Embedding

1 Definition

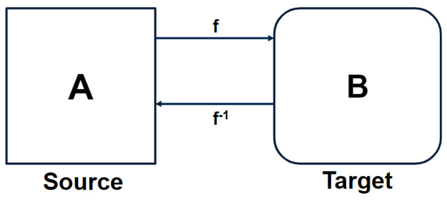


Figure 1: Embedding Definition

Assume we have a algorithm designed for *Source* network, but the system we have is *Target* network. Now we want to do the transfer from *Source* to *Target*, so that the algorithm developed on one model could solve problems on another. This process is called **embedding**. In this case, source and target can be modeled as graphs.

Note that our ability to run an algorithm designed for model A on a different model A without loss of efficiency **does not mean that our algorithm is efficient on B**. It merely states that efficiency is preserved when we move across the models, i.e. the algorithm will run as efficiently on B as it did on A.

2 Embedding Performance Metrics

Assume we do the embedding a source graph $G(V, E)$ into a target graph $G'(V', E')$. The performance metrics include:

1. **Congestion**: Maximum number of edges from E that are mapped onto a single edge in E'. Or in other words, it refers to the maximum number of paths (or communications) that go through a single resource (like a processor).

   High congestion means that a particular resource is being heavily used compared to others, which can lead to bottlenecks and reduce the overall performance of the parallel system. The goal in optimizing embeddings is to minimize congestion, ensuring a more balanced use of resources and smoother communication paths.

2. **Dilation**: Maximum number of edges in E' that any edge in E is mapped to. In other words, it refers to the longest communication path that needs to be taken between any two tasks in the mapping. It measures the distance messages must

travel in the network, which can be a significant factor in the communication overhead of parallel applications.

A lower dilation means that tasks that need to communicate are closer together, leading to faster communication and better overall performance. In embedding and mapping strategies, the objective is often to minimize dilation to reduce the latency and bandwidth usage associated with inter-process or inter-node communication. **Communication slows down by a factor of Congestion $\times$ Dilation.**

3. **Load**: maximum number of nodes in V that map to a single node in V'. **Computation slows down by a factor of load.** We also define **mapping** as Load = Congestion = Dilation = 1.

4. **Expansion**: The ratio of the number of nodes V' to that in V. **If expansion is greater than one, it indicates waste of resources.**

3 Mapping

There are some characteristics of mapping:

1. Mapping only needs to specify node mapping. Edge mapping is implied.
2. Parallel run-time on the target topology is the same as that on the source topology.
3. Efficiency is preserved when execution of the algorithm is shifted from source topology to target topology.
4. It is possible there can be a different parallel algorithm with better efficiency directly designed for the target topology.
5. This concern is eliminated if the parallel algorithm designed for the source topology is optimally efficient.

4 Network Topology

4.1 Linear Arrays

4.1.1 Examples

The linear array structures are shown below:

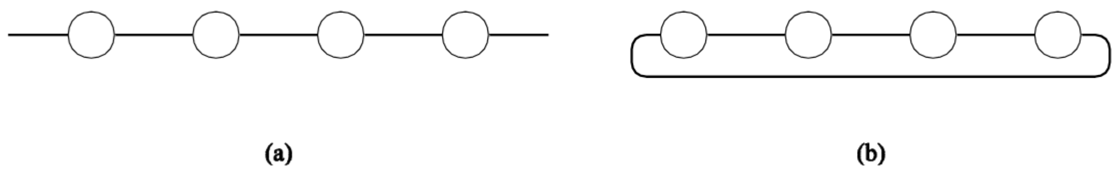


Figure 2: Linear arrays: (a) with no wraparound links; (b) with wraparound link.

Ring:

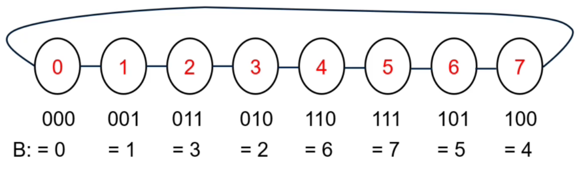
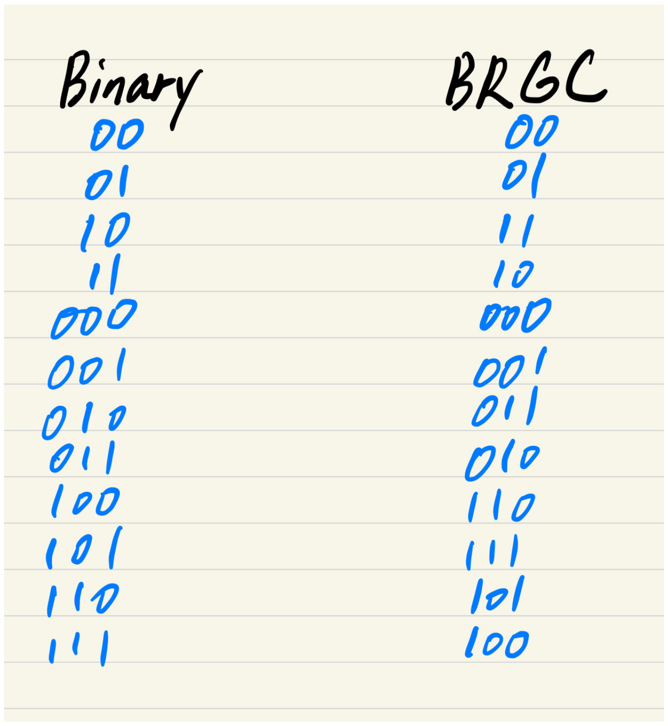


Figure 3: Ring.

4.1.2 Binary Reflected Gray Code

In order to make the change each time only changes 1 digit in the binary code, we introduce the **binary reflected gray code (BRGC)**:



Binary	BRGC
00	00
01	01
10	11
11	10
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Figure 4: BRGC.

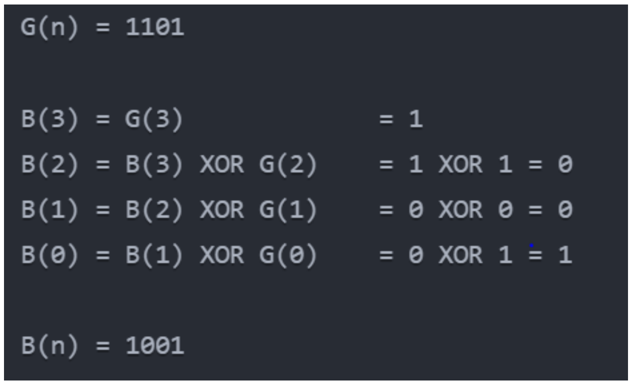
The convention rule is shown below:

1. Binary code to gray code:

$$G(n) = B(n)\text{XOR}(B(n) >> 1) \tag{1}$$

   Here, $B(n) >> 1$ means that the binary code is right shifting one position. For example, 1101 to 0110. For XOR operation, it returns false only if two values are the same.

2. Gray code to binary code:



```
G(n) = 1101

B(3) = G(3)              = 1
B(2) = B(3) XOR G(2)     = 1 XOR 1 = 0
B(1) = B(2) XOR G(1)     = 0 XOR 0 = 0
B(0) = B(1) XOR G(0)     = 0 XOR 1 = 1

B(n) = 1001
```

Figure 5: Gray code to binary code.

4.1.3 Embedding Ring to Hypercube

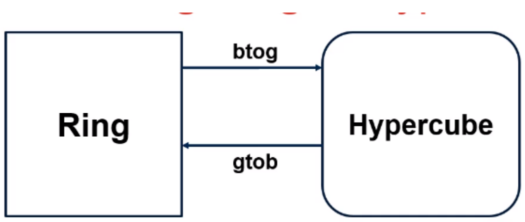


Figure 6: Ring Embedding

Here, *btog* means binary to gray, and *gtob* means gray to binary. Assume the processor rank is r, then:

1. Ring rank: $gtob(r)$
2. Left neighbor in hypercube: $btog[(gtob(r) - 1 + p) \bmod p]$
3. Right neighbor in hypercube: $btog[(gtob(r) + 1) \bmod p]$

4.2 2D and 3D Meshes

4.2.1 Examples

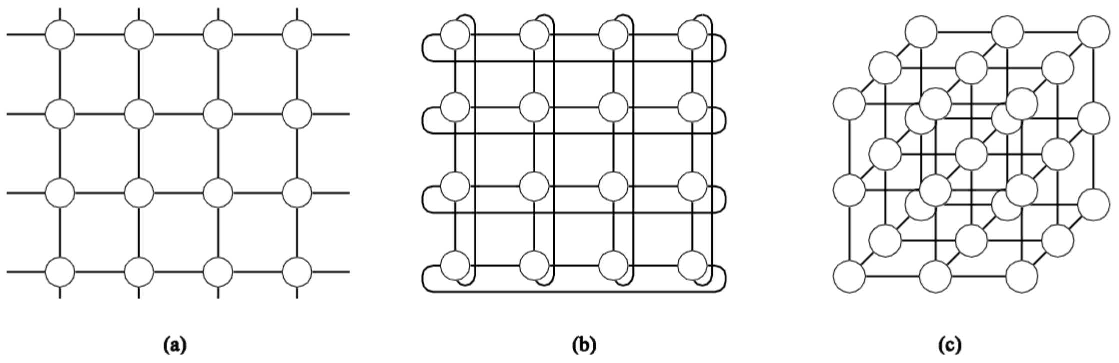


Figure 7: 2D, 3D meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

4.2.2 Embedding a Mesh into Hypercube

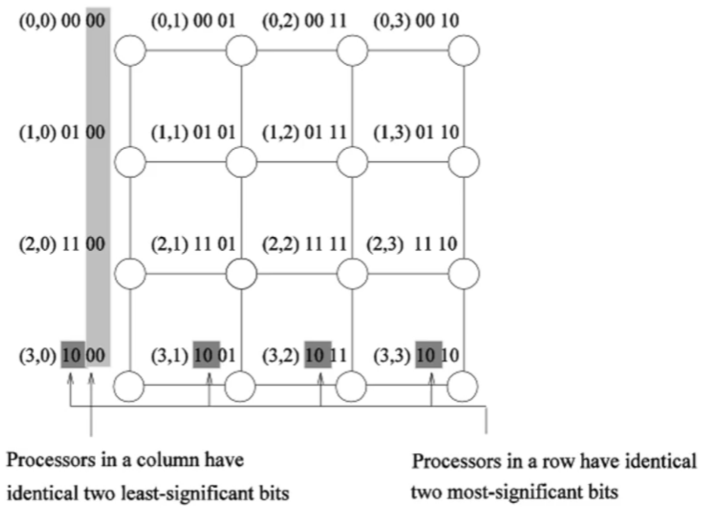


Figure 8: Mesh Embedding.

A $2^r \times 2^s$ wraparound mesh can be mapped to a 2^{r+s} node hypercube by mapping node (i, j) of the mesh onto node $btog(i)||btog(j)$ of the hypercube, where $||$ denotes concentration of the two gray nodes. Some detailed relationship is shown below:

1. Rank in hypercube: bit string of $(r + s)$ bits, or $x||y$, where x = first r bits, and y = last s bits.
2. Rank in mesh: $(gtob(x), gtob(y))$
3. East neighbour in hypercube: $btog[(gtob(r) + 1) \bmod 2^r] \,||\, y$

4. West neighbour in hypercube: $btog[(gtob(r) - 1 + 2^r) \bmod 2^r] \,||\, y$
5. North neighbour in hypercube: $x \,||\, (gtob(r) - 1 + 2^s) \bmod 2^s$
6. South neighbour in hypercube: $x \,||\, (gtob(r) + 1) \bmod 2^s$

Suppose we want to embed (5,2,3,1,7) into a $8 \times 4 \times 16 \times 2 \times 8$, therefore we have:

$$(5, 2, 3, 1, 7) \to 111, 11, 0010, 1, 100 \tag{2}$$

Concentrate all the gray code and transfer it to decimal, we can get the rank of processor.

Another example of 2×4 mesh to 2^3 3D hypercube:

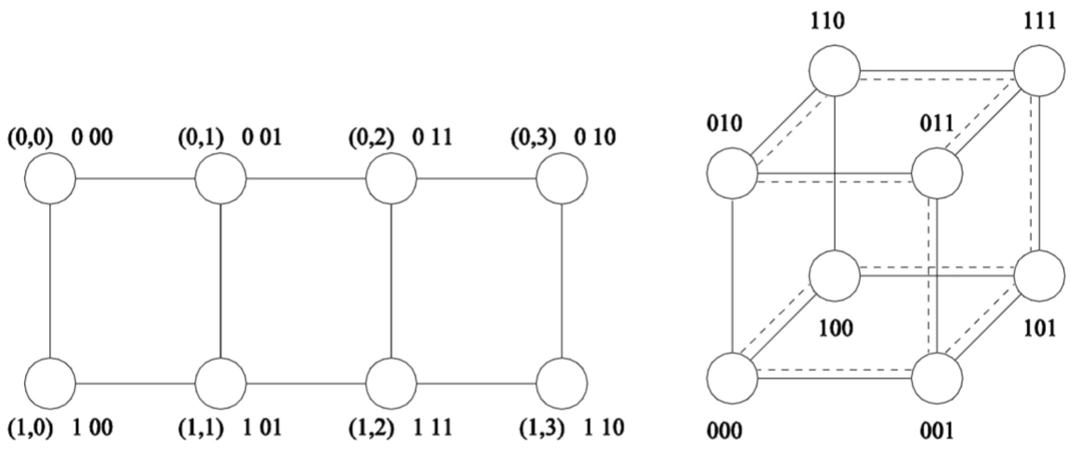


Figure 9: 2x4 mesh embedding

4.3 Binary Tree

Suppose we have a binary tree:

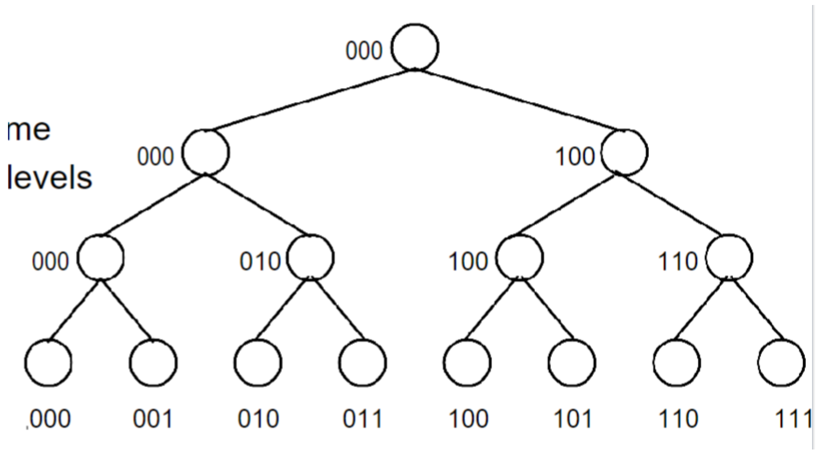


Figure 10: Binary Tree

with the following features:

1. Total number of processors as p, p is a power of 2.
2. The computation process is one level of tree at a time
3. The communication process is between consecutive levels
4. The tree has $\log p + 1$ levels, with root at level 0
5. Level i has 2^i nodes, numbered 0 to $2^i - 1$.

In tree, (i, j) is connected to $(i+1, 2j)$ and $(i+1, 2j+1)$, then in hypercube is:

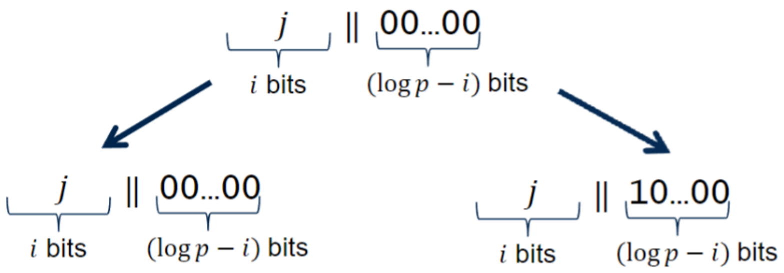

$\underbrace{j}_{i \text{ bits}} \,\|\, \underbrace{00...00}_{(\log p - i) \text{ bits}}$
$\underbrace{j}_{i \text{ bits}} \,\|\, \underbrace{00...00}_{(\log p - i) \text{ bits}}$
$\underbrace{j}_{i \text{ bits}} \,\|\, \underbrace{10...00}_{(\log p - i) \text{ bits}}$

Figure 11: Gray Code Tree

The rules could be summarized as below:

1. Processor with rank r will participate in levels $(\log p - 1)$ to $(\log p - a)$, where a is the number of trailing zeroes in r.
2. At level k, the parent node changes $(\log p - k)^{th}$ bit to 0.
3. The left child will be the same as parent node.
4. The right child will change $(\log p - k - 1)^{th}$ bit to 1.