Adaptive refinement recovery after fault simulation

Linda Stals [1]

February 11, 2016

[1]Mathematical Sciences Institute, Australian National University, Canberra ACT 0200, Australia. linda.stals@anu.edu.au

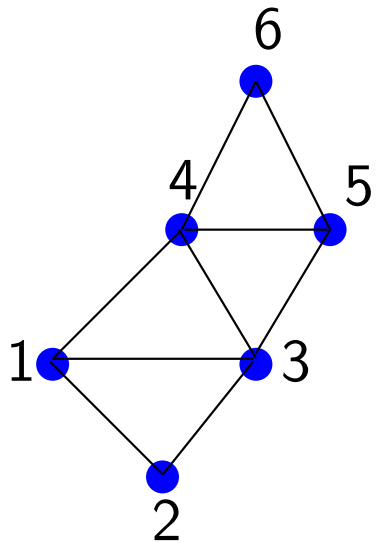
Resilience

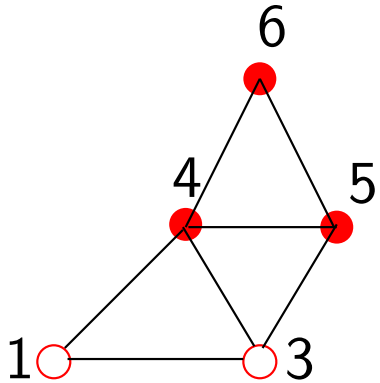
- With the advent of high performance machines containing an increasingly larger number of processors and system components, the chances of a fault occurring becomes more likely.
- Achieving resilience is expensive since it inevitably requires redundancy, and thus more system resources and additional energy. Alternatives that can exploit specific features of the algorithms may offer substantial savings.

Multilevel adaptive grids

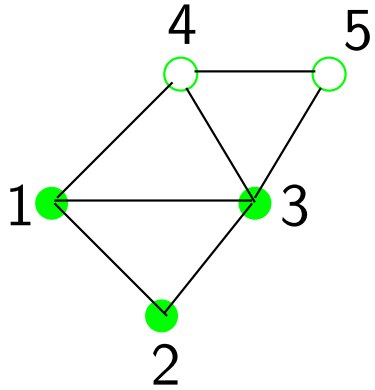
- We study parallel algorithms with automatic adaptive mesh refinement.
- Thus the grid structure itself requires nontrivial distributed and dynamic data structures that store a hierarchy of locally refined finite element meshes.
- In case of a fault not only the state of the solution within the iterative process is lost in a subdomain but also the information about the adaptively refined mesh structures themselves.
- The recovery process must take into account both the intra-grid data dependencies as well as the inter-grid dependencies.

Parallel implementation





Processor 1



Processor 2

Inter-grid connections

Observe that the algebraic connections include the inter-grid connections defined by the interpolation and restriction operators.

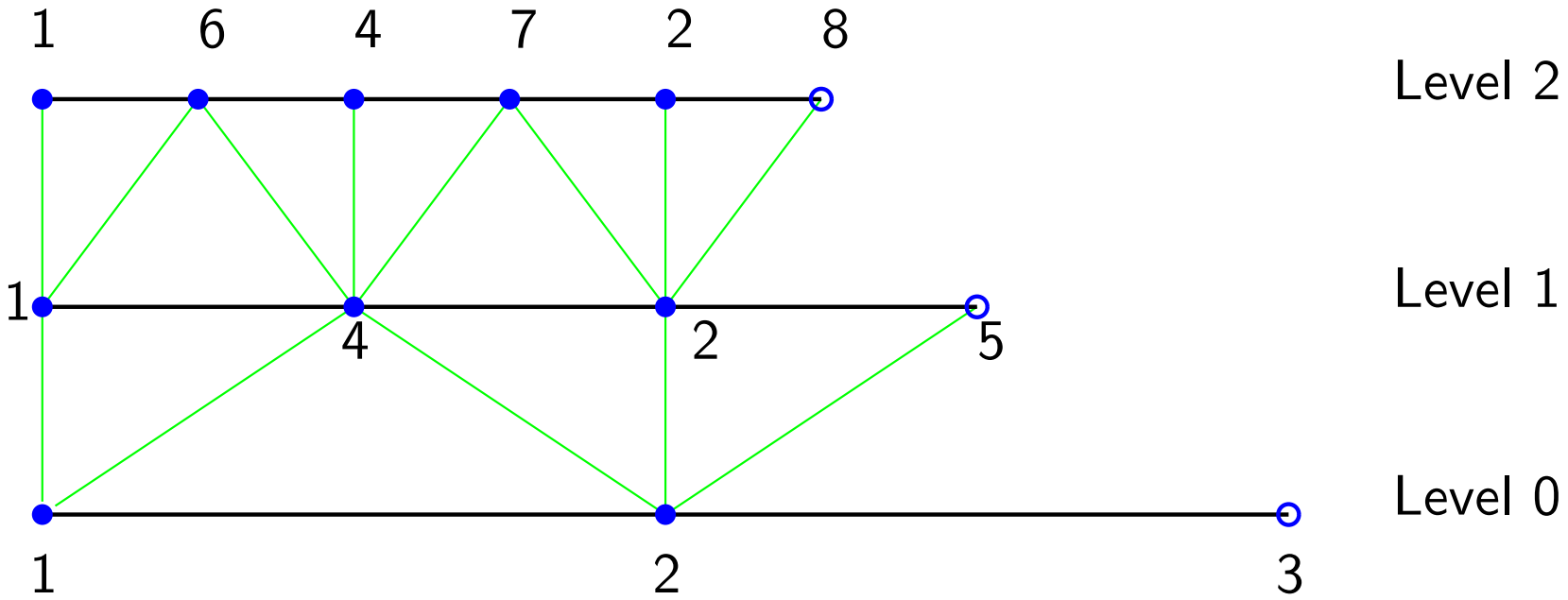


Figure: The ghost nodes are used to complete the intra-grid and inter-grid connections. The full nodes are drawn as dark circles while the ghost nodes are drawn as open circles.

Inter-grid connections

- The inter-grid connections add an extra degree of complexity to the data dependencies. For example, when a node is moved to another processor, the neighbour node list must be updated on the current grid level as well as on the previous grid level and the next grid level.
- The use of ghost nodes as a communication buffer or as a way of storing updates from neighbouring processors is standard practise. However, we have extended their application. For example, during refinement the communication pattern has to be updated when new nodes are added and we will show that by exploiting the relationship between the ghost nodes and full nodes the communication pattern may be updated independently across the processors.
- Unfortunately, it will turn out that this is not true during the fault recovery process. The load balancing routine breaks many of the assumptions or rules that are relied upon during the standard refinement procedure.

Triangle subdivision

- There are two ways to subdivide a triangle; bisection or quadrisection. We will use the bisection method as it lends more easily to adaptive refinement and the general ideas extend more readily to tetrahedral grids.
- The grids are refined by using the newest node bisection method. In this method the triangles are bisected along the edges that sit opposite the newest nodes.
- It can be shown (Mitchell references a proof by Sewell) that if the angles in the initial triangulation are bounded away from 0 and π then the angles in the refined grid will be bounded away from 0 and π. Indeed there are only a finite number of similar shapes that arise.

Newest node bisection

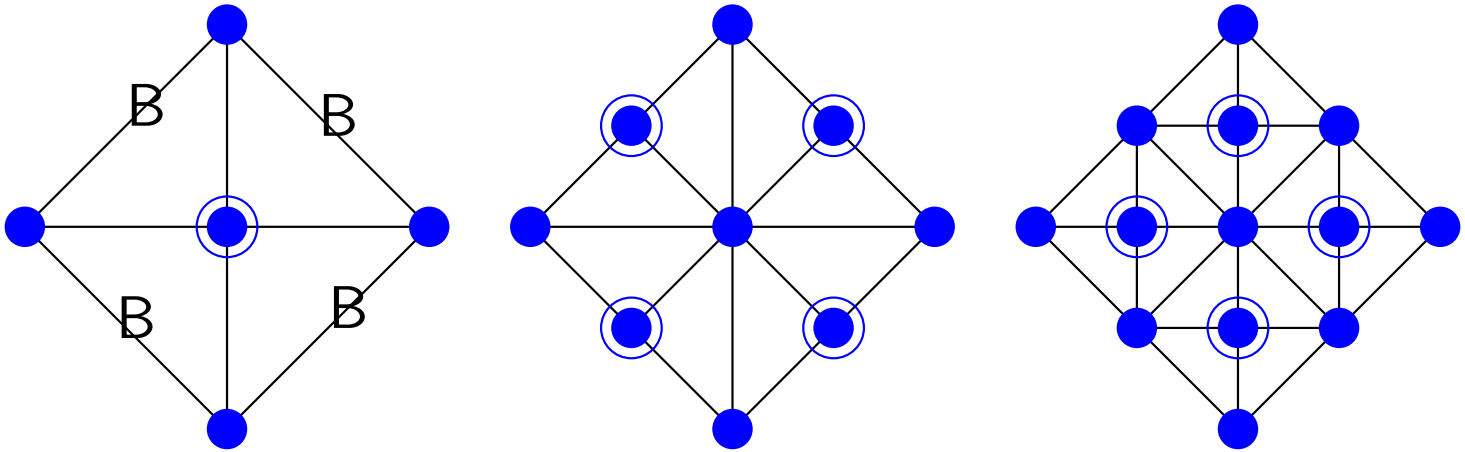


Figure: Initial Triangulation. The outlined circles represent the newest nodes.

Parallel Implementation

The parallel implementation of this method is equivalent to the steps outlined above except that it must be extended to handle the addition of new nodes.
Note that the grid sitting on processor p is given by

$$\mathcal{M}_p^m = \mathcal{M}_p^m\{\mathcal{F}_p^m, \mathcal{G}_p^m, \mathcal{E}_p^m, \mathcal{C}_p^m, \mathcal{Q}_p^m\},$$

where $\mathcal{F}_p^m$ is the set of full nodes, $\mathcal{G}_p^m$ is the set of ghost nodes, $\mathcal{E}_p^m$ is the set of edges, $\mathcal{C}_p^m$ is the set of algebraic connections and $\mathcal{Q}_p^m$ is the neighbour node tables associated with processor p.

New nodes

- If both N_i and N_j are ghost nodes we do not know if the processor contains enough of the grid to complete all of the connections. Consequently N_d is assigned as a ghost node.
- If N_i and N_j belong to the set of full nodes for processor p then the new node must be added to the full node table.
- If $N_i \in \mathcal{F}_p$ and $N_j \in \mathcal{F}_q$ where $p \neq q$. Then N_d can be added as a full node to either p or q. The midpoint, N_d, is then added to the processor that contains the smallest number of full nodes according to *population_table*.
  If processors p and q have the same number of full nodes according to *population_table* then the global I.D. is used.

Parallel refinement

The program loops through the edges table and bisects the triangles along the base edges. We can bisect the triangles independently across the processors.

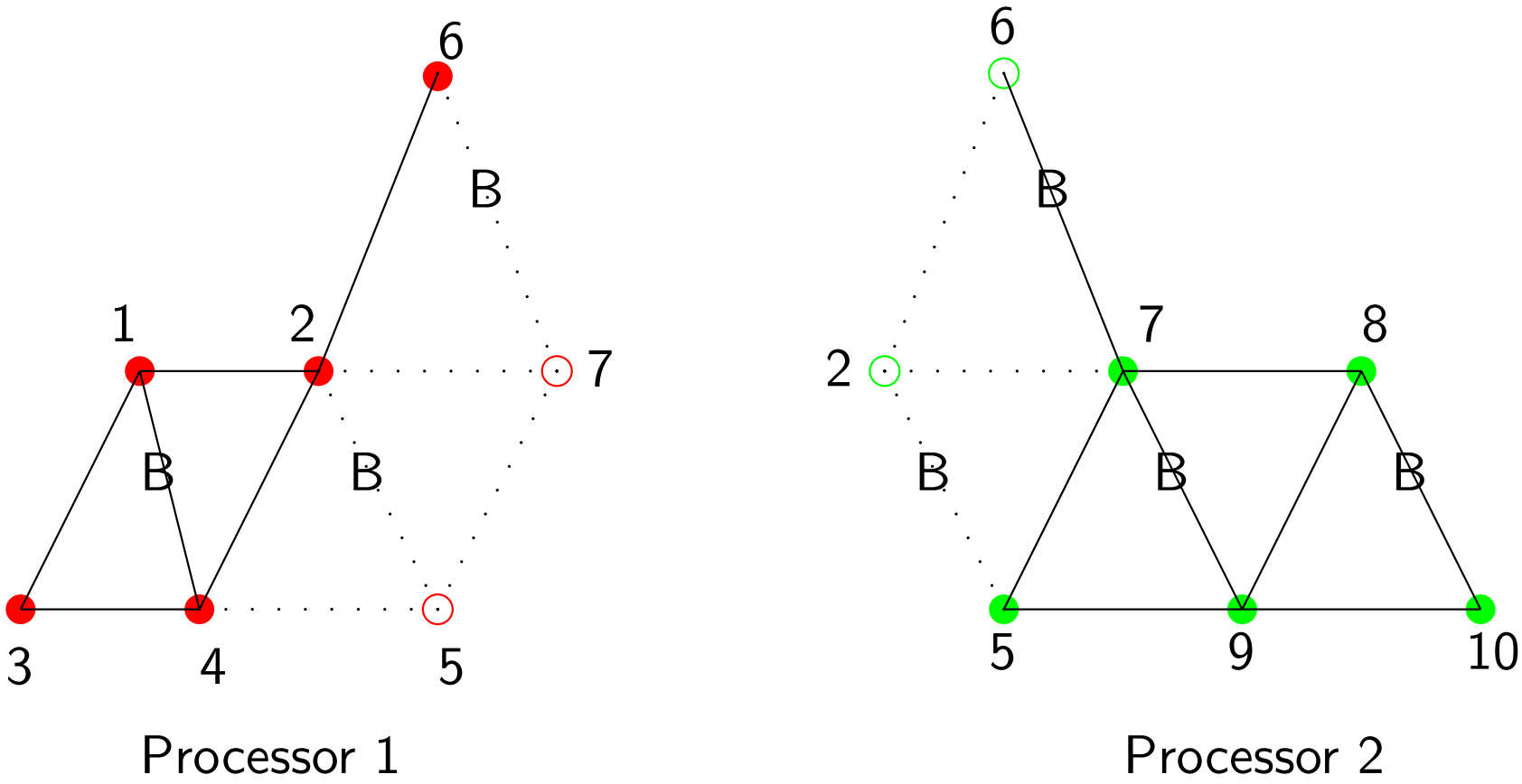


Figure: The base edges, marked by a B, show which triangles need to be bisected. The full nodes dark circles, ghost nodes are open circles.

Trim

Thus, after each level of refinement a trim routine is called to prune any ghost nodes no longer connected to a full node and to remove any full nodes from $\mathcal{Q}_{\mathcal{F}_p}^{m+1}$ that are no longer connected to a ghost node. Care must be taken to check both the intra- and inter-grid connections.

Interface base edges

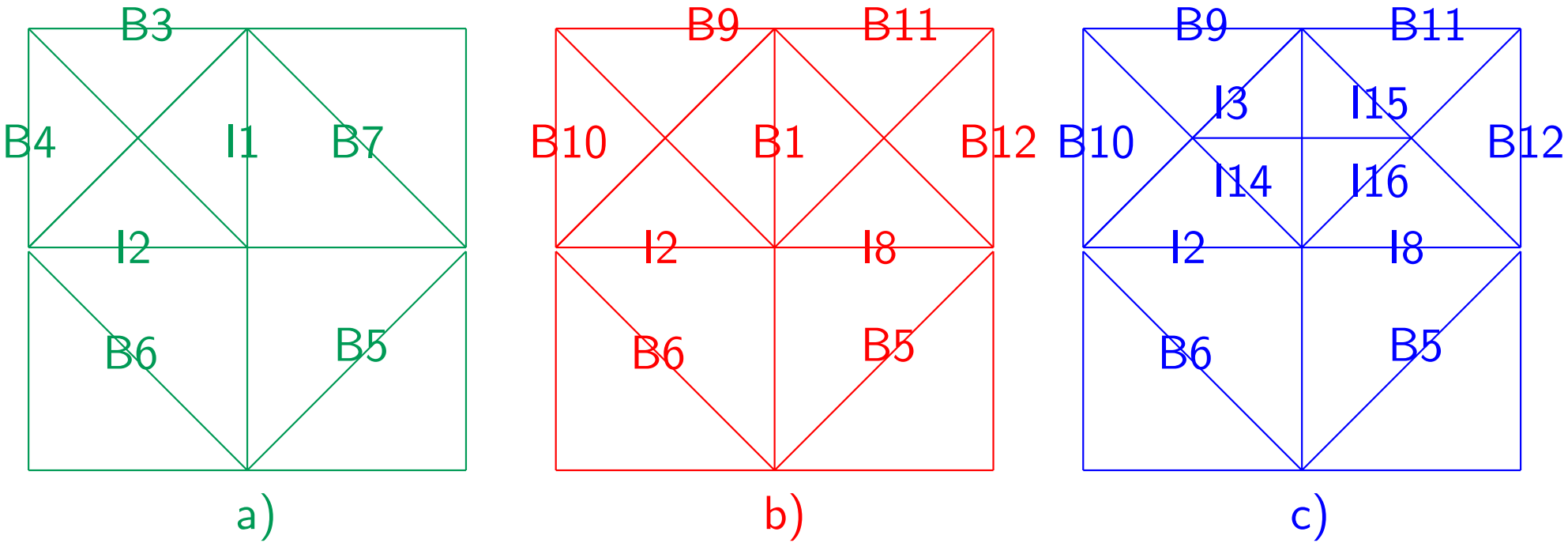


Figure: Example triangulation with interface-base edges I* and base edges B*. a) The base edge B7, should be refined before the interface-base edge I1. b) Result of bisecting base edge B7. Note that the interface-base edge I1 has been updated to a base edge B1. c) The edge B1 is now bisected to give the final grid.

Split edges

To determine the edges that must be bisected we use a recursive routine that follows the interface-base edges down the refinement levels until it reaches a base edge.

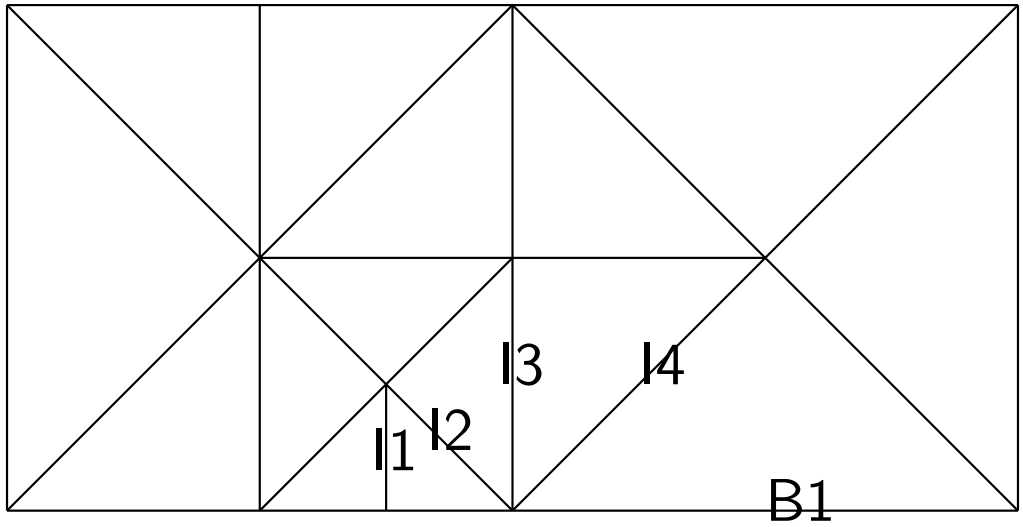


Figure: Follow the interface-edges down the coarse triangles until a base edge. is found

Parallel implementation

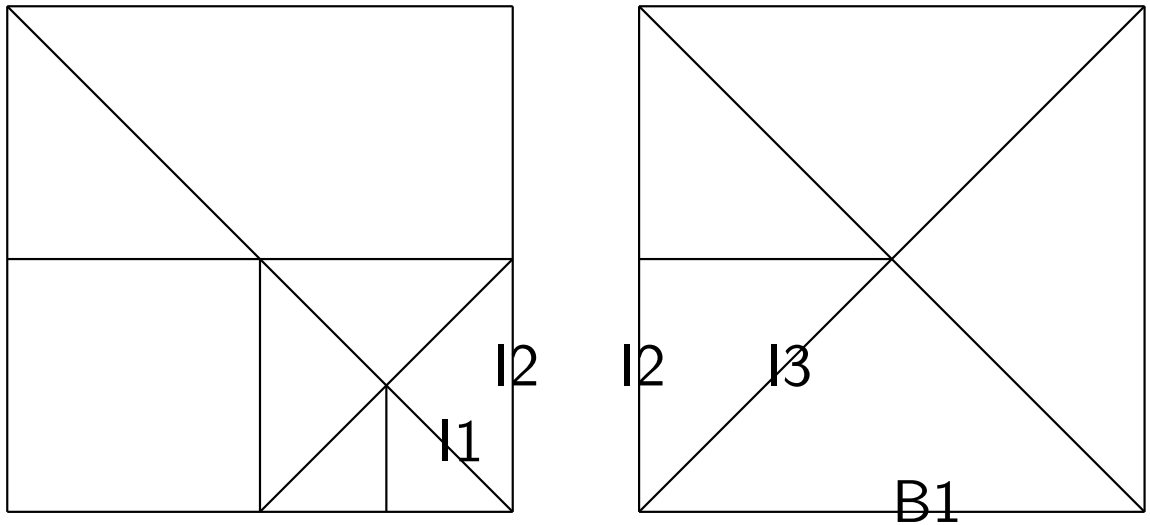


Figure: The edges B1, I3 and I2 in Processor 2 need to be bisected before edges I2 and I1 in Processor 1.

Load balancing

- After refinement, particularly after adaptive refinement, we may find that the number of nodes per processor is poorly balanced. Consequently the code uses a load balancing routine to redistribute the load.
- The load balancing routine works solely on the node-edge table, it does not take the shape of the finite elements into account.
- If a full node is moved from one processor to another on the finest grid level, then any corresponding node on any of the coarser grids will also be moved. In other words, the load balancing routine ensures that the full nodes on a given processor will be nested. The same is not necessarily true for the ghost nodes.

Load balancing

Grid reconstruction

- We next develop techniques to reconstruct the grid levels after a fault has occured.
- We simulate a fault by removing all of the grid levels from a given processor, except the coarsest level.
- In all of the approaches we discuss here we assume we that the coarsest grid can be recovered (e.g. be read in again from file).

Load balancing

The biggest challenge is taking into account the fact that the grids are being reconstructed after load balancing.

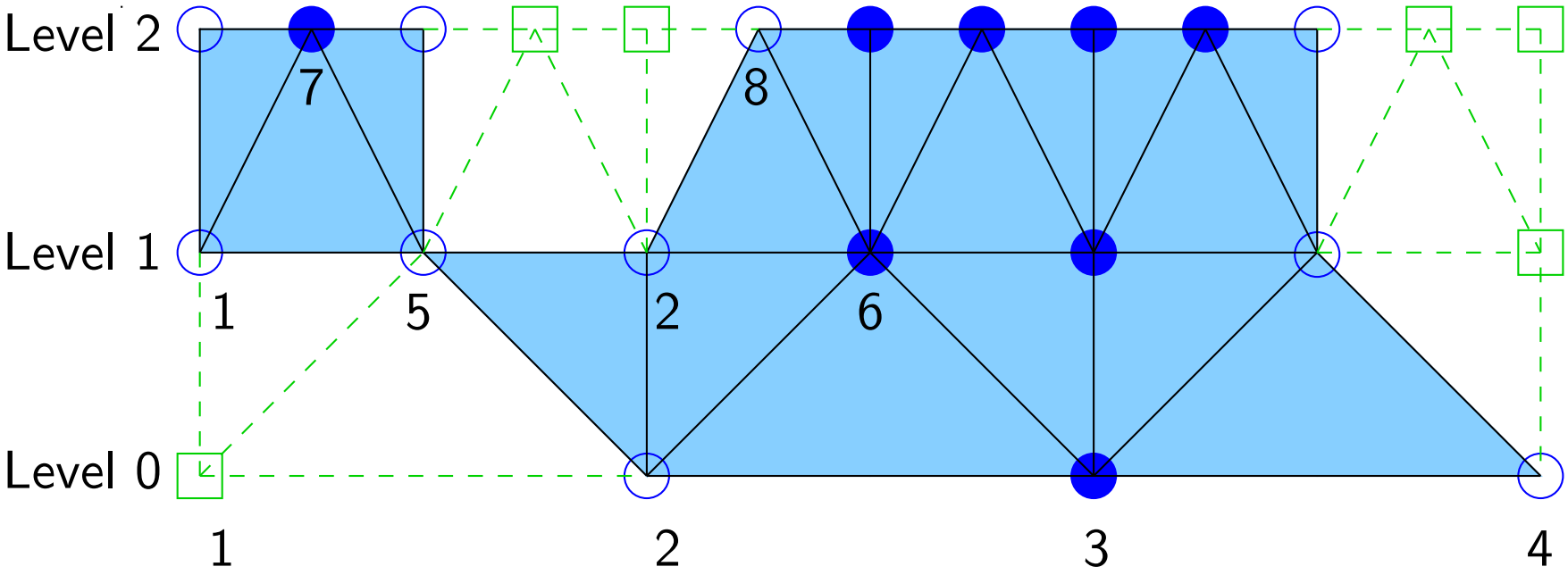


Figure: Example grid refinement. The filled blue circles represent full nodes, the open blue circles are ghost nodes and the green squares represent parts of the grid that will not be stored in the processor

Neighbouring grids

- Unlike the original refinement routine communication is needed to recover the grids. The purpose of the communication calls is to build two grids; $\widehat{\mathcal{M}}_G^{m+1}$ and $\widehat{\mathcal{M}}_F^{m+1}$.
- Neighbouring, healthy, processors send copies of their ghost nodes to be stored in $\widehat{\mathcal{M}}_F^{m+1}$.
- Neighbouring, healthy, processors send copies of their full nodes to be stored in $\widehat{\mathcal{M}}_G^{m+1}$.
- Neighbouring processors also send the edges and connections joined to the full nodes to be stored in $\widehat{\mathcal{M}}_G^{m+1}$.

Ghost vs full nodes

In our fault recovery routine the process of bisecting a triangle is exactly the same as described previously with only one exception. Rather than relying on a set of rules to determine which processor contains a full node copy we use the information from the neighbouring processors given in $\widehat{\mathcal{M}}_G^{m+1}$ and $\widehat{\mathcal{M}}_F^{m+1}$.

Missing information

The edge between Node 1 and Node 2 in Level 0 is not stored in the processor so the refinement routine will not know that the edge needs to be bisected when building Level 1. The extra information stored in $\widehat{\mathcal{M}}_G^{m+1}$ addresses that issue.

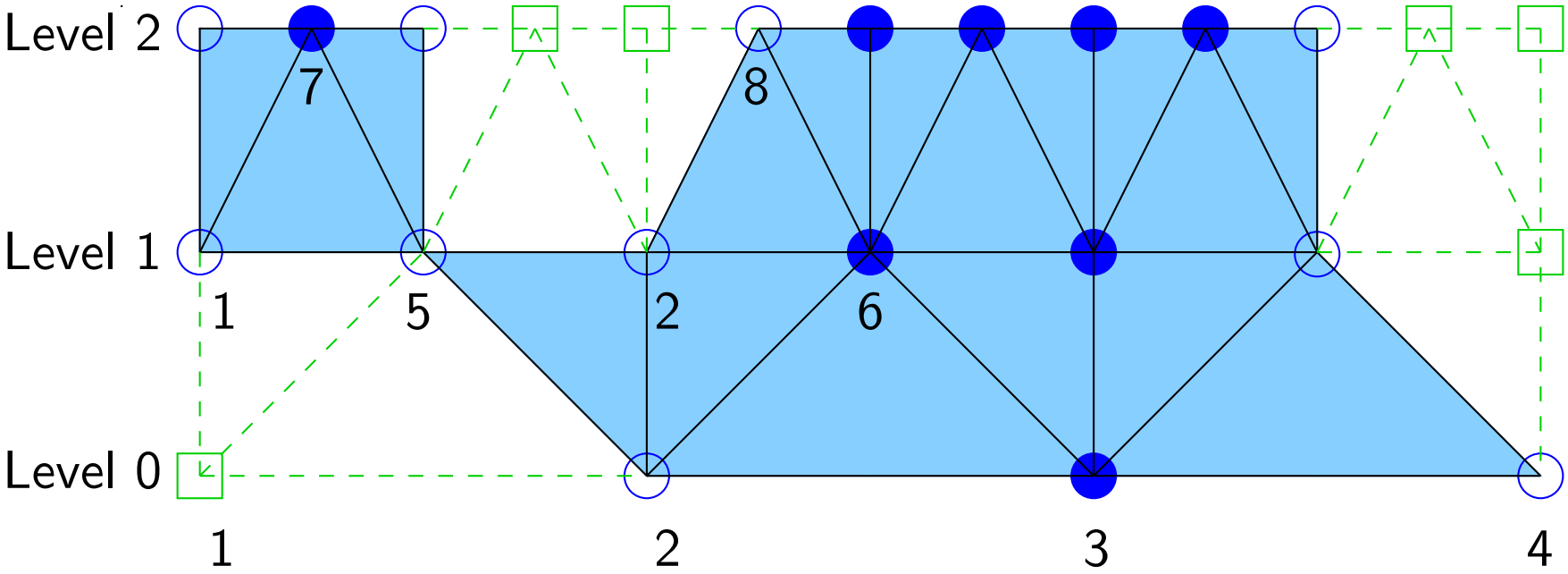


Figure: Example grid refinement. The filled blue circles represent full nodes, the open blue circles are ghost nodes and the green squares represent parts of the grid that will not be stored in the processor

Adjust full

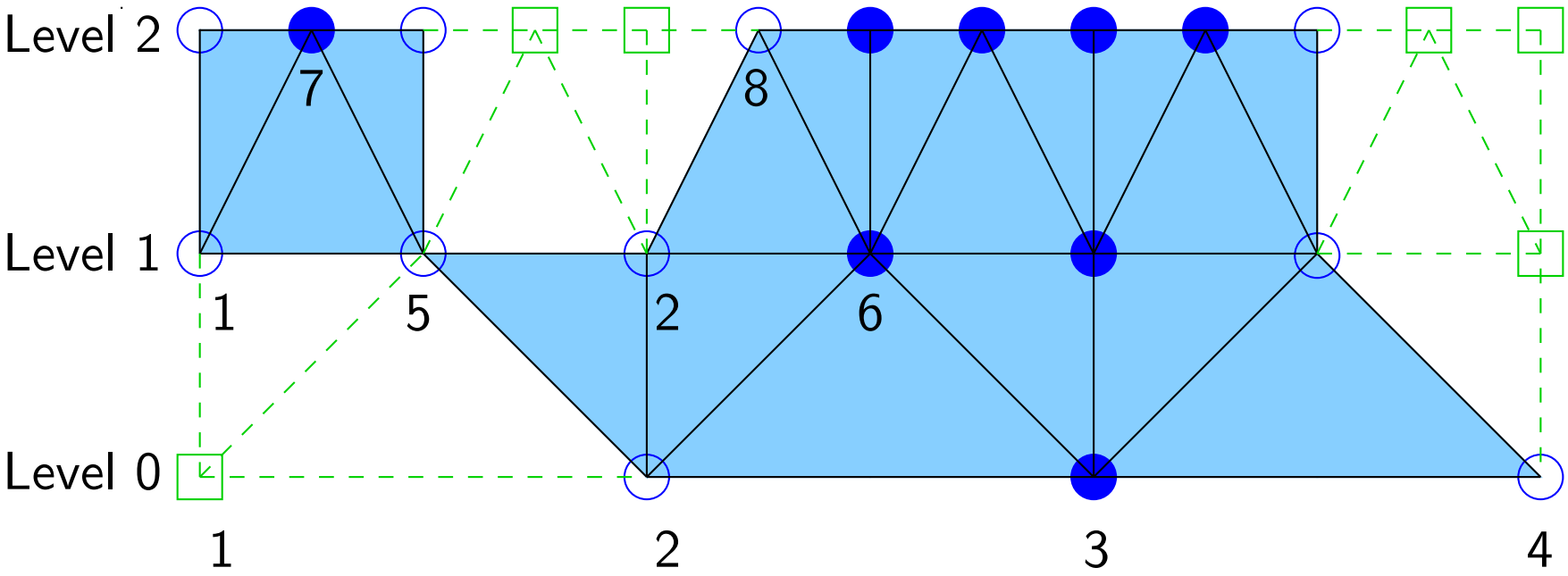


Figure: Example grid refinement. The filled blue circles represent full nodes, the open blue circles are ghost nodes and the green squares represent parts of the grid that will not be stored in the processor

Information stored in $\widehat{\mathcal{M}}_F$ is used to correct the full neighbour node table in processor p.

Adaptive refinement

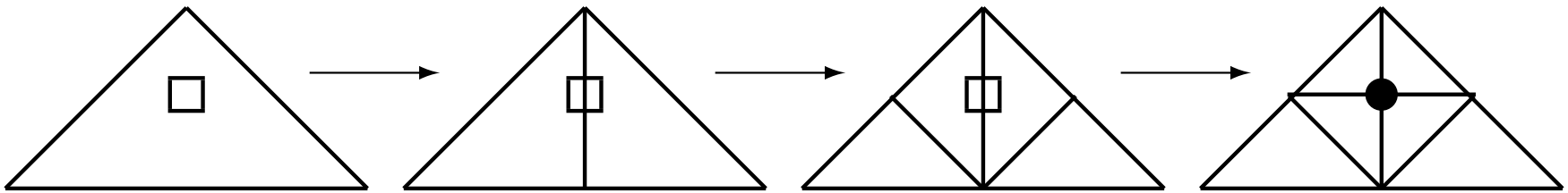


Figure: The nodes in the healthy processors can be used to guide the refinement in the faulty processor. The square represents a node that exists in a healthy processor. If that node sits within a triangle in the faulty processor, that triangle must be bisected until the node is added to the faulty processor.

Results

- With non-adaptive refinement we are able to reconstruct the original grid. Once the system of equations have been built, the computations can continue as usual and the results have the same accuracy that would have been achieved if no fault had occurred.
- With adaptive refinement we do not fully reconstruct the grid. We do however reconstruct enough so that the computations may continue, albeit with the final result having a reduced degree of accuracy.

Non-adaptive 1D

The model problem used in this case is $u_{xx} = -4\pi \sin(2\pi x)$ on the line $0 \leq x \leq 1$. The grid was divided amongst three processors and six levels of non-adaptive refinement were carried out.

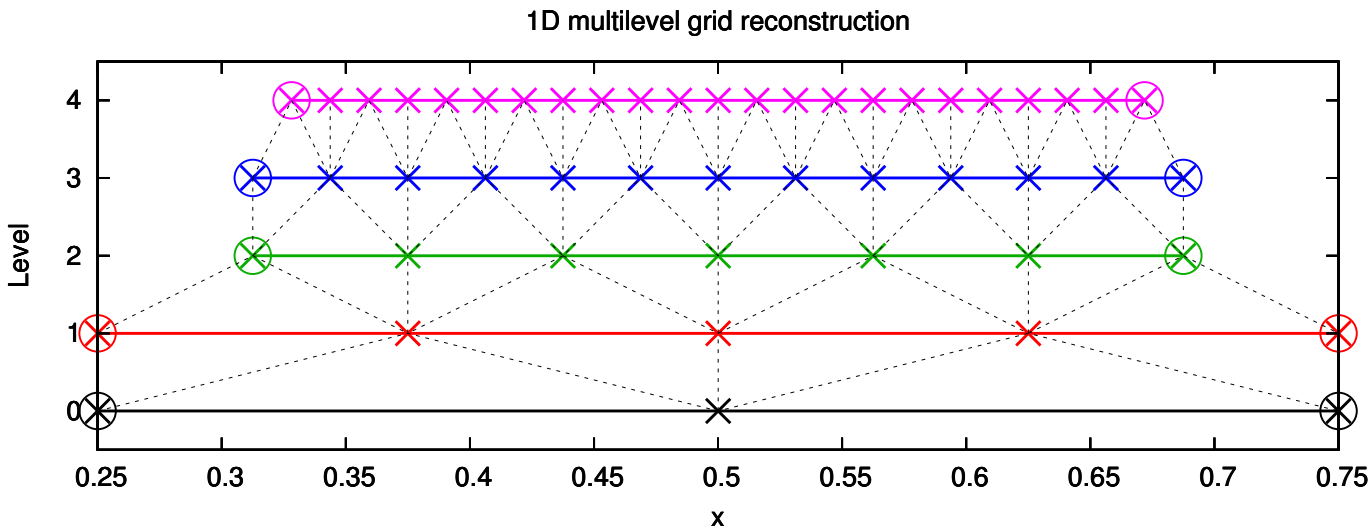


Figure: Example multilevel 1D grid recovered after fault simulation. The nodes joined by a line are the full nodes, while the nodes enclosed in a circle are the ghost nodes

Non-adaptive 1D

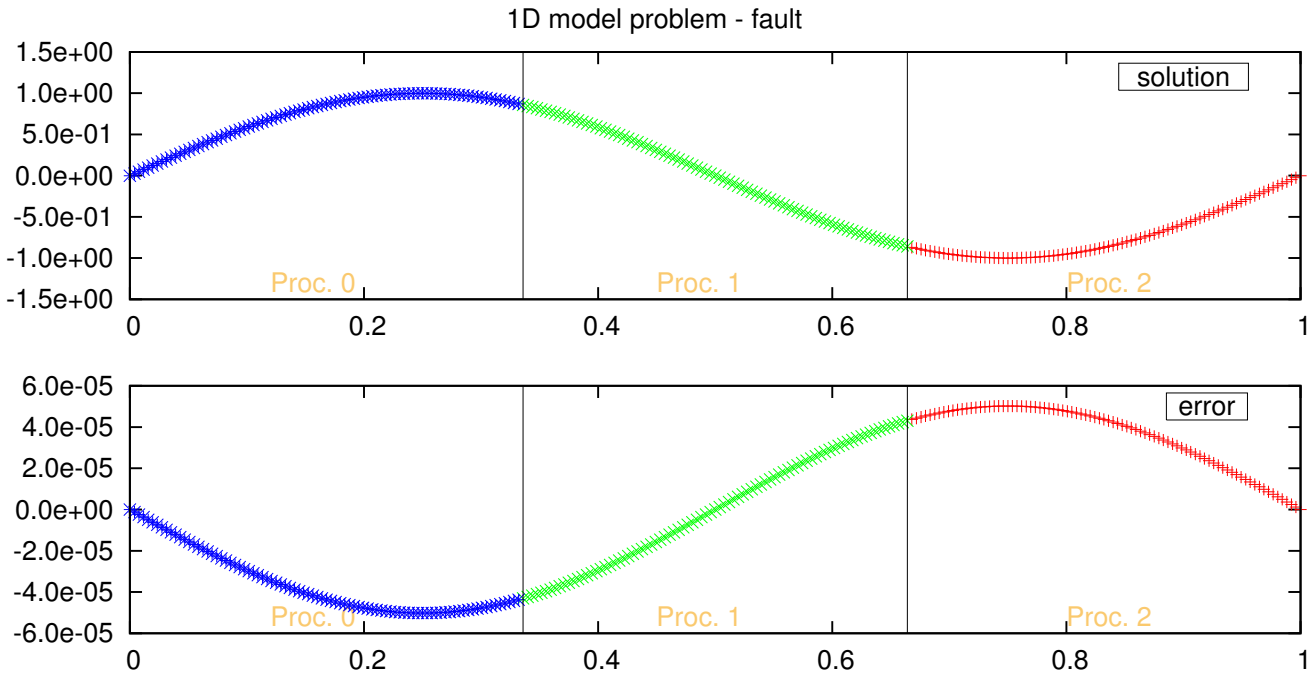


Figure: Solution and error of 1D model problem with fault simulation.

Non-adaptive 1D

- Model problem $u_{xx} = -4\pi \sin(2\pi x)$ and increased the grid size by using eighteen levels of refinement giving 1048577 nodes on the finest level.
- Runs were carried out on 1 to 32 processors.
- In all cases the original grid is recovered in the (simulated) faulty processor.
- To ensure that the original grid is indeed being recovered we built and solved the system of equations and then calculated and checked the error norms of the solution.
- The expected convergence rate was observed even when a fault occurred.

Non-adaptive 2D

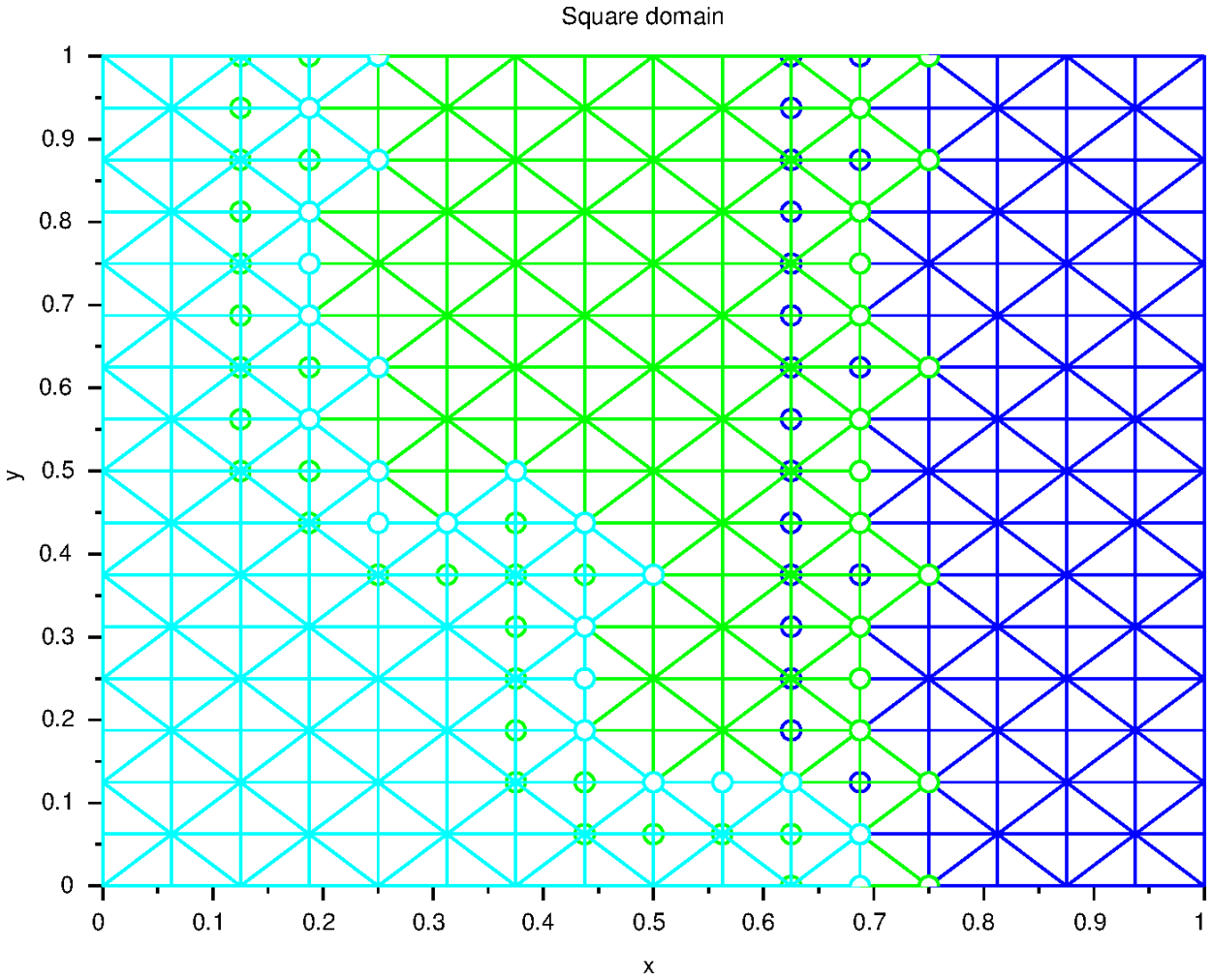


Figure: Example 2D grid divided over three processors (before fault simulation).

Non-adaptive 2D

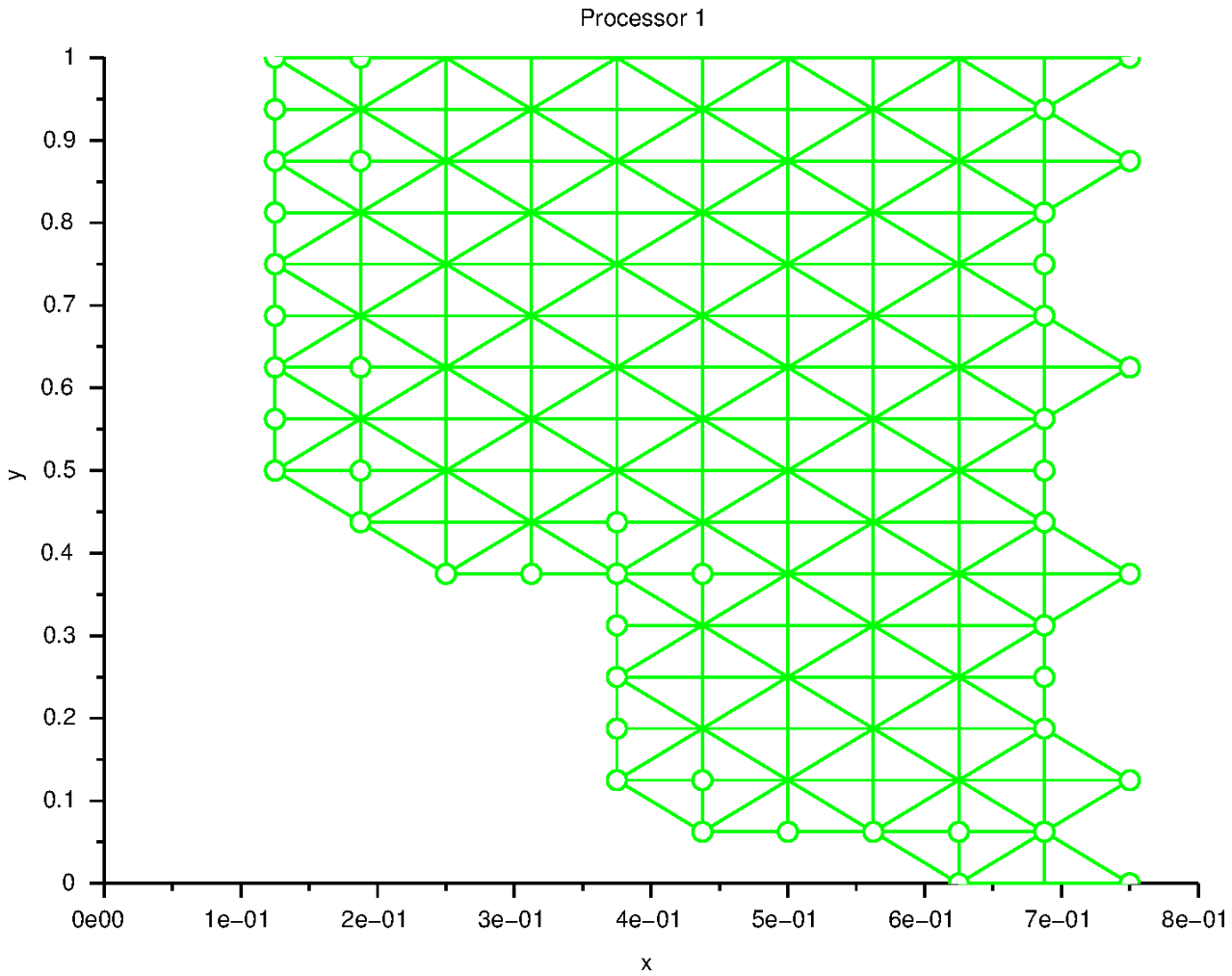


Figure: Example 2D grid recovered after fault simulation.

Non-adaptive 2D

- To test the recovery process in two dimensions a model problem is $\Delta u = \sin(\pi x)\sin(\pi y)$ on the square domain $0 \leq x, y \leq 1$.
- Refinement routine was used to construct 10 levels of grids, with the finest grid consisting of 1050625 nodes.
- Runs were carried out on 1 to 32 processors.
- The error norm was again checked to ensure that the solution is the same irrespective of whether a fault occurred.

Adaptive 1D

With adaptive refinement the original grid is not fully reconstructed. This is as expected since the recovery procedure only ensures that the communication pattern is recovered.

Adaptive 1D

Laplace's equation with Dirichlet boundary conditions where the exact solution is given by $\exp\left(-50(0.5-x)^2\right)$.

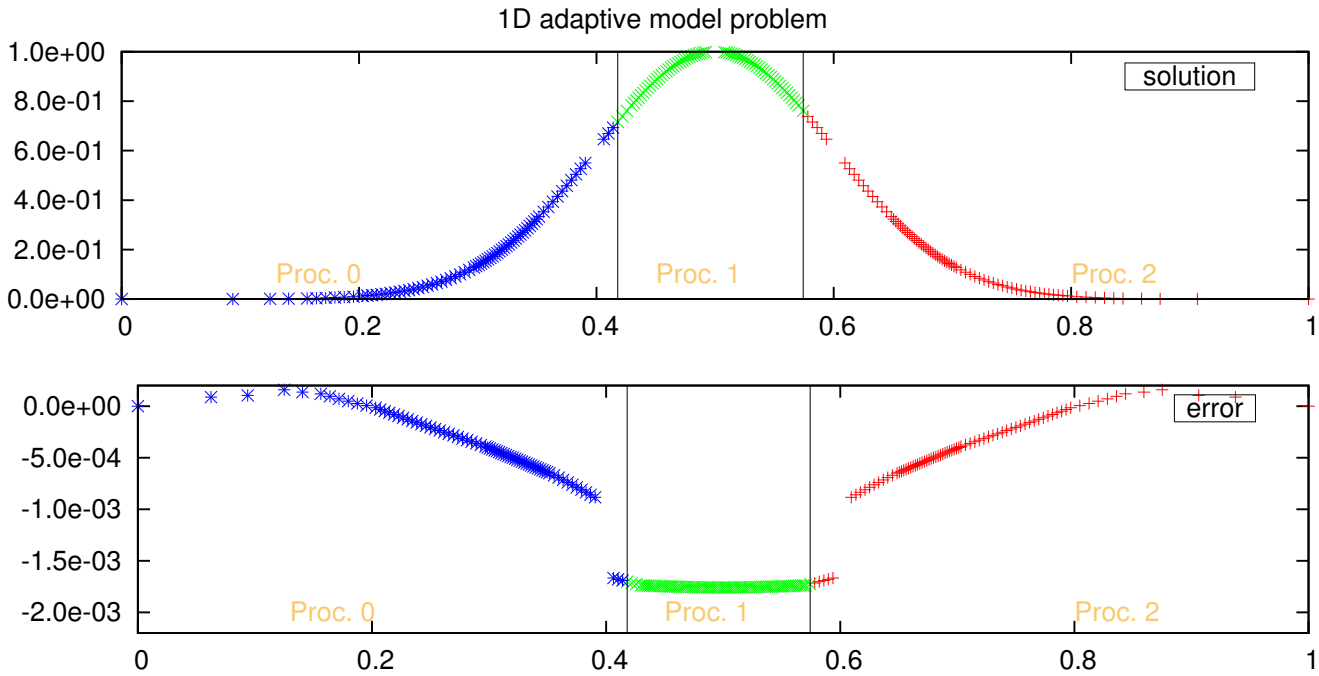


Figure: Solution and error of 1D model problem on adaptive grid without fault simulation.

Adaptive 1D

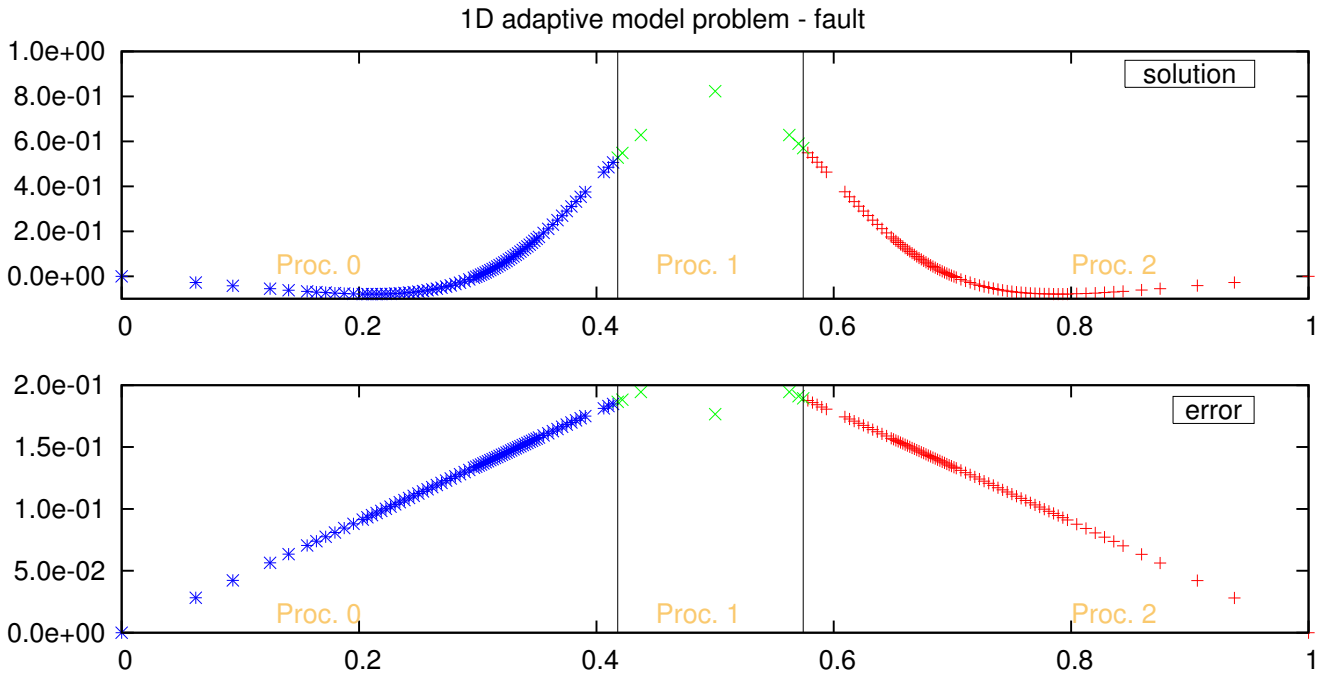


Figure: Solution and error of 1D model problem on adaptive grid with fault simulation.

Adaptive 1D

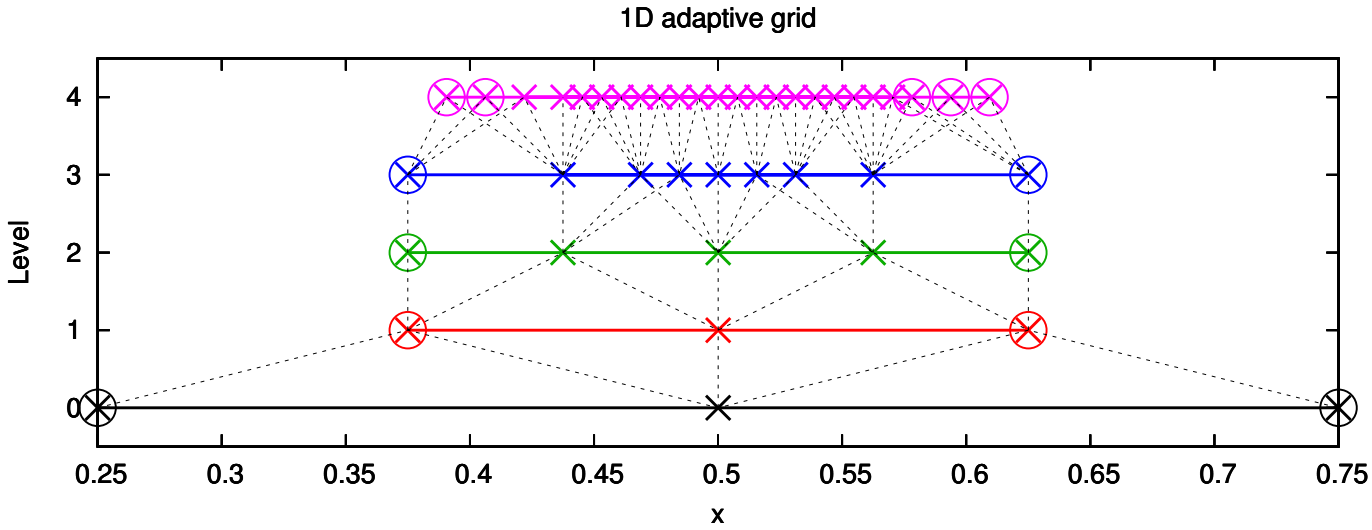


Figure: Example multilevel 1D grid after two levels of uniform refinement followed by four levels of adaptive refinement.

Adaptive 1D

Grid that is recovered after a simulated fault. Clearly the original grid is not fully recovered.

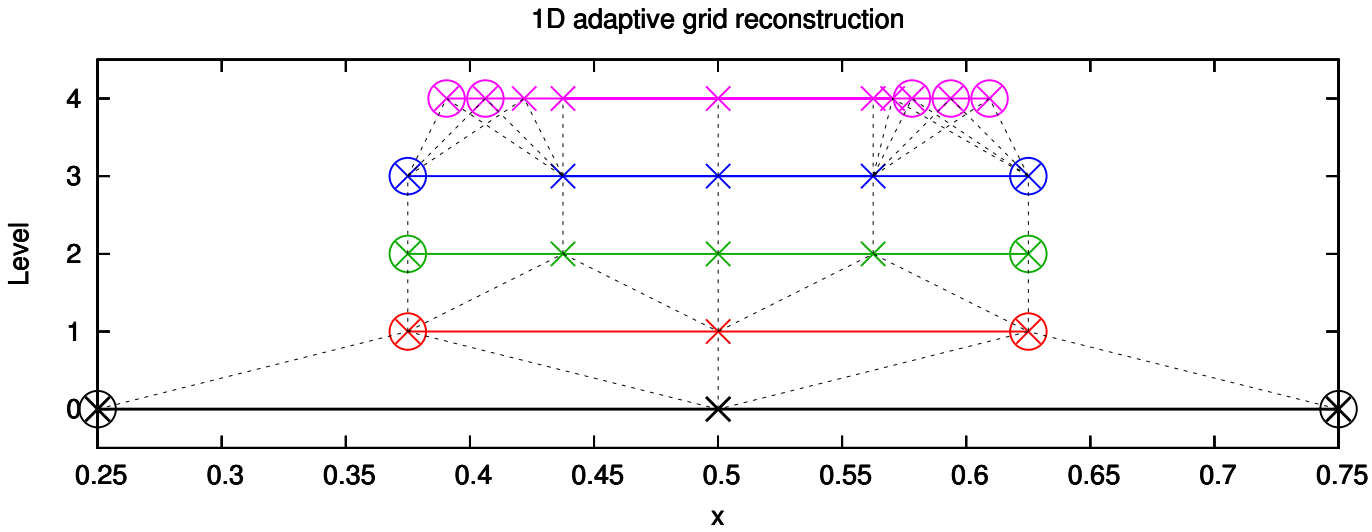


Figure: Example multilevel 1D adaptive grid recovered after fault simulation. The nodes joined by a line are the full nodes, while the nodes enclosed in a circle are the ghost nodes

Adaptive 2D

L shaped domain, Poisson equation with zero Dirichlet boundaries.

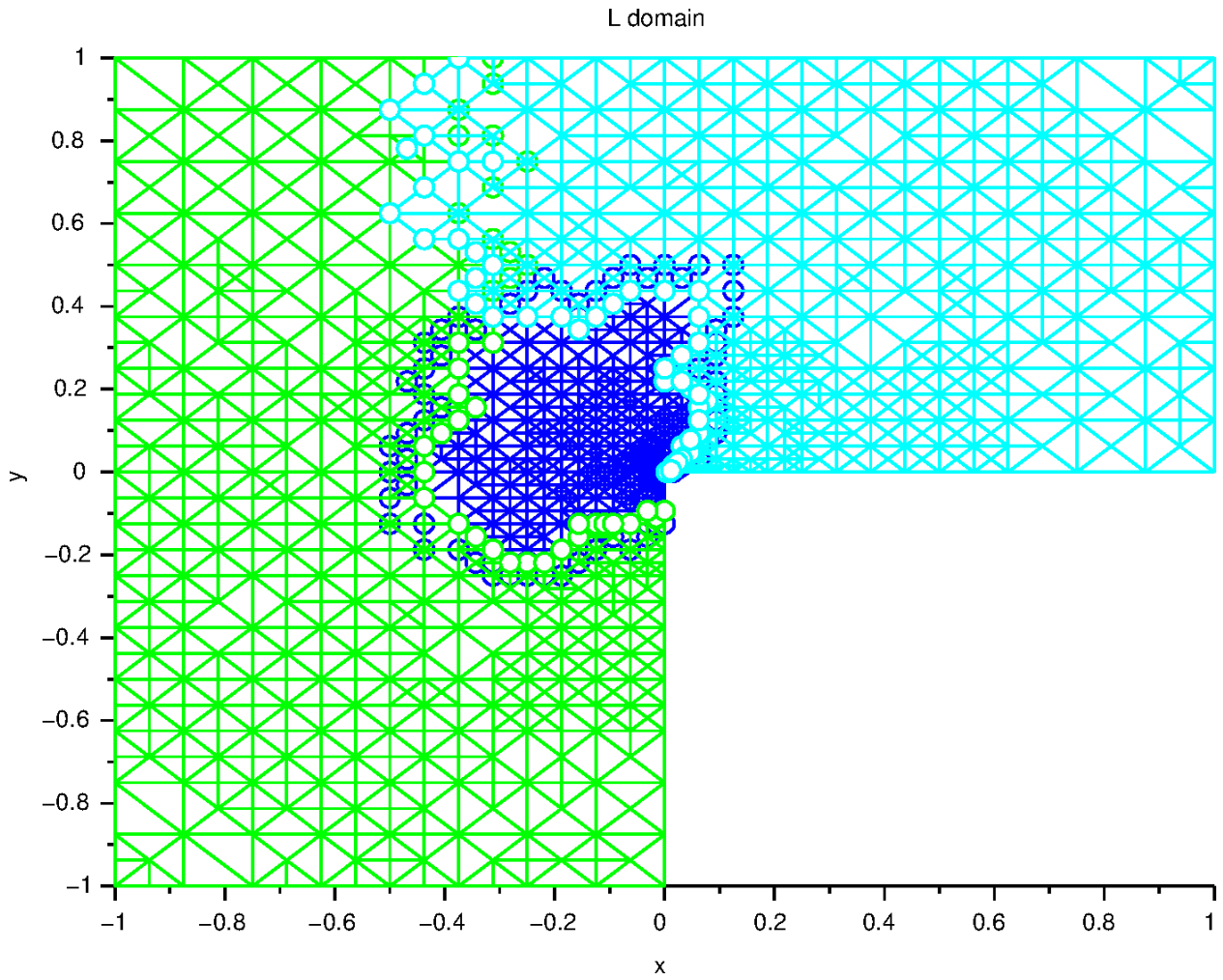


Figure: Example 2D adaptively refine grid distributed over three processors.

Adaptive 2D

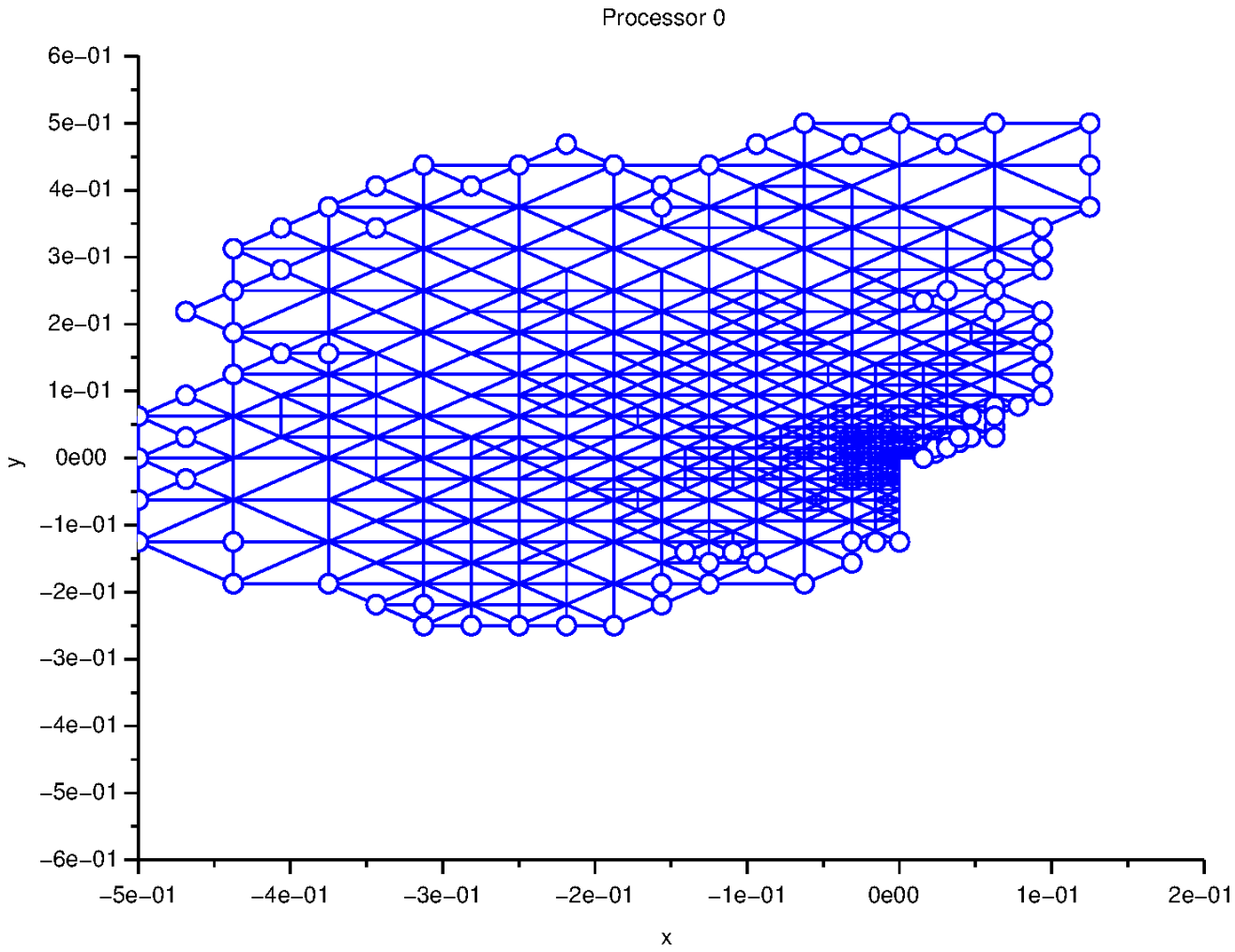


Figure: A close-up of the part of the grid stored in Processor 0.

Adaptive 2D

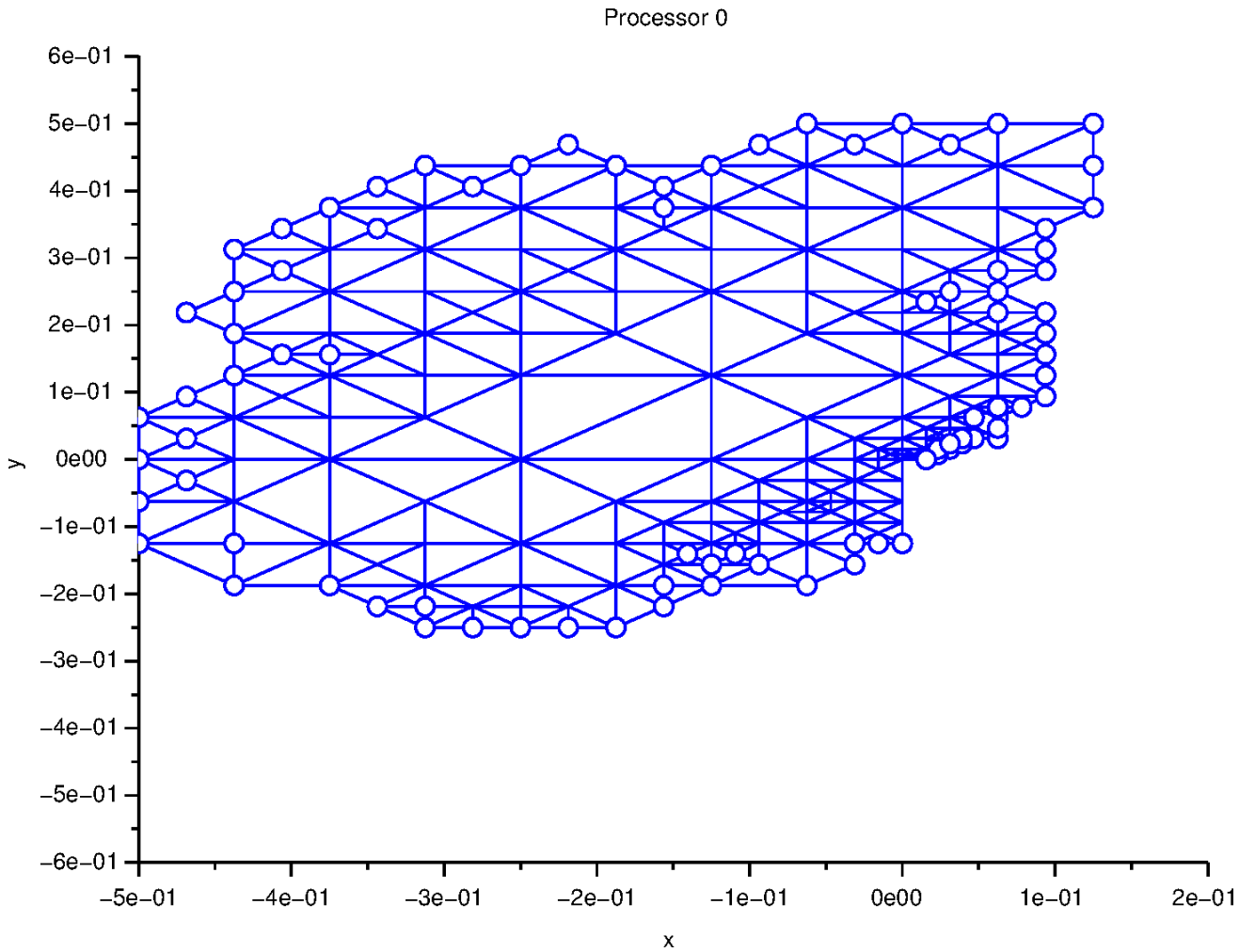


Figure: The part of the grid stored in Processor 0 recovered after fault simulation.

Adaptive 2D

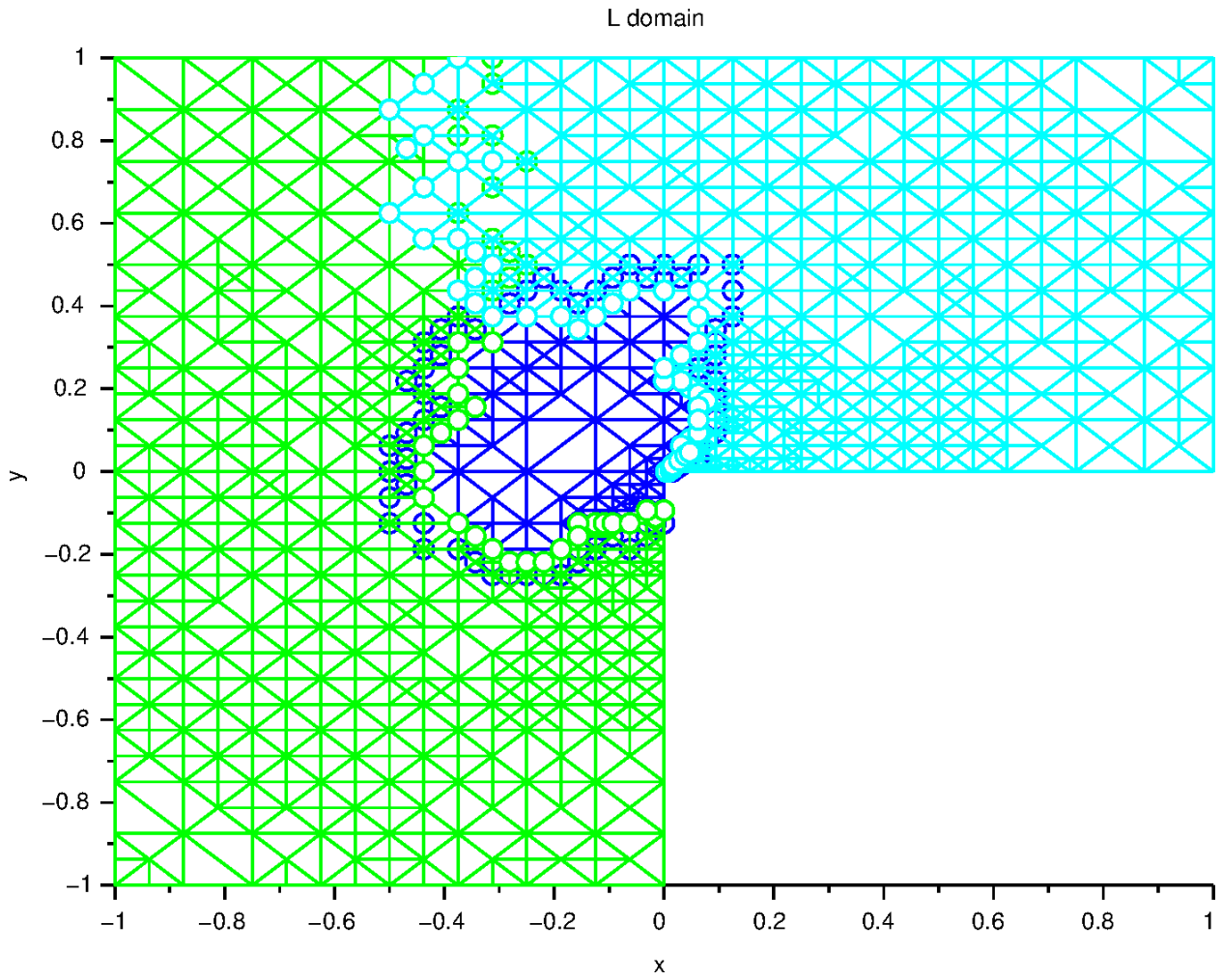


Figure: Example multilevel 1D grid recovered after fault simulation.

Adaptive 2D

- Ran a test problem consisting of 2 levels of uniform refinement followed by thirteen levels of adaptive refinement, resulting in a fine grid with 176081 nodes.
- The tests were carried out on 1 to 32 processors.
- After each fault recovery the solver was called to ensure the grid had been recovered correctly.

Recover missing data

We present some initial attempts to recover the missing data.

- Fault recovery routine reestablishes communication pattern.
- So can call adaptive refinement routine again to fill in region in interior of faulty domain.
- Currently apply the refinement routine to the whole domain, but it should be possible to modify the refinement routine to only work on the faulty processor.

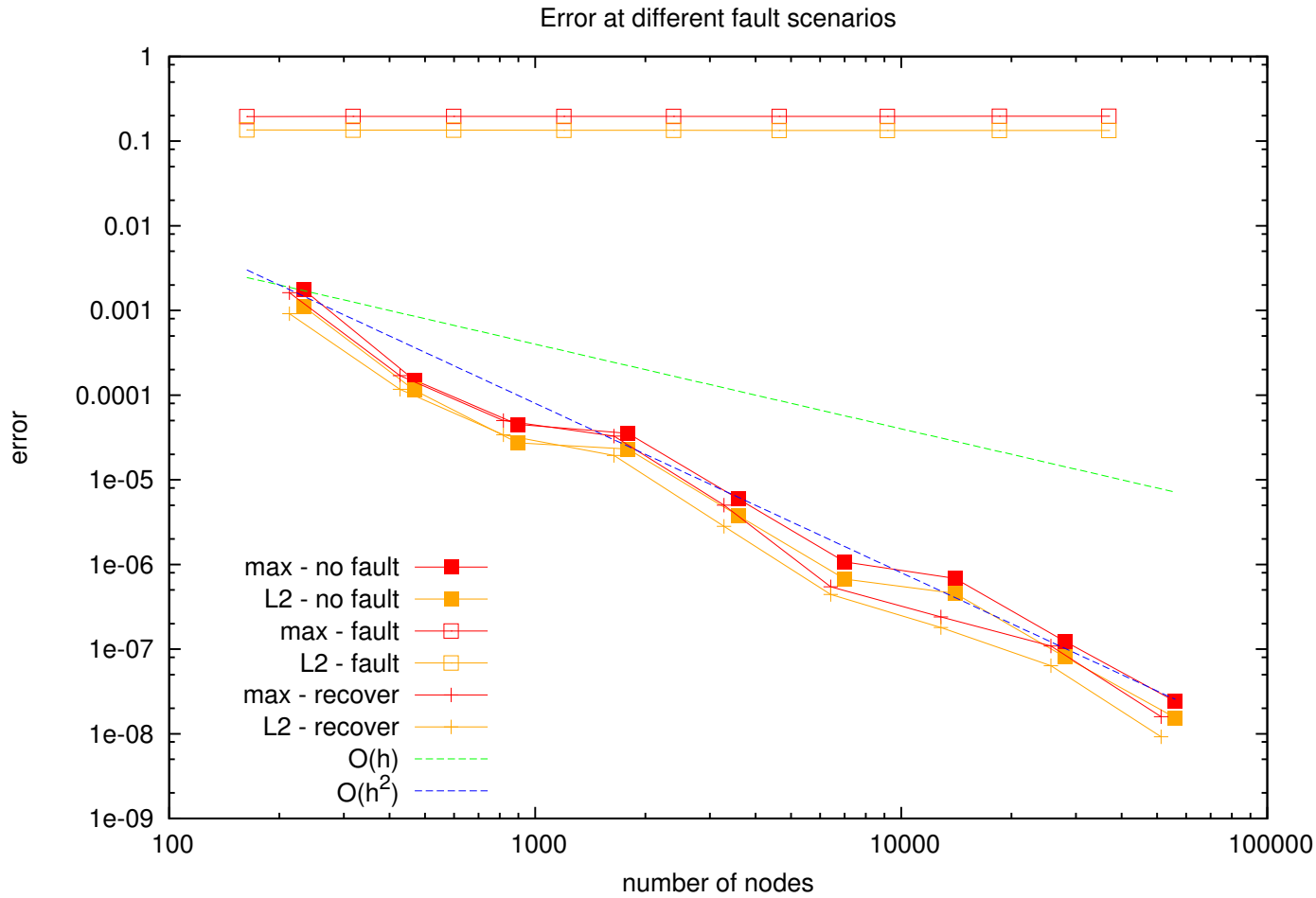


Figure: The maximum and discrete l_2 norm of the error for the 1D model problem.

MG convergence

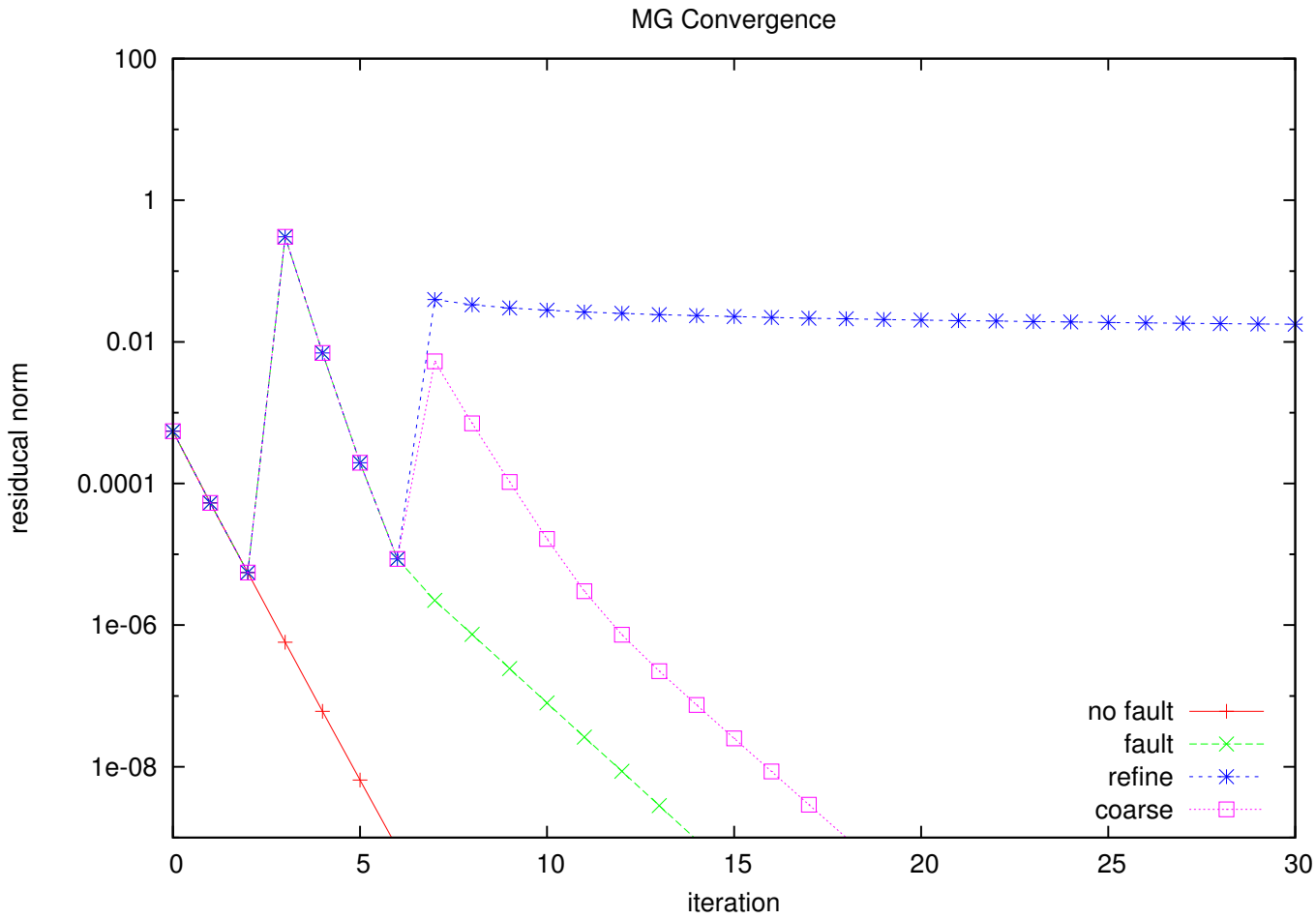


Figure: The discrete l_2 norm of the residual for the 1D model problem.

Conclusion

- We presented an algorithm based fault recovery routine for adaptively refined multigrids.
- The algorithm does not fully recover the initial grid rather, it recovers enough of the data structures to ensure that the communication pattern has been reestablished.
- Once the data structures are again consistent the computations may continue, but potentially with reduced accuracy.
- Applying additional refinement routines can recover lost information.