written. If these flags are not set, the runtime system must write the arrays back to disk when they are released. This illustrates the cooperation between the compiler and the runtime system to achieve a simple but efficient solution. It is this clear division of responsibilities that allows us to control random alias proliferation and centralize alias resolution at the level of attach and release directives.

3.3 Array Folding

Folding is needed for very large single dimensional arrays. Higher dimensional grids are unlikely to need folding. As illustration, consider the code example in Figure 7. If it is not possible to store a complete row of A, we can fold the array and apply strip mining (Figure 8). In general, a reference $A[i][j]$ is replaced by $A[i][j1][j2]$ where $j1$ is equal to $j/B$ and $j2$ is equal to $j \mod B$ and each reference to $j$ is replaced by $j1*B+j2$. The catch here is that we may not be able to remove the computation of $j1$ and $j2$ from the inner loop. If there are few statements in the loop, such as in the example, we have the same problem as before.

For the case of a very small outer dimension, we can apply the reverse of the previous transformation called loop collapsing [3]. For example the code in Figure 9 results in the code in Figure 10.

4 Experiments

A prototype runtime system and code generator have been built as a back end for the SUIF compiler system. The test suites were translated directly into UHF and code was generated from this format. A runtime system was constructed for storing two dimensional arrays using the standard C 1/0 library. The runtime system supports APIs to map and unmap files of arbitrary two dimensional data.

Data files are constructed and filled with random values on the interval $(-1, 1)$. The system used for testing actual performance has 64MB of memory of which around 50MB are available to the program. Virtual memory is tested by mapping files into memory. This is absolutely necessary, since swap space can be filled easily when running large tests. Elapsed times were recorded on an unloaded machine and repeated to see that they did not deviate too greatly between test runs.

It was quickly determined during initial testing that memory mapping large files has a significant impact on system resources. An additional test was added to compare the impact of multitasking on selected applications. The test suite was also executed under the MemStat virtual memory simulator (see[5]) and the results were compared to data collected from an instrumented version of the runtime system. These comparisons helped in constructing the custom optimizations presented in the last section of this chapter.

5 Test Suite

The test suite was constructed to provide a uniform mechanism for specifying the total memory accessed. Benchmarks were not considered since varying array dimensions can have an adverse
effect on numerical stability. Inputs take a command line argument for total memory consumption which is used to compute actual array dimensions. For example, stencil computes the dimension of the 2D grids with the following code

\[ n = \text{(int)} \quad \text{sqrt(count / (3 * sizeof(double)))}; \]

(count is the total memory to be accessed). This code uses three 2D grids of double precision values.

Two dimensional data on disk is assumed to be in row major order. While block mappings may perform better, we opted for simplicity based on the reasoning that large data sets will, in general, already exist in a row major or column major order. Also, we want to constrain the number of possibilities to ease detailed exploration of access patterns and for correlation purposes.

A detailed presentation of the test suite is in [4] with source code and diagrams for the more unusual access patterns. We provide a short list here.

- **stats** uses a one dimensional vector with sequential read access such as would be found in a statistics collection program.
- **window1** uses a one dimensional vector with a window of neighbors similar to a finite difference computation in one dimension.
- **window2** accesses a two dimensional grid with a window of neighbors in the two dimensions. It is similar to SOR when the window size is one.
- **stencil** accesses three two dimensional grids in a stencil pattern like that found in the 2D wave equation.
- **matvec** is a standard matrix vector product.
- **matmul** is matrix multiplication using a transposed "B" matrix.

## 6 Systems Performance

First, we look at the performance of the test suite executing on a standard Unix workstation. The workstation is a PentiumPro 200Mhz processors with 64Mb of RAM running the Linux operating system. Tests were performed on an unloaded system without the X-windows server; only the application(s) and basic system services were running. Problem parameters were adjusted to use 80Mb of data. Codes were tested first by comparing output files on several input test cases.

### 6.1 Out-of-Core

The first test was on simple out-of-core performance for a single application. The results of this test are in Table 1. Times are measured using \texttt{hours:minutes:seconds} notation. The ratio represents the time of the virtual memory version over the time of the Comanche version. Values greater than one favor Comanche and values less than one favor VMM. All but one of the test cases executed faster under Comanche than under VMM. The exception is window1 which runs 20% slower. The optimization for this case is discussed in Section 6.3.