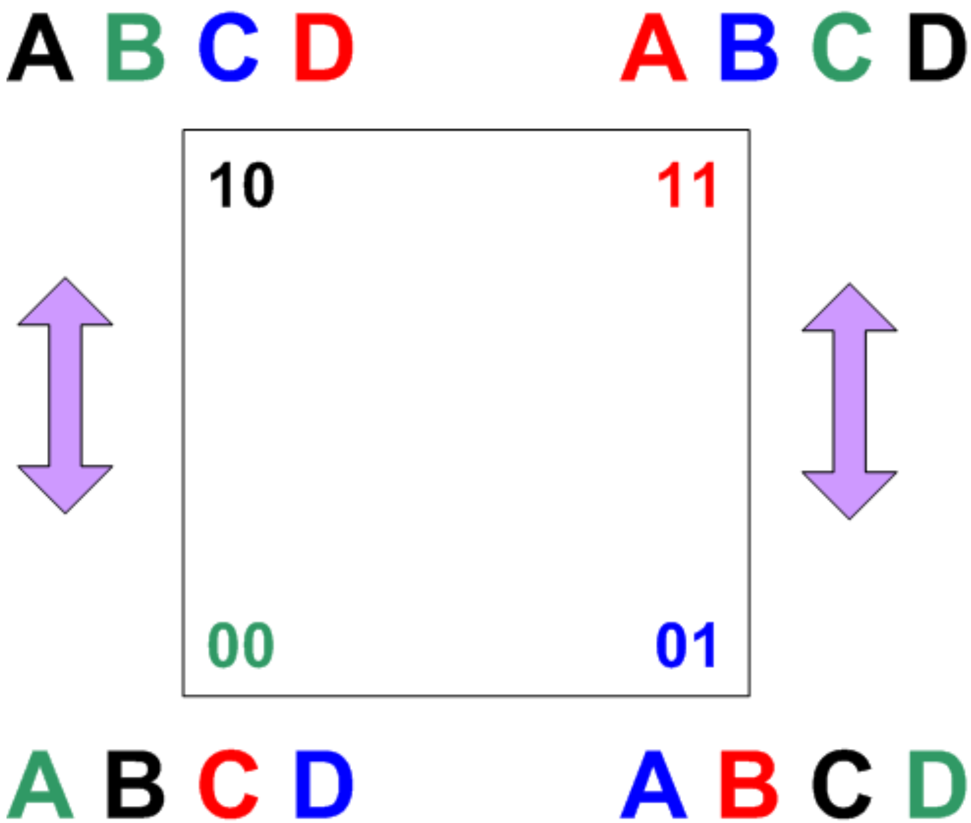


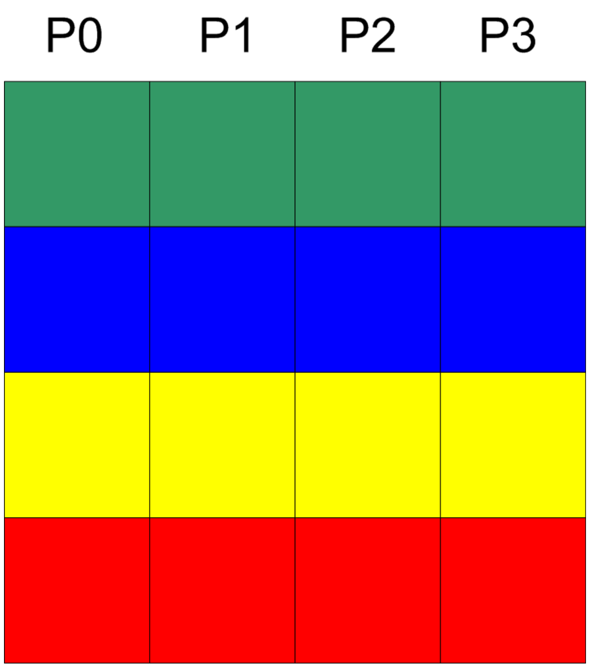
Flow of information





Flow of information





Summarizing all to all

- Short messages $\lceil \lg P \rceil \alpha$

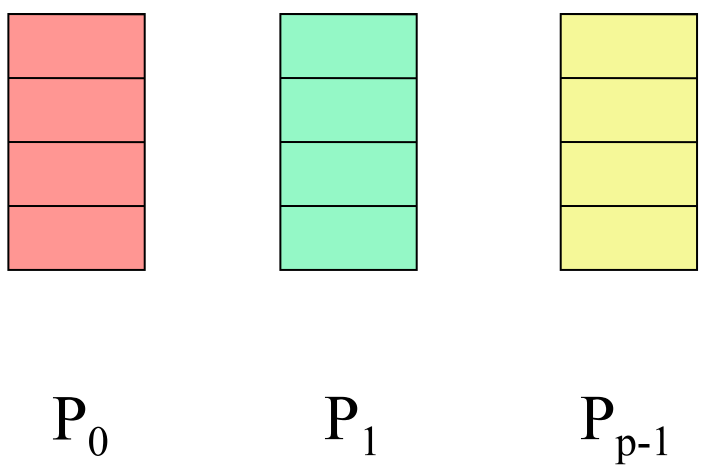
- Long messages $\frac{P-1}{P} n\beta$

Details of the algorithms

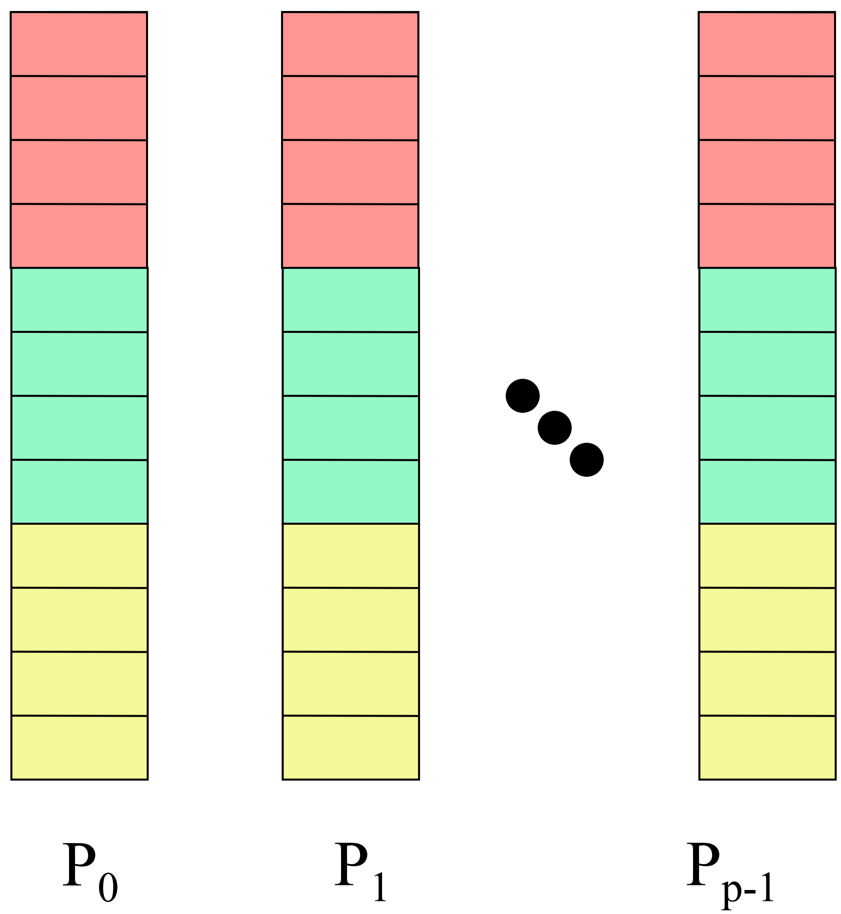
- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast

AllGather

- Equivalent to a gather followed by a broadcast
- All processors accumulate a chunk of data from all the others



AllGather



Allgather

- Use the all to all recursive doubling algorithm
- For P a power of two, running time is

$$\lceil \lg P \rceil \alpha + \frac{p-1}{p} n\beta$$

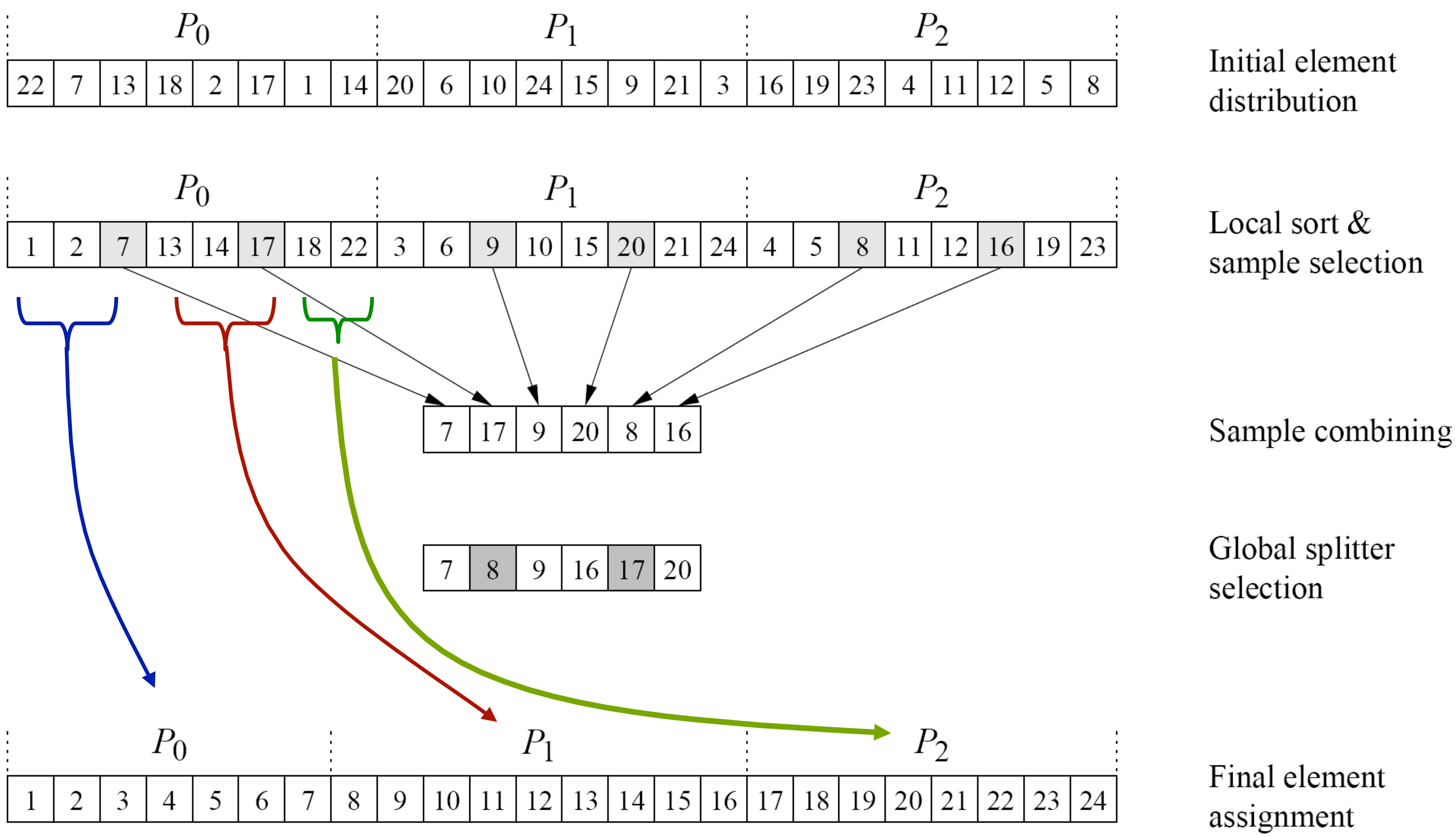
- Also used in Allgatherv

"Vector" variants

- Generalize all-to-all, gather, etc.
- Processes supply varying length datum
- Vector all-to-all

```
MPI_Alltoallv (
    void *sendbuf, int sendcounts[], int sDispl [],
    MPI_Datatype sendtype,
    void* recvbuf, int recvcnts[], int rDispl[],
    MPI_Datatype recvtype, MPI_Comm comm )
```

Alltoallv used in sample sort



Introduction to Parallel Computing, 2nd Ed,, A.Grama, A.l Gupta, G. Karypis, and V. Kumar, Addison-Wesley, 2003.

Details of the algorithms

- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast

Revisiting Broadcast

- P may not be a power of 2
- We use a binomial tree algorithm
- We'll use the hypercube algorithm to illustrate the special case of $P=2^k$
- Hypercube algorithm is efficient for short messages
- We use a different algorithm for long messages

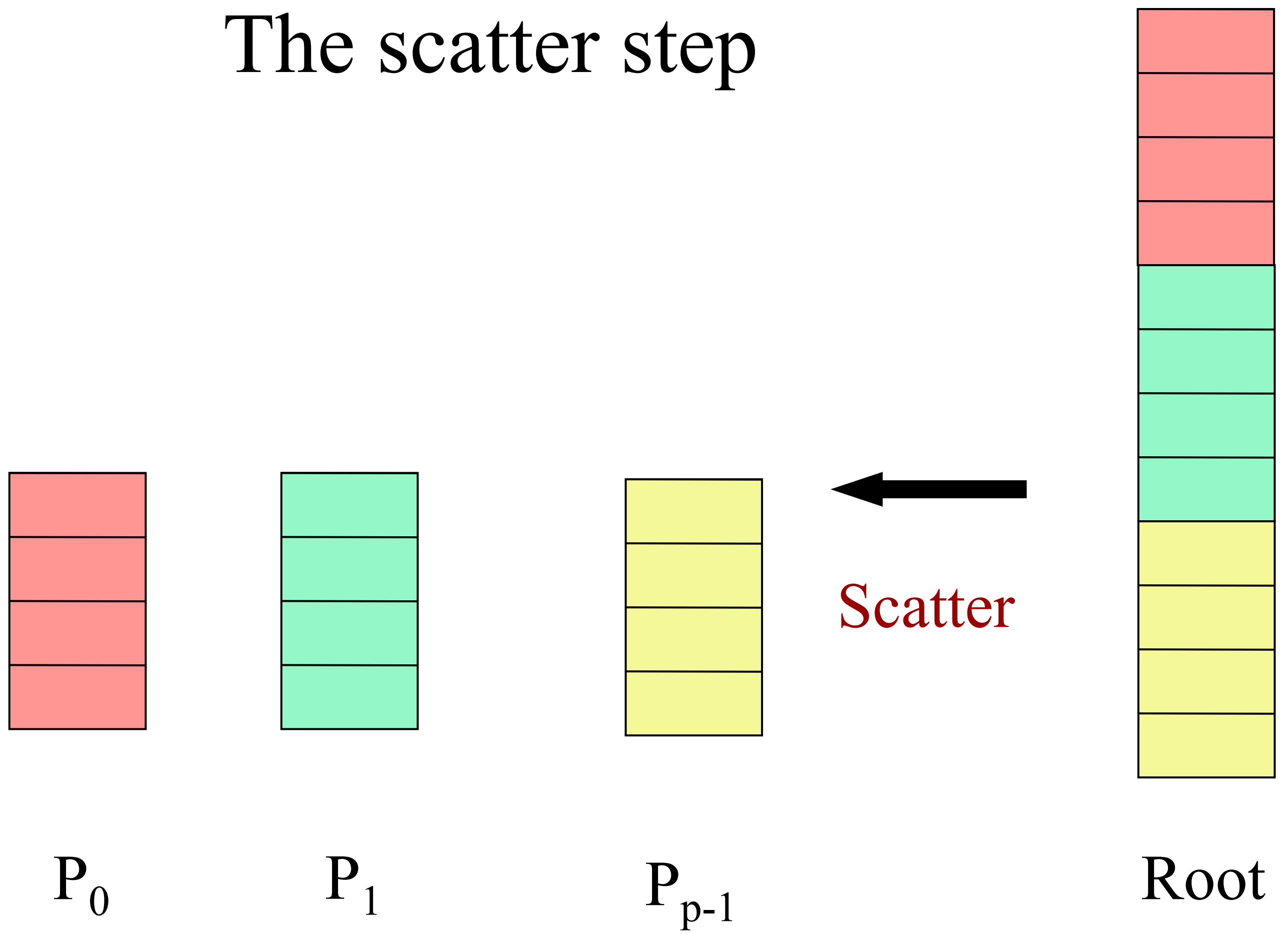
Strategy for long messages

- Based van de Geijn's strategy
- Scatter the data
  - Divide the data to be broadcast into pieces, and fill the machine with the pieces
- Do an Allgather
  - Now that everyone has a part of the entire result, collect on all processors
- Faster than MST algorithm for long messages

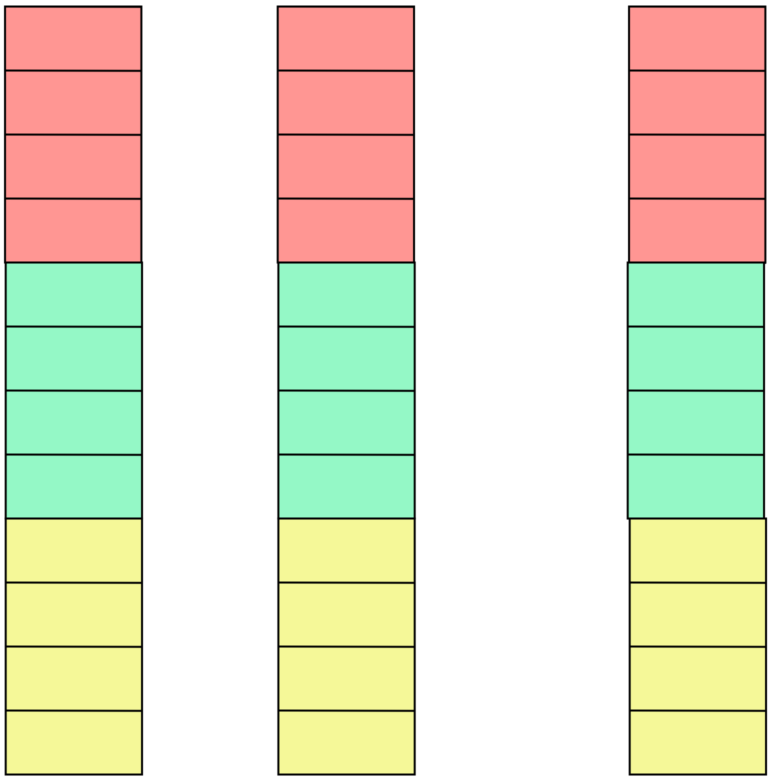
$$2\frac{p-1}{p}n\beta << \lceil \lg p \rceil n\beta$$

Algorithm for long messages

The scatter step



Algorithm for long messages



P_0 P_1 P_{p-1}

AllGather step

References

- S. L. Johnsson and C.T.Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. on Computers*, 38(9): 1249-1268 (1989).
- R. M. Karp et al., "Optimal broadcast and summation in the LogP model," Proc. 5th Annual ACM Symp. Par. Algor. and Arch., pp. 142 - 153 (1993)
- A. Bar-Noy and S. Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems," *Math. Sys. Theory*, 27:431-452 (1994).