Lecture 7: Network Topologies

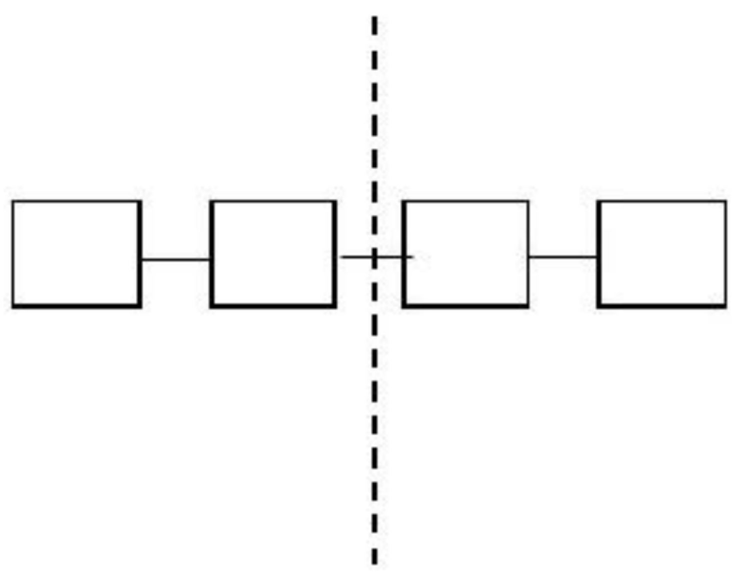
CMSE 822: Parallel Computing

Prof. Sean M. Couch

Bisection width (or bandwidth)

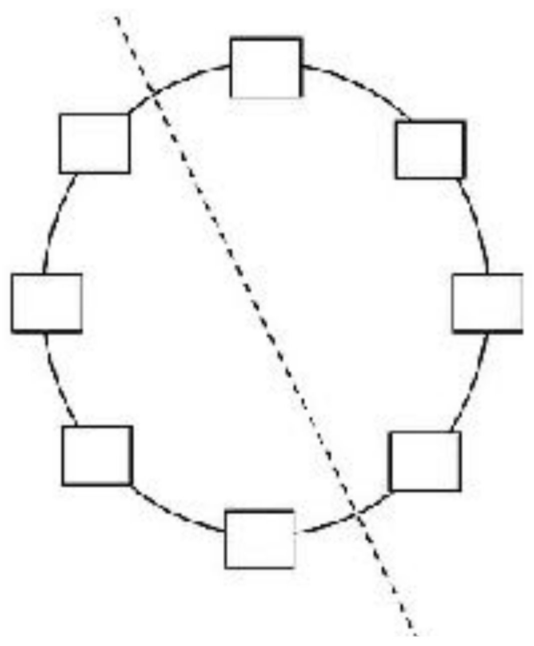
- Smallest number of links between two equal partitions of a network

Linear array



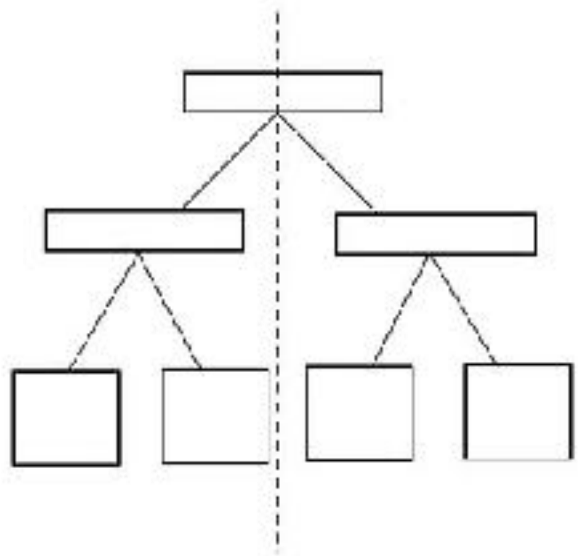
w=1

Ring



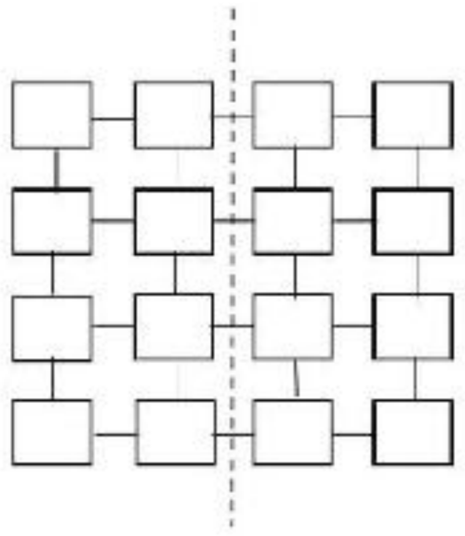
w=2

Tree



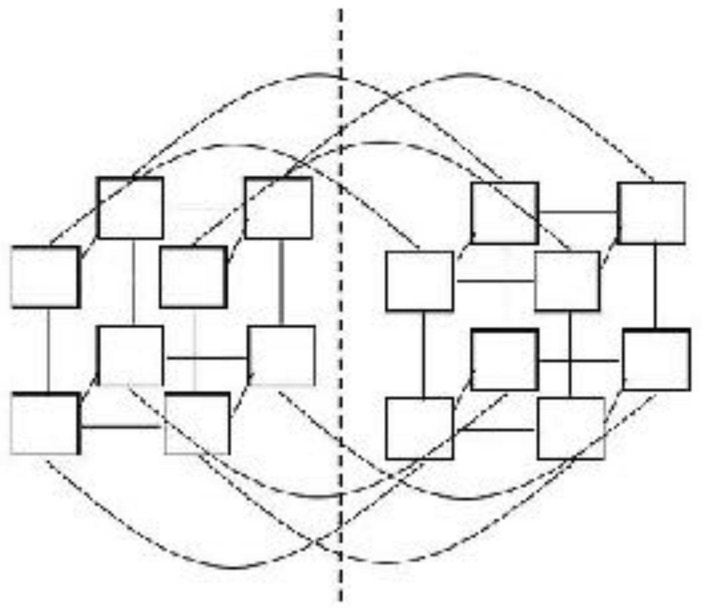
w=1

Mesh



w=$\sqrt{n}$

Hypercube



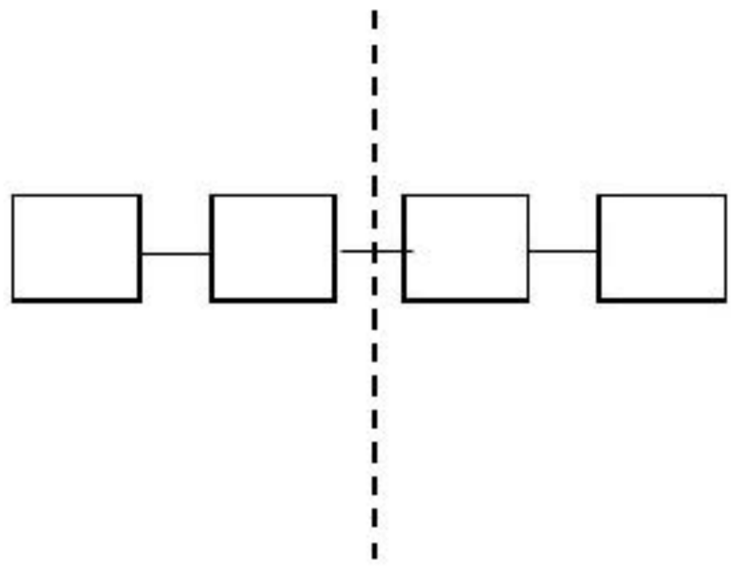
w=$n/2$

Diameter of network

$$d(G) = \max_{i,j} |\text{shortest path between } i \text{ and } j|.$$

- The *longest* shortest distance between two nodes

Linear array



$d = n - 1$

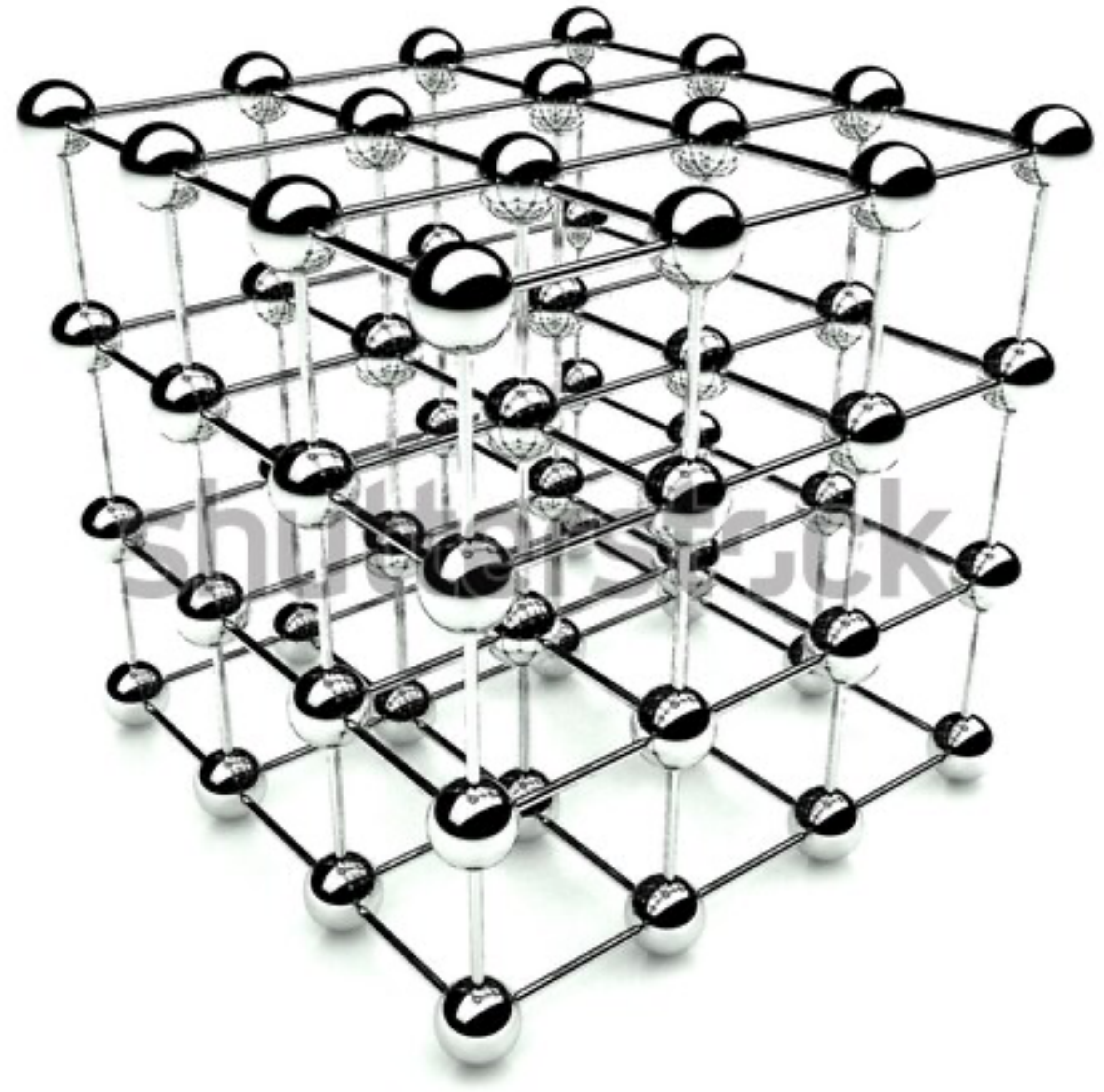
Group exercise

- What is the diameter of a 3D cube of $n \times n \times n$ processors? What is the bisection width? How does that change if you add wraparound torus connections?

Group exercise

- What is the diameter of a 3D cube of $n \times n \times n$ processors? What is the bisection width? How does that change if you add wraparound torus connections?

- A cube has $\sqrt[3]{P}$ processors per side, so the corners are $3\sqrt[3]{P}$ apart. The bisection is $n \times n$ (or $P^{2/3}$). Adding torus connections, the diameter is $\frac{3}{2}\sqrt[3]{P}$ and the bisection width is $(\sqrt[3]{P}+1)^2$.

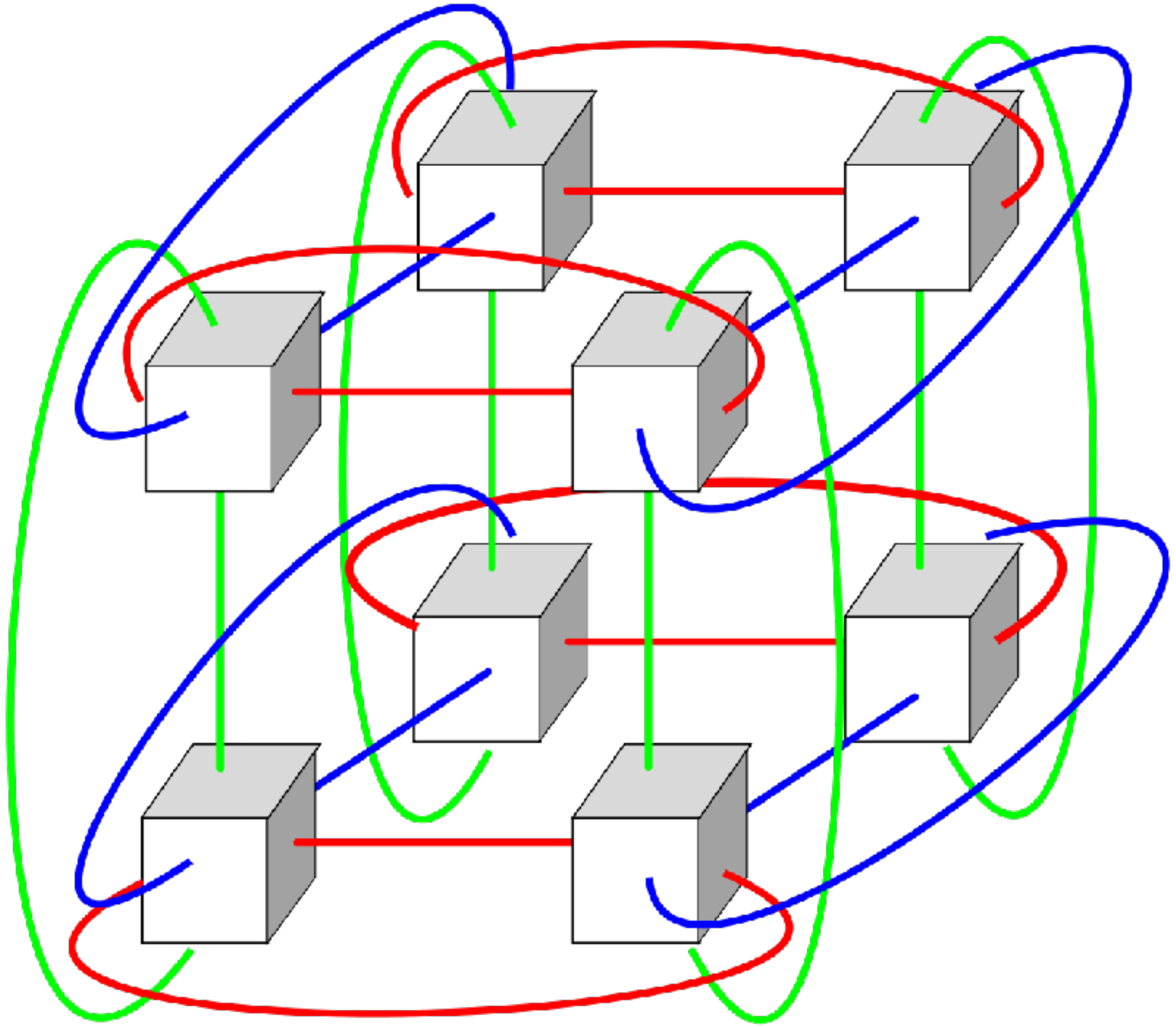


3D torus

Table 4.2. Topological Parameters of Selected Interconnection Networks

Network name(s)	No. of nodes	Network diameter	Bisection width	Node degree	Local links?
1D mesh (linear array)	k	$k-1$	1	2	Yes
1D tours (ring, loop)	k	$k/2$	2	2	Yes
2D mesh	k^2	$2k-2$	k	4	Yes
2D torus (k-ary 2-cube)	k^2	k	$2k$	4	Yes[1]
3D mesh	k^3	$3k-3$	k^2	6	Yes
3D torus (k-ary 3-cube)	k^3	$3k/2$	$2k^2$	6	Yes[1]
Pyramid	$(4k^2-1)/3$	$2\log_2 k$	$2k$	9	No
Binary tree	2^l-1	$2l-2$	1	3	No
4-ary hypertree	$2^l(2^{l+1}-1)$	$2l$	2^{l+1}	6	No
Butterfly	$2^l(l+1)$	$2l$	2^l	4	No
Hypercube	2^l	l	2^{l-1}	l	No
Cube-connected cycles	$2^l l$	$2l$	2^{l-1}	3	No
Shuffle–exchange	2^l	$2l-1$	$\geq 2^{l-1}/l$	4 unidir.	No
De Bruijn	2^l	l	$2^l/l$	4 unidir.	No

[1]With folded layout.



Group exercise

- Your parallel computer has its processors organized in a 2D grid. The chip manufacturer comes out with a new chip with same clock speed that is dual core instead of single core, and that will fit in the existing sockets. Critique the following argument: ‘the amount of work per second that can be done (that does not involve communication) doubles; since the network stays the same, the bisection bandwidth also stays the same, so I can reasonably expect my new machine to become twice as fast.’

Group exercise

- Your parallel computer has its processors organized in a 2D grid. The chip manufacturer comes out with a new chip with same clock speed that is dual core instead of single core, and that will fit in the existing sockets. Critique the following argument: ‘the amount of work per second that can be done (that does not involve communication) doubles; since the network stays the same, the bisection bandwidth also stays the same, so I can reasonably expect my new machine to become twice as fast.’
- The existing bandwdith through each wire will now be shared by two cores, so the per core bandwdith is in fact halved. What’s more, your new configuration has a different graph, so you need to recompute the bisection bandwidth.

Group exercise

```
for (s=2; s<2*n; s*=2)
  for (i=0; i<n-s/2; i+=s)
    x[i] += x[i+s/2]
```

- Consider the parallel summing example and give the execution time of a parallel implementation on a hypercube. Show that the theoretical speedup from the example is attained (up to a factor) for the implementation on a hypercube.

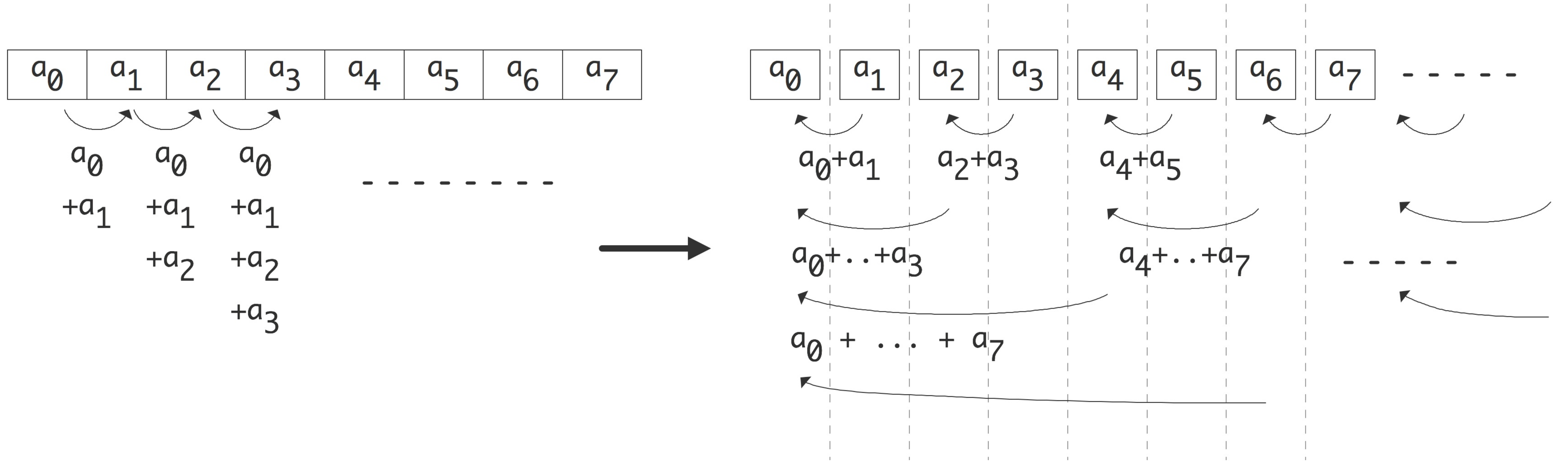


Figure 2.2: Parallelization of a vector reduction

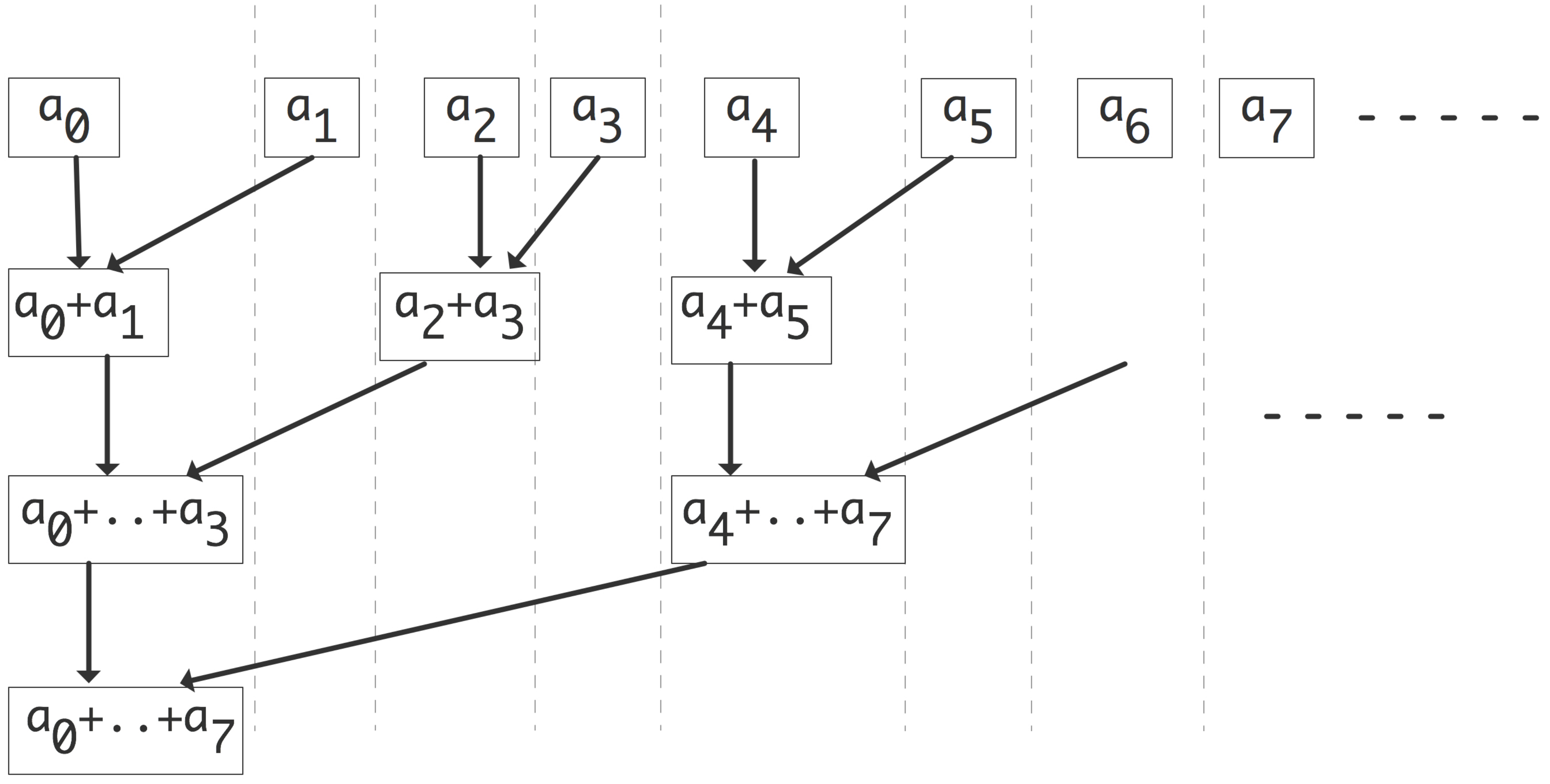
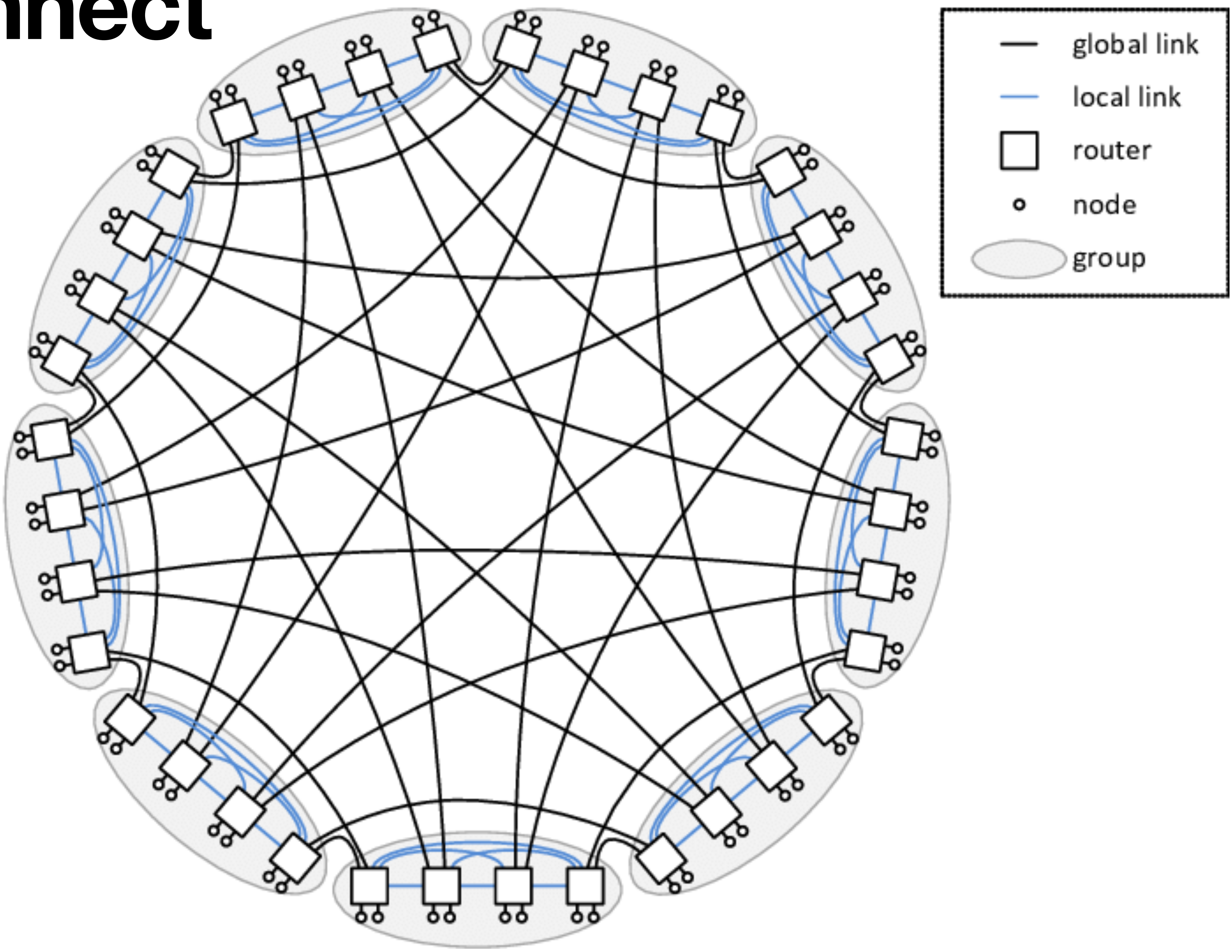
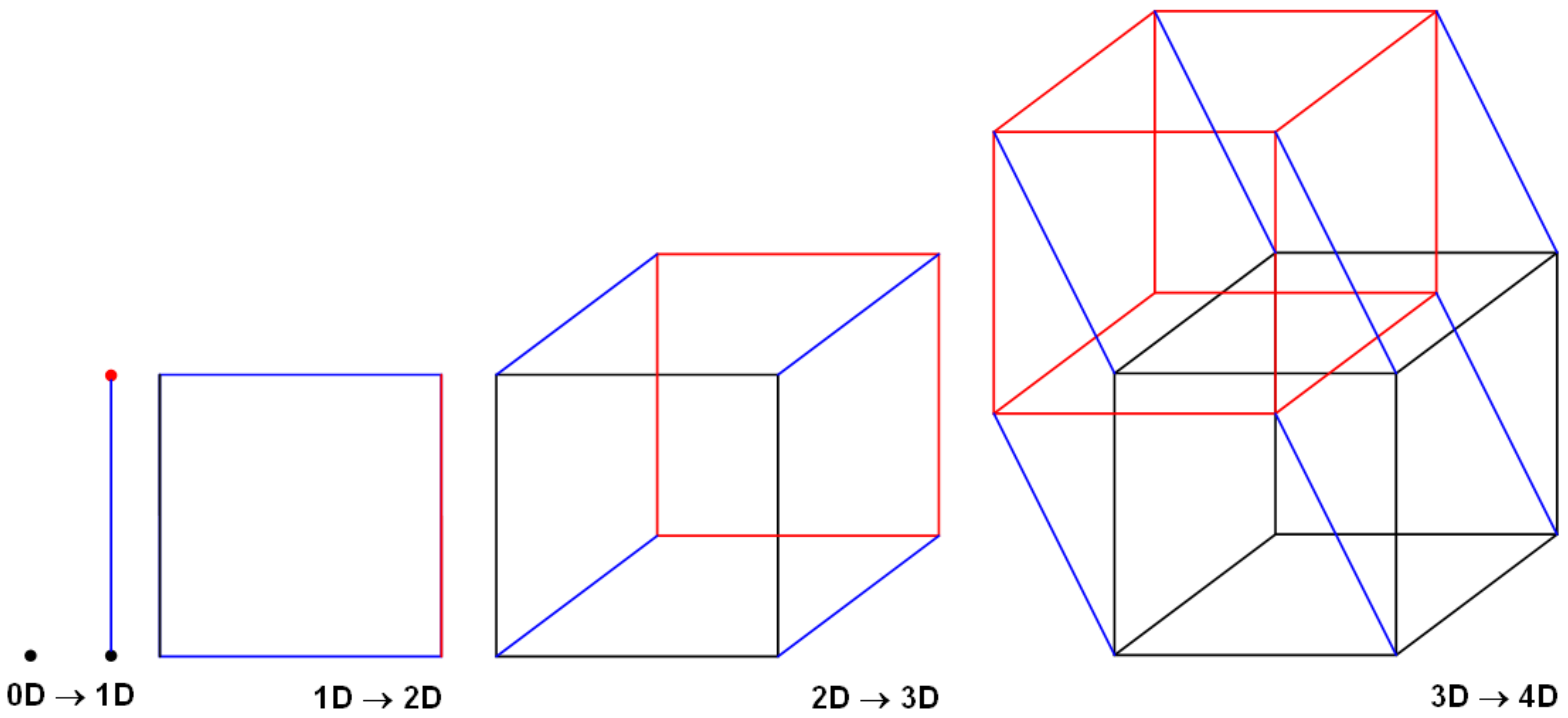


Figure 2.3: Communication structure of a parallel vector reduction

Dragonfly interconnect

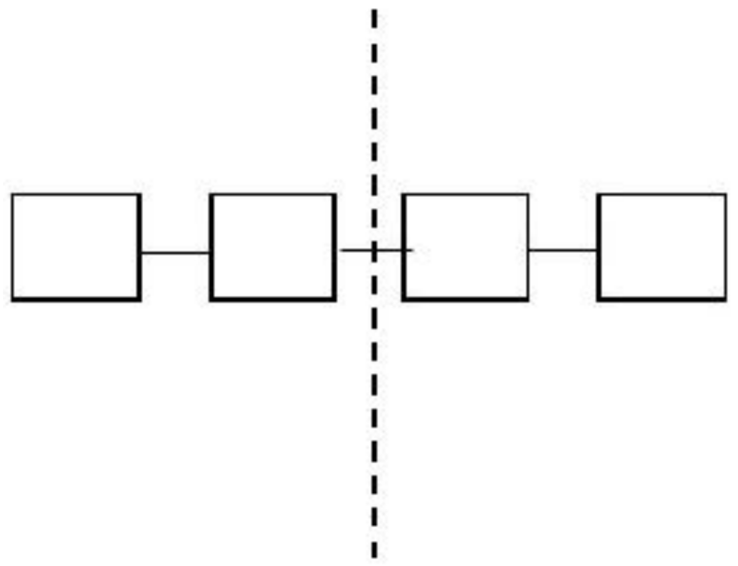
- Messages travel *at most* one long, global hop
- Reduced cost
- Risk of contention, but smart adaptive routing algorithms give nearly ideal performance



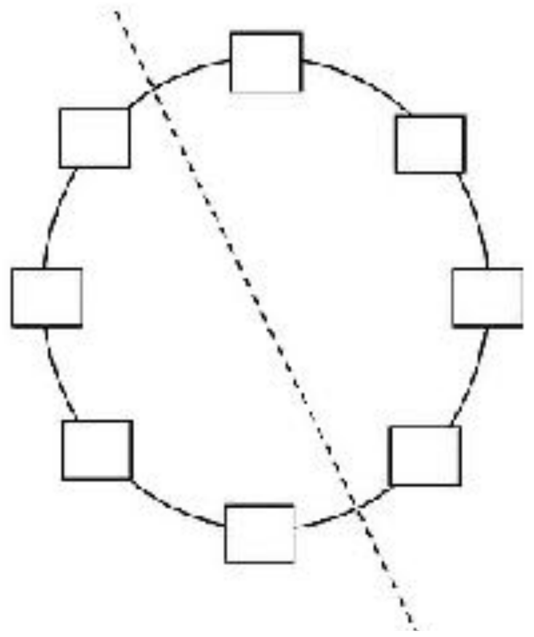

0D → 1D
1D → 2D
2D → 3D
3D → 4D

Exercise 2.30

Linear array



Ring



Exercise 2.30. With the limited connections of a linear array, you may have to be clever about how to program parallel algorithms. For instance, consider a 'broadcast' operation: processor 0 has a data item that needs to be sent to every other processor.

We make the following simplifying assumptions:

- a processor can send any number of messages simultaneously,
- but a wire can can carry only one message at a time; however,
- communication between any two processors takes unit time, regardless of the number of processors in between them.

In a fully connected network or a star network you can simply write

for $i = 1 \ldots N - 1$:

send the message to processor i

With the assumption that a processor can send multiple messages, this means that the operation is done in one step.

Now consider a linear array. Show that, even with this unlimited capacity for sending, the above algorithm runs into trouble because of congestion.

Find a better way to organize the send operations. Hint: pretend that your processors are connected as a binary tree. Assume that there are $N = 2^n - 1$ processors. Show that the broadcast can be done in $\log N$ stages, and that processors only need to be able to send a single message simultaneously.