2.24

- Idea of the comparison with minimum congestion mapping: If an interconnection network A is mapped to a network B with a congestion *r* but network B is *r* times faster than A, then B is strictly superior than A (fewer links, at least same performance).
- The mapping of a hypercube on a mesh follows the inverse of the mesh on the hypercube. A sub-cube of √p processors is mapped on each row of the mesh (assume a √p*√p mesh). We count the number of hypercube links going from one half of the mesh (on a row) to the other half (see Fig. 2.33). Every node of one half has a link to another node on the other half. We have √p/2 links. The mesh has one link (no wrap-around). The congestion on a mesh without wrap-around is √p/2 and with wrap-around √p/4 (since we have 2 links connecting each half).
- We need to check the ratio √p/2 (or √p/4) to compare the hypercube with the mesh. √1024/2=16, √1024/4=8. The mesh is 25/2=12.5 times faster than the hypercube so a wrap-around mesh is strictly better (at least 8 times faster), not the mesh without wrap-around.

3.11

- 2 ways to see it:
  - Either count directly with the help of slide 24 lecture 5: tasks for the first loop n(n-1)/2 to compute the L[j,k] but also U[k,j] + the "splitting" of the element of the diagonal (n) + the loop on the smaller square matrix (size k at every iteration).

$$2\frac{n(n-1)}{2}+n+\sum_{i=1}^{n-1} i^2 = \sum_{i=1}^{n} i^2$$

  - Or recursively: at a given iteration every element of the sub-matrix of size k is touched, hence k^2 tasks, and you add the count for the previous iteration, and you have $t(m)=t(m-1)+m^2$, or the sum of squares directly.

3.12 & 3.13

- 3.12) Maximum degree of concurrency is given by
  - Either the first loop: 2(m-1) tasks in parallel (m-1 for L and U),
  - Or the second loop $(m-1)^2$ tasks in parallel (sub-matrix).
  - There is a dependency between the first and the second loop so it is the max(2(m-1), $(m-1)^2$).
- 3.13) Critical path length: Let's check the dependencies. Every element in the diagonal (except the first) needs an update from the second loop of the algorithm (on the sub-matrix) but its coefficients are computed by the first loop. That gives us a sub-path of length 2 between every "split" of the diagonal element to its L and U parts. There are m splits with a sub-path of length 2 in-between. The critical path length is then 2(m-1)+m.