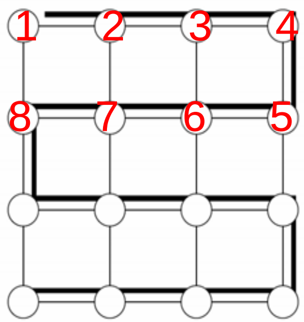
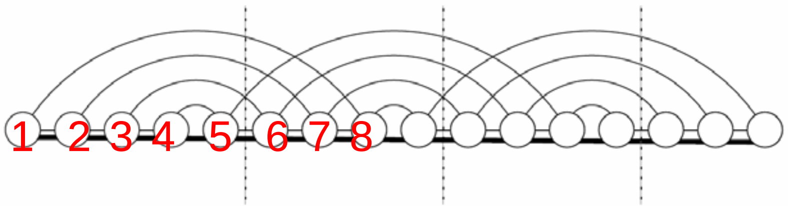
Embedding a Mesh in a Linear Array

- Embedding linear array in mesh:



congestion: 1; dilation: 1

- Embedding a Mesh in a Linear Array by using the inverse mapping:



congestion: 5

in general: $\sqrt{p} + 1$

Cost – Performance – Tradeoff: Comparison Fat-Mesh / Hypercube - p nodes identical costs: proportional to the number of wires

	fat-mesh	**hypercube**
costs	k (=2p * f)	k (= p/2 * log p)
costs per channel	f = (log p)/4	1
average distance of two nodes	$l_{av} = \sqrt{p}/2$	$l_{av} = ½ * \log p$
time for sending message of size m between two random nodes (cut-through routing)	$t_s + t_h \cdot l_{av} + t_w/f \cdot m$	$t_s + t_h \cdot l_{av} + t_w \cdot m$
per word transfer time	$t_w / f = 4\ t_w/(\log p)$	t_w
average communication latency	$t_s + t_h \sqrt{p}/2 + 4\ t_w\ m/(\log p)$	$t_s + t_h \cdot (\log p)/2 + t_w\ m$

⇨ for p>16 and m sufficiently large, the fat-mesh is better

Note: cut-through routing and light load conditions!

Cost – Performance – Tradeoff: Comparison Fat-Mesh / Hypercube - p nodes identical costs: bisection width

	fat-mesh	**hypercube**
costs	k (= $2\sqrt{p}$ * f)	k (= p/2)
costs per channel	f = $\sqrt{p}/4$	1
average distance of two nodes	$l_{av} = \sqrt{p}/2$	$l_{av} = ½ \cdot \log p$
time for sending message of size m between two random nodes (cut-through routing)	$t_s + t_h \cdot l_{av} + t_w / f \cdot m$	$t_s + t_h \cdot l_{av} + t_w \cdot m$
per word transfer time	$t_w / f = 4 \cdot t_w / \sqrt{p}$	t_w
average communication latency	$t_s + t_h \sqrt{p}/2 + 4\ t_w\ m / \sqrt{p}$	$t_s + t_h \cdot (\log p)/2 + t_w\ m$

⇨ again: for p>16 and m sufficiently large, the fat-mesh is better

even when the network is heavily loaded, the performance is similar to that of the hypercube at the same cost