Lecture 9: Emulating the faulty mesh with constant slowdown

4th and 5th November, 2008

9.1 Motivation

In the last lecture we were showing that in a $n \times n$ p-faulty mesh with $p > p_c$ we can achieve an embedding with $O(\log n)$ slowdown with high probability. It is still an open problem if we can do better than that, i. e. if there is an embedding with constant slowdown for $p > p_c$. It can be shown, however, that "holes", regions containing too many faulty nodes, in the mesh of size $\Theta(\log n)$ will happen with high probability, resulting in an $\Theta(\log n)$ dilation.

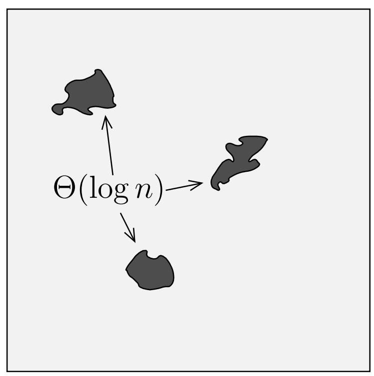

Figure 1: Holes of size $\Theta(\log n)$

Since the dilation has a direct influence on the slowdown we clearly cannot achieve a constant slowdown with a direct embedding. We have to find a smart way to avoid those holes.

9.2 A multi-embedding with constant slowdown

Theorem 9.1 *Any computation on a fault-free $m \times m$ mesh G that takes time T can be emulated by a faulty mesh H with at most $\frac{\log n}{2}$ wost case faults in $O(T + n)$ time steps.*

To prove this theorem we are first going separate areas containing defective nodes in a way, that they are surrounded by enough functional nodes.

Definition 9.2 *A finished box of size k is a $3k \times 3k$ large part of the faulty mesh that is divided into two regions as illustrated in Figure 2:*

- *The core, a square of size $k \times k$ in the middle of the box. This area contains faulty nodes.*
- *The skirt, the area surrounding the core. All the nodes are functional in this region.*

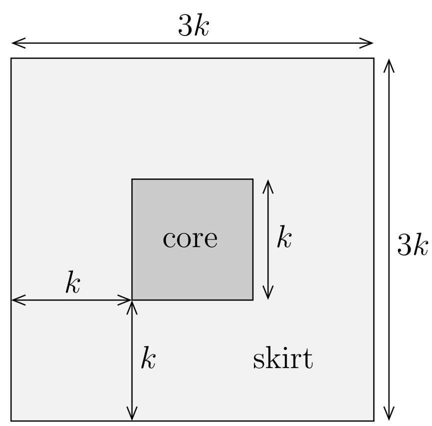

Figure 2: A *finished box* of size k

Note that here we don't care about the borders but assume that the nodes on the left of the mesh are connected to the nodes on the right and same for up and down.

The process of separating the errors into *finished boxes* can be achieved by the following simple algorithm:

1. put each faulty node into a *finished box* of size 1
2. while there exist two *finished boxes* b_1 and b_2 that overlap:
   Merge the two *finished boxes* to one larger *finished box* such that the core of the new *finished box* is the smallest square that covers the core of both b_1 and b_2. This step is illustrated in Figure 3.

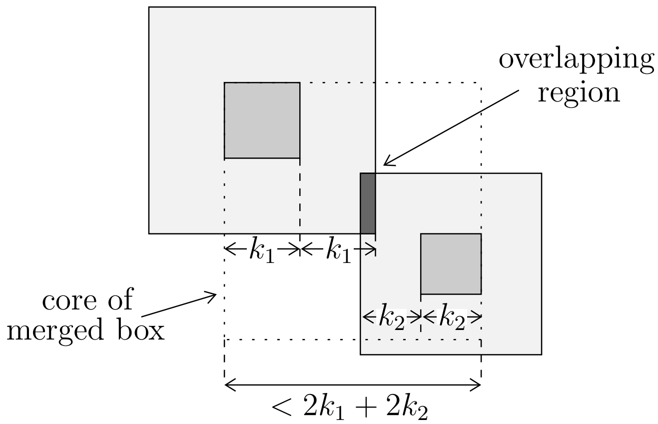

Figure 3: Merging two *finished boxes* with sizes k_1 and k_2

If this algorithm fails then we would have a *finished box* with core size $> \frac{n}{3}$; otherwise it has found a set of non-overlapping finished boxes where errors are only found within the cores of *finished boxes*.

In Lemma 9.1 we will prove that a core of size $\frac{n}{3}$ contains at least $\frac{\log n/3}{2} + 1$ faulty nodes which is more than the $\frac{\log n}{2}$ allowed faults. Therefore the algorithm will always provide us with a set of *finished boxes*.

Lemma 9.1 *With $F(k)$ denoting the minimum number of faults in a core of size k the following equation holds:*

$$F(k) \geq \frac{\log k}{2} + 1$$

Proof by Induction: For $k = 1$, $F(k) = 1$ which satisfies the inequality. If a core has size > 1, then it was created by merging two smaller *finished boxes* with sizes $0 < k_1, k_2 < k$. From Figure 3 we can easily see that

$$\begin{aligned} 2k_1 + 2k_2 &> k \\ k_1 + k_2 &> \frac{k}{2} \\ k_1 + k_2 &\geq \left\lfloor \frac{k}{2} \right\rfloor + 1 \end{aligned} \tag{1}$$

Using the induction hypothesis for the smaller boxes we get

$$\begin{aligned} F(k) &\geq F(k_1) + F(k_2) \\ &\geq \frac{\log k_1}{2} + \frac{\log k_2}{2} + 2 \end{aligned}$$

using Equation 1 and setting $k_1 := \left\lfloor \frac{k}{2} \right\rfloor$ and $k_2 := 1$ in order to make this inequality as small as possible we get

$$\begin{aligned} F(k) &\geq \frac{\log \left\lfloor \frac{k}{2} \right\rfloor}{2} + \frac{\log 1}{2} + 2 \\ &\geq \frac{\log k}{2} + 1 \end{aligned}$$

This concludes the proof for this lemma. ■

Knowing that we can separate all the errors within in the core of *finished boxes* that do not overlap we will start working on the embedding. Since all the nodes outside of *finished boxes* are fault-free, we will just map the nodes in the original network corresponding to this region directly to these nodes.

In the next step, we construct a mapping to emulate a $3k \times 3k$ mesh with a *finished box* of size k without using the core region, such that the slowdown is constant. By applying this mapping to all regions corresponding to *finished boxes* in our faulty mesh, Theorem 9.1 follows immediately.

To do this embedding we will define a new region, the patch, within the $3k \times 3k$ mesh. The patch is a sub-mesh of size $2k \times 2k$ centered in the *finished box*. Note that the patch overlaps with the skirt in a ring-like area around the core. This is illustrated in Figure 4.

The skirt and the patch now have two rings: a border ring, called b-ring, and an internal ring, called i-ring. This is illustrated in Figure 5. The important point here is, that the i-ring in both of the reason lies at least $\Theta(k)$ steps away from the border. We will shortly take advantage of this fact.

Figure 5 also illustrates how we are going to embed the corresponding $3k \times 3k$ subset of the fault-free mesh G within a *finished box*:

- The region corresponding to the skirt is mapped straight forward one-to-one. This doesn't cause any problems, because the skirt is fault-free.

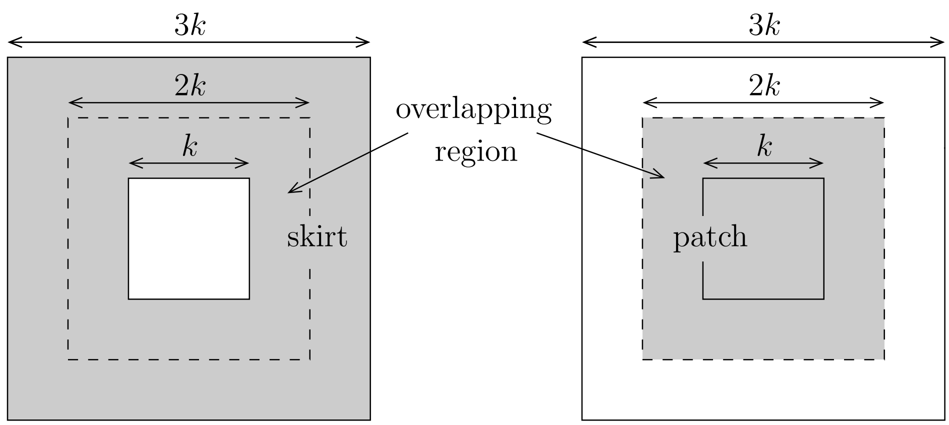

Figure 4: Skirt and patch

- The region corresponding to the patch is mapped to a $\frac{k}{2} \times \frac{k}{2}$ sub-mesh located to the right of the core. Since the patch-region covers $4k^2$ and the new region only $\frac{k^2}{4}$ nodes, each node has a load of 16 plus 1 for the already embedded node of the skirt.

Note that we have embedded all the nodes in the overlapping region twice.

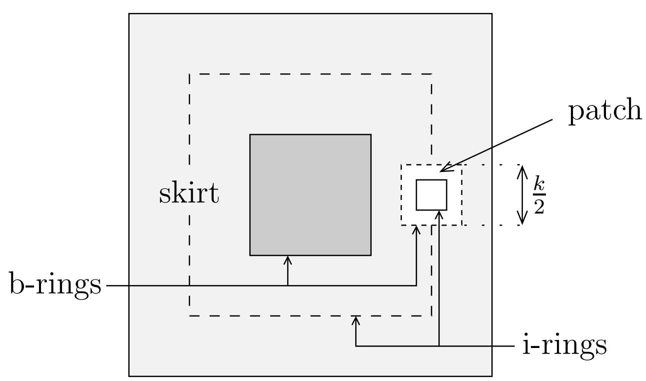

Figure 5: The embedding with i-rings and b-rings

The next step is to connect the i-rings with the b-rings by constructing paths. We only need paths from i-rings to b-rings (and not the other way around).

We will (without proof) use the fact that we can construct paths between those rings with constant congestion and a length of $\Theta(k)$.

Let B_H be the area of the *finished box* we are currently working with and B_G be the $3k \times 3k$ sub-mesh of the non-faulty mesh G that corresponds to this region. To formalize a run of the mesh we will use the following terms:

- *s-pebbles* $\langle v, t \rangle$: the state of node v at time t, $\forall v \in B_G$, $\forall t$.
- *c-pebbles* $[e, t]$: the information sent on edge e, $\forall e \in E(B_G)$, $\forall t$.

Now, for each time step t the each node $v \in B_G$ does the following:

- compute the s-pebble $\langle v, t \rangle$
- compute and send the 4 c-pebbels $[e_1, t], [e_2, t], [e_3, t], [e_4, t]$, one on each outgoing edge

To emulate this process in our faulty mesh we will define "macrosteps" that have to be executed for each $m \in B_H$. A macrostep consists of the following three steps:

1. **computation step**: for each $v \in B_G$ mapped to m, m creates a s-pebble $\langle v, t \rangle$ provided m has computed $\langle v, t-1 \rangle$ and received the c-pebbles $[e, t-1]$ for every edge e incident on v.
2. **communication step**: for every $v \in B_G$ s. t. m created $\langle v, t \rangle$ in this macrostep generate $[e, t]$ for all edges e incident on v. If $[e, t]$ is needed by a neighbor $m' \in B_H$ forward $[e, t]$ to m'.
3. **routing step**: If m lies on an i-ring, then m makes copies and forwards any c-pebble sent to it in this macrostep to the node on the b-ring corresponding to m.
   All nodes (whether or not they lie on an i-ring) forward any c-pebble that has not yet reached its destination.

The time taken for one macro step is as follows:
computation step ≤ 17
communication step $\leq 17 \times 4$
routing step $=$ congestion of the path system $=$ constant
Therefore, the execution of one macrosteps takes constant time only.

To show that using this embedding and executing these macrosteps indeed gives a constant slowdown we will trace back the very last s-pebble $S = \langle v, T \rangle$ that was created by a run of the system. We do that by building a dependency tree, as shown in Figure 6, with S as the root. The children of S are all the resources required to compute S: that is the previous s-pebble from the same node and the four c-pebbles from the neighbors in B_G. We are descending the tree on the path that contains the object that was waited for the longest. Since from any c-pebble there is only one path that leads to a s-pebble, we get a critical path of s-pebbles: $S_T, S_{T-1}, \dots, S_0$.

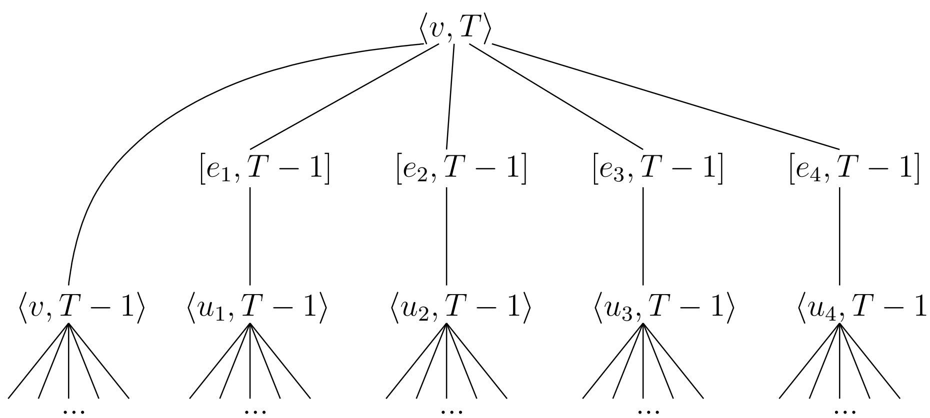

Figure 6: Dependency tree, starting from $\langle v, T \rangle$

Let $\tau(s)$ denote the number of macrosteps that were executed when the s-pebble s was created and l_i the number of macrosteps taken to get s_i from s_{i-1}; the total time taken (counted in makrosteps) can be expressed as

$$T' = \sum_{0 \leq i \leq T} (\tau(s_i) - \tau(s_{i-1})) = \sum_{0 \leq i \leq T} l_i \tag{2}$$

If all l_i along the critical path are 1 then we have only constant slowdown because each macrostep takes only constant time.

For each $i \in \{1, \dots, T\}$ where $l_i > 1$ we are (by construction of the critical path) tracing a c-pebble back from a b-ring to an i-ring. Because the paths between the rings have constant congestion and length $\Theta(k)$ this takes $\Theta(k)$ steps. After that, our critical path continues from a node on an i-ring, which is $\Theta(k)$ steps away from any border. That means $l_{i-1}, l_{i-2}, \dots, l_{\max(1,i-q)} = 1$ for a $q = \Theta(k)$. Because $l_i = O(n)$ we have a total complexity of $O(T+n)$. This concludes the proof of Theorem 9.1. ■

9.3 Increasing the error bound

The number of worst case faults from Theorem 9.1 can be improved such that $(\log n)^k$ worst case faults are allowed for a constant value k. Even though we will not prove this here, we will give an idea on how to achieve this.

When a *finished box* is completely contained within the skirt of another, we can do the same embedding internally with this box - this only boosts the constant factor. With this in mind, we call all newly created finished boxes level 0 boxes. In the merging step, if we find that a level 0 box is completely contained within the skirt of another box, we do not merge them but call the outer box level 1 box instead.

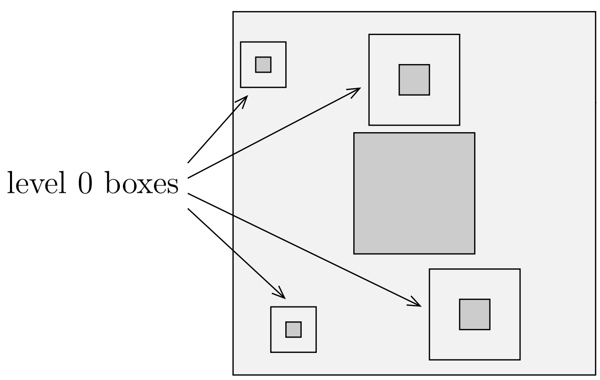

Figure 7: A level 1 box

We are not limited to level 1 boxes here – we can go up to level k boxes for some constant value k. This gives us a fault tolerance of up to $(\log n)^k$ with a running time of $O(T + kn)$.

This result can be even further improved, such that we get a constant time embedding with up to $n^{1-\epsilon}$ worst case faults for some constant ϵ. This result however requires considerably more work and will not be discussed here.