We say that ${}^pA$ is *partially assembled*, because it has been assembled everywhere except at processor boundaries. Clearly, we could *fully assemble* A by communicating the shared values between processors and summing; but as we shall see, a fully assembled A would be less useful.

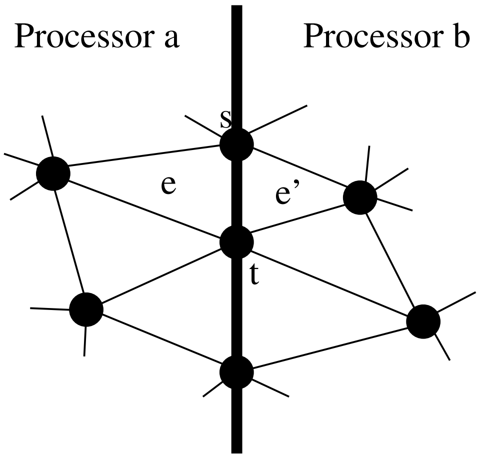

Figure 1: An element-partitioned mesh.

Again, the multiplication is performed in two phases; however, the computation phase is first. Each processor p calculates a local matrix-vector product of the form ${}^py = {}^pAx$.

The communication phase follows. For each vertex u shared by two or more processors, these processors exchange their respective entries for ${}^py_u$, and take the sum. This gives us correct values because $y = Ax = (\sum_p {}^pA)x = \sum_p {}^pAx = \sum_p {}^py$. We see that vectors, like matrices, have both partially assembled and fully assembled forms; and the product of a partially assembled matrix and a fully assembled vector is a partially assembled vector. (There is no similar property for fully assembled matrices.)

4.3 Comparison of Partitioning Approaches

When using the element-partitioning approach, processors a and b share only one "layer" of vertices at the interface between G_a and G_b, so less storage space is required for a vector than when using the vertex-partitioning approach. The sparse matrix A, which dominates memory usage, requires less storage in the element-partitioning approach if A is symmetric, and requires less storage in the vertex-partitioning approach if A is asymmetric (for reasons we omit here). The element-partitioning approach makes initial matrix generation much easier, because A is built element-by-element, so no communication is required.

So far, these are small differences. There are two far greater advantages to the element-partitioning approach. First, in nonlinear problems where the matrix A changes over time, A can be recomputed without communication.

Second, the element-partitioning approach allows assembly of a vector to be delayed, and thus may reduce the amount of communication which must be done. For instance, suppose one wishes to calculate $z = Ax + By$, where A and B are sparse matrices with the same sparsity structure. (This arises in the solution of second-order differential equations.) The products Ax and By each produce a partially assembled vector. These two partially assembled vectors can be summed locally to form a third partially assembled vector; this vector, if necessary, can be assembled to form the result z. Note that only one assembly, and thus only one communication step, is needed for two matrix-vector products. This is called the *delayed assembly optimization.*

Because of the communication reduction and speed advantages of matrix generation and the delayed assembly optimization, we strongly prefer the element-partitioning approach.

4.4 Code Generation Issues

Our goal is a code generator for finite element methods, that takes as its input the mesh M and a machine-independent description of a finite element algorithm, and automatically partitions the problem and generates multiprocessor code to execute the algorithm.

A code generator can handle the delayed assembly optimization in two ways. One is to determine at compile time when vector assembly should