in Appendix B.

Four sets of timing are shown: 1) where array updating is always done destructively, 2) where the fragmented shallow binding scheme is used, 3) where the shallow binding scheme is used, and 4) where the whole array is copied when being updated. The same program is used in all four sets of experiments; it just uses different implementation of the array module in each case. Also note that the destructive updating implementation of functional arrays is not applicable to the 8–queens, transitive closures, and histogram programs because those programs are multi–threaded.

According to the results in table 5.2, the fragmented shallow binding scheme performs well. It is as fast as the shallow binding scheme when arrays are mostly used in single–threaded ways; and is far better in case when a program uses arrays mostly in a multi–threaded way, as demonstrated in the histogram example. But it is clear from the 8–queens example that for the fragmented shallow binding scheme to be effective, the array size had better be large.

## 5.6 Future Work

There remains several performance issues to be addressed. For example, does the fragmented shallow binding scheme perform well in a lazy functional language? How about in a parallel system where the voluminous read sequences are not coordinated and may intervene with one another?

We believe the fragmented shallow binding scheme will perform well in a lazy evaluation setting because an update operation is in fact carried out lazily. In our scheme, the result of an *Update* operation is computed only if there is a need to read the resulting array afterward. That is, the rerooting operation, which makes the desired version of an array current, is performed (if necessary) only during a *Read* operation, not during an *Update* operation. On the other hand, however, it remains to be seen if read operations will typically lead to voluminous sequences in a lazy evaluation setting.

Regarding uncoordinated voluminous *Read* sequences in a parallel system, it has been observed that performance will suffer if the read sequences intervene with one another. This is because the cost spent in rerooting may not be paid off by voluminous reads. In the worst case, each read in a voluminous sequence may have to do a long rerooting. But

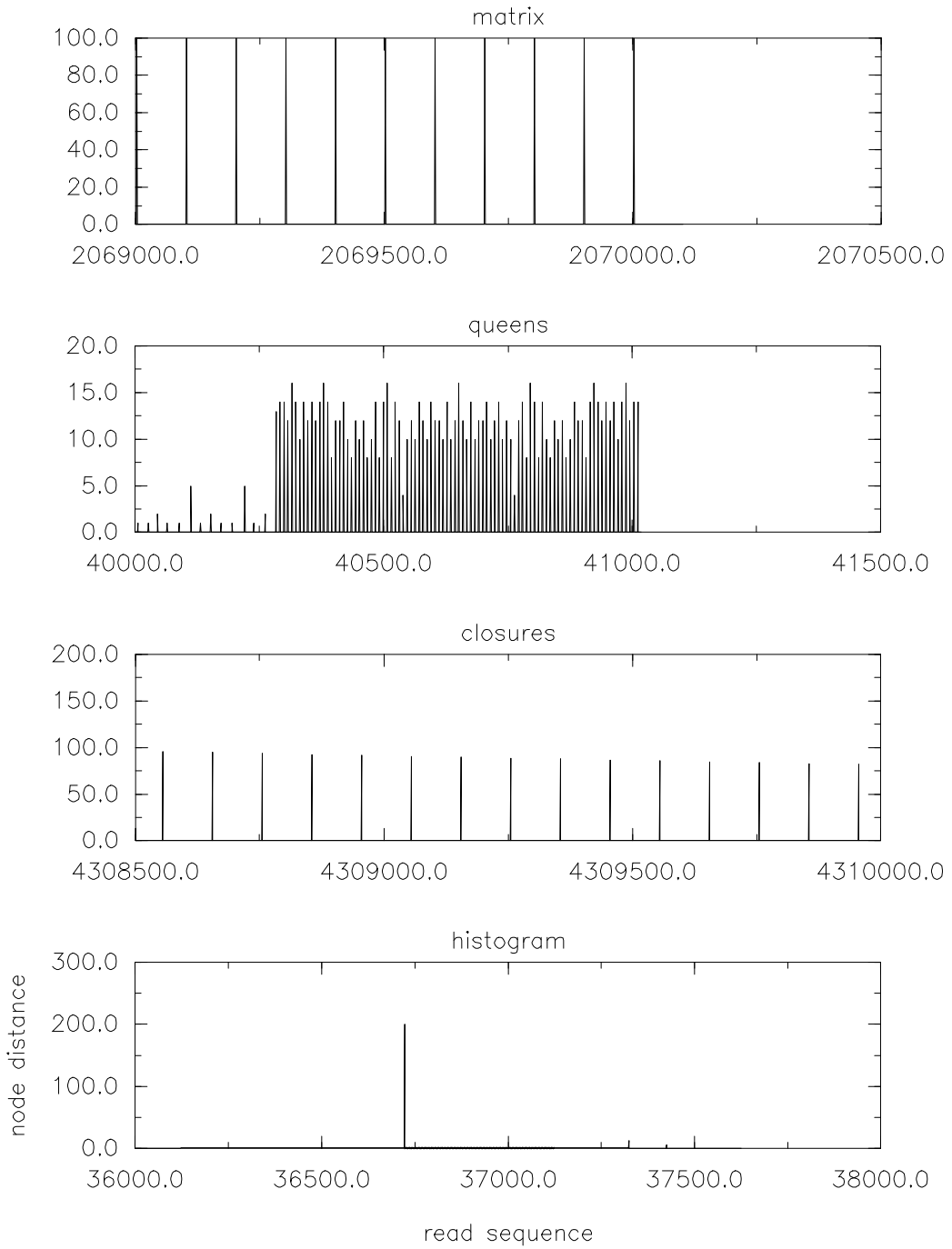


Figure 5.6: Magnified versions of the trace graphs.

**Note**. Only some interesting portions of the traces are shown.