access requirement i to the new location j is

$$\Delta EST_C(i,j) = \Delta A_i - k_j + \sum_{s \in S} \Delta A_s. \quad (2)$$

Moving versus Copying: Let T be the set of access requirements whose span will increase by accessing d_{new} instead of d_{old}. Further, let k_{old} be the number of access requirements in whose span d_{old} is. If we force all the access requirements that uses d_{old} to use d_{new} and then delete d_{old} – in other words, if we move d instead of copying – then the benefit of this move would be given by

$$\begin{aligned}\Delta EST_M(i,j) &= \Delta A_i - k_j + \sum_{s \in S} \Delta A_s + \sum_{t \in T} \Delta A_t + k_{old} \\ &= \Delta EST_C(i,j) + \sum_{t \in T} \Delta A_t + k_{old},\end{aligned}$$

where $\Delta EST_M(i,j)$ gives the benefit of *moving* a start or end data unit of the access requirement i to position j. Note that each of ΔA_t is negative. Hence the benefit of moving might be more or less than the benefit of copying depending on the relative values of $\sum_{t \in T} \Delta A_t$ and k_{old}. But the main advantage of moving instead of copying is that this operation does not increase the redundancy thus it keeps the storage requirement the same. So we perform moving instead of copying as long as $\Delta EST_M(i,j)$ is positive.

Data Unit processing order: We now need to figure out how to use this information to decide in what order the copying and moving should be done. We will make two heaps: E_M and E_C. The E_M heap will organize the move operations and consist of the values of $\Delta EST_M(i,j)$ for the start and end data units for all access requirements i where the units are put in their optimal location j. The E_C heap will be the same ~~thing~~ except it will organize the copy operations and consist of the values of $\Delta EST_C(i,j)$.

We process the E_M heap first as long as the top of the heap is positive and effect the move of the data unit at the top of the heap. After each removal and processing, ΔEST_M and ΔEST_C of the affected access requirements and the corresponding heaps are updated. If there are no more data units where ΔEST_M is positive, then one element from the top of the heap E_C is processed. After processing and copying a data unit from the top of heap E_C, the heaps E_C and E_M are again updated with new values for the affected access requirements. If this introduces an element in the top of E_M heap with positive values, the E_M heap is processed again. This process gets repeated until the user defined bound on redundancy factor is reached. As a summary, the pseudo-code of this algorithm is shown ~~as~~ Algorithm 1.

```
Input: Data units and their access requirements (AR) ;
for start and end unit of each AR i do
    Find optimal location j for copy;
    Calculate ΔEST_M(i, j) and insert into E_M ;
    Calculate ΔEST_C(i, j) and insert into E_C ;
end
while true do
    while top element of E_M is positive do
        Pop top element and move the data unit to its destination ;
        Update E_M and E_C ;
    end
    if there is more space for redundancy then
        Pop top element and copy the data unit to its destination ;
        Update E_M and E_C ;
    else
        break
    end
end
```

Algorithm 1: Pseudo-code for our algorithm

4 Complexity Analysis

We now analyze the running time and storage requirements of our algorithm. Let N be the number of data units and A be the number of access requirements. We will use m as the average span of a single access requirement. Let r be the redundancy factor limit specified by the user so that $O(rN)$ units can be copied. For the sake of analysis each data unit will be used by $O(A)$ access requirements and at each location there will be $O(A)$ access requirements whose span overlaps it.

Time Complexity: The construction of the heaps E_M and E_C involves computing the benefit information for all A access requirements and inserting each one into the heap. For a single access requirement, computing the benefit information of moving or copying one of its data units involves scanning each data unit in its span. This approach takes $O(m)$ operations. Calculating $\sum_{s \in S} \Delta A_s$ and $\sum_{t \in T} \Delta A_t$ will take $O(A)$ operations since there are $O(A)$ access requirements to potentially have to sum over. Inserting this benefit information into the heap takes $O(log(A))$ operations. In total then it takes $O(m + A + logA)$ or $O(m + A)$ operations per access requirement to get the benefit information. The initial construction thus takes $O(A(m + A))$ operations.

After the initial construction, the move and copy loops are executed. In every iteration of move or copy, an element from the top of the heap is removed and processed, the benefit function is recalculated for affected access requirements, and the heap is updated. There are potentially $O(A)$ overlapping access requirements whose benefit information needs to be recalculated. As shown above, for each of these access requirements $O(m + A)$ operations are required to perform the recalculation and update the heap. Each iteration of move or copy thus takes a total of $O(A(m + A))$ operations.

For simplicity we will assume that the move loop runs $O(N)$ times total. This comes from the fact that the cache oblivious layout [Yoon et al. 2005] should be a good approximation so the number of moves that would be useful should be limited. There are $O(rN)$ copies made so there are that many iterations of the copy loop. We thus can assert that there are $O(rN + N)$ iterations of the move or copy loops. We can simplify this to $O(rN)$ operations since $r \geq 1$. In total then the moving and copying loops will take $O(rNA(m + A))$ operations, which is also the running time for the whole algorithm.

Space Complexity: During the run of the algorithm, we have to store the number of overlapping spans at each data unit, which will require $O(N)$ storage. We will also have to store a heap of access requirements, which can be stored using $O(A)$ space. We also have a list of access requirements and that information will take up $O(A)$ space. In total we thus have $O(A + N)$ storage space used during the run of the algorithm.

5 Experimental Results

Experiment context: In order to implement our algorithm, we used a workstation that is a Dell T5400 PC with Intel (R) Core (TM) 2 Quad and $8GB$ main memory. The hard drive is a 1TB Seagate Barracuda with 7200 RPM and the graphics card is an nVIDIA Geforce GTX 260 with 896 MB GPU memory. The data