6.2 Multitasking

Running the initial tests demonstrated an important performance problem with virtual memory management of files. When the first out-of-core tests were run, the X-windows system became very sluggish and it took a long time for applications to respond. The longer the execution time of the memory mapped code, the worse the problem became. When the Comanche tests were run, the problem did not arise; disk access was slow since the application was moving large amounts of data, but the X-windows system responded quickly as did the other applications.

A test was constructed to measure this behavior. Two applications were executed simultaneously on four data files (see Table 2). The ratio of the VMM to Comanche execution performance was more than doubled between the single out-of-core test and the multitasking out-of-core test. This means that system performance degrades much more rapidly for VMM than Comanche as more and more out-of-core applications are executed.

This illustrates our main issue with virtual memory management—its lack of information about the application’s actual memory needs and access patterns. As the application starts faulting in pages, the VMM system will take pages away from other applications even though those pages will only be used once and then discarded. Comanche uses the source code to determine the maximum number of rows it needs at any given moment in time; this becomes the working set for the application. For example, an eighty megabyte file representing a 2D grid needed only one megabyte of physical memory to store its row data.

6.3 Optimizations for window2

The problem with window2 was difficult to diagnose. System performance showed that our initial version was 80% slower than VMM. While not terribly worse, it is unacceptable according to our original goals. Simulation showed that while latency was less for our model (Table 3), the volume of data transferred (Table 4) was one and half times greater for our model than for VMM.

The problem stems from the simple way the code generator inserts attach operations before the inner loop and release operations after the inner loop. There are at most three attached rows at any given time. The leading array reference (A[i+nwin]) would release an array that would shortly be attached by the middle reference (A[i]). But, since an unattached array can be the next victim, arrays were being swapped out even though they were to be used in a few short iterations (see Figure 11 (A)).

The optimization is based on a seeker/reaper paradigm. A seeker follows the leading reference (A[i+nwin] in this case) in a loop and inserts an extra attach operation to the same row that the leading reference attaches. Without a corresponding release, the buffer pool would quickly get filled and the system would halt due to lack of resources. A reaper is used to avoid this by inserting a release operation at the same point that the trailing reference (A[i-nwin]) is placed. This pairs the two correctly with the seeker attaching rows some number of iterations ahead of the reaper’s release (see Figure 11 (B)).

In our case, the seeker/reaper code was inserted by hand and the simulations were repeated for window2. The results are in Table 5. As can be seen, the performance in the after case is much more in line with our performance goals than before.
7 Conclusions

The primary contribution of this research is a compiler and runtime system model that provides efficient out-of-core programming without virtual memory management. The system performs better than VMM in all of the test cases and uses significantly less system resources to do so. We demonstrated that such a system can be built by implementing key components on top of established compiler technology.

A novel contribution is the compiler managed cache approach that uses attach and release directives generated by the compiler to guide the runtime system on the management of resources. This division of responsibility greatly simplifies the overall model while still supporting high performance. This approach proved successful as demonstrated by the prototype.

References


