Haverford College Computer Science Tech Report 2011-01
Haverford College, Haverford, PA 19041
Archived 2011-10-14

(With minor revisions from 2011-08-05)

Experiences with expressing and optimizing dense numerical algorithms in AlphaZ

Micah John Walter, Haverford College
David G. Wonnacott, Haverford College

Abstract

As multi-core processor architectures and parallel programming paradigms become more and more commonplace, there are those who concentrate on producing parallel code by hand and those who advocate the development of tools to parallelize code automatically. The AlphaZ system represents a middle road: the user provides a high-level description to the system about how the parallelization of the problem is to be accomplished, while the system generates the actual C code. In our experiences in using the AlphaZ system to express time-skewing of dense matrix codes that are not 'embarrassingly parallel', such as Jacobi stencils, we achieved qualified success in producing speedup across a cluster of computers. We are optimistic about the future use of AlphaZ in similar projects.

Table of Contents

The AlphaZ system

The AlphaZ* system (`http://www.cs.colostate.edu/AlphaZ/wiki/doku.php`) is a tool that generates code for mathematical algorithms using the *polyhedral model*. What was interesting for our research was that, using this model, AlphaZ can generate parallel code as well as perform other optimizations traditionally done by a compiler. Unlike other tools (such as the Pluto compiler), AlphaZ generates this code not using existing C code, but rather from a series of equations written in a simple pure-functional programming language specially designed for the system, called Alphabets. (The AlphaZ website calls this method the *polyhedral equational model*, since it implements the polyhedral model using pure mathematical equations as input.) In addition to the equational Alphabets file, AlphaZ also takes a *schedule* as input; this schedule file contains specifications on how AlphaZ should handle statement ordering and memory mapping, as well as giving the option of enabling automatic *tiling* (or parallelization) of the code. The schedule also contains the path of the Alphabets file (or files) that it is using; it is the schedule that is compiled by the user. (To compile the schedule, the user right-clicks the schedule file in the Eclipse Project Explorer and selects Run As → Compiler Script.)

The easiest way for us to install AlphaZ was to use a separate Eclipse installation provided to us by one of the developers of the system. The files that compile the schedule are located in the `eclipse/dropins/dropins/plugins` folder; the most recent version can be simply copied (or 'dropped') into this folder, while the Eclipse installation is not open and after the old version has been removed. Builds of the system are archived nightly; the most recent version is available from the CSU website at `http://www.cs.colostate.edu/AlphaZ/AlphaZ-update.tgz`.

AlphaZ initializes arrays by filling them with the value `1`. The output of a plain AlphaZ program is simply the time it took the entire function to execute; in order to compare performance of AlphaZ runs against ones using different problem sizes and ones using other tools, we used a script that modified the AlphaZ-generated C code to compute and print the approximate number of MFLOPS (millions of floating-point operations per second) achieved† (see page 118). To test the correctness of AlphaZ-generated code, we used the `check` option in the AlphaZ Makefile, which took parameters with user input at runtime and printed out the results rather than the execution time.

Experimental setup

The hardware used in our experiments consisted of the same computer lab as was used by Tim Douglas for his thesis work [Dou11]:

* The name *AlphaZ* is pronounced [æl'faz], with the accent on the second syllable.

† Our method of calculating the number of MFLOPS achieved is approximate because although in our Jacobi stencils we do not change the values of the edges, the total size of the array (rather than the length less 2) is used for convenience' sake. This normally does not affect our benchmark; however, the potential does exist, especially for experimental problems with extremely small parameters.

Each of the machines is powered by an Intel Core i7 860 microprocessor running at 2.8 GHz, with 4 GB of DDR3 memory. The processors are capable of hyper-threading, i.e., running multiple threads on one logical core, but that was disabled for our benchmarks. The cache payout is as follows (L1 and L2 are per-core, and L3 is shared between all cores):

Level	Size	Associativity	Line Size
L1 data	4 × 32 KB	8-way set associative	64 byte
L1 instruction	4 × 32 KB	4-way set associative	64 byte
L2	4 × 256 KB	8-way set associative	64 byte
L3	8 MB	16-way set associative	64 byte

The machines are connected via a 10/100 Ethernet switch administered by Haverford College.

We did many preliminary runs using Ubuntu 10.04 LTS, since that was our primary developing environment; however, because of our research needs (the latest Linux kernel that Cluster OpenMP supports is 2.6.24), we performed our research tests using a modified version 4.0 of Debian GNU/Linux.* Similarly, although we made use of `gcc` when using Ubuntu (since we have `icc` installed only on Debian on our research cluster), our actual research results were achieved using version 10.1.022 of `icc`.

Regarding compiler optimization, we found that even with `icc`, performance was best when the optimization flag `-O3` was used. Our Jacobi stencils also required that we use the compiler flag `-lm`.

In our research, we used essentially three different benchmarks in testing the performance of the AlphaZ system: two Jacobi stencils (one-dimensional and two-dimensional), and a modified version of the Tomcatv benchmark from the SPEC95 benchmark suite. The stencils involved many dependencies across time-steps; as a result, time-skewing was necessary, although this was still not enough to give us good performance. The Tomcatv-like benchmark, on the other hand, is highly vectorizable, but the complex computations needed at each time-step made the algorithm difficult to implement using the AlphaZ system.

The problem sizes that we used covered a range from as small to as large as possible, given the memory limitations of our machines. In general, we reduced the number of time-steps as the array size increased so that our run times would be manageable; however, for our Cluster OpenMP runs, we increased both array size and number of time-steps, resulting in realistically long run times as well as heightened efficiency.

Memory allocation and access with AlphaZ

AlphaZ-generated code allocates memory using `malloc()`. As contrasted with other methods of allocating memory, `malloc()` has its advantages and disadvantages. On the positive side, it allows the user to specify the parameters of the problem at runtime;

* The kernel version of our installation was `2.6.24-etchnhalf.1-amd64`.

however, it is a more inefficient method of allocating memory than some other methods, such as static arrays. For example, when testing our Tomcatv-like program (see page 12), we achieved speedup of a factor of 2.5 over AlphaZ output merely by changing the method of memory allocation. In addition to its inefficient memory allocation, AlphaZ's memory access uses multiple levels of pointers; to our knowledge, there is no advantage in using such a tree of pointers over a more straightforward implementation of `malloc()`. With both memory allocation and memory access, It remains a possibility that future versions of AlphaZ will implement one or more of these improvements.

Part of the information that is in the schedule file for an AlphaZ program is memory mapping. Without this memory mapping, the system would continue to store all values ever assigned to all variables whether they were needed or not; this kind of 'memory haemorrhaging' would, of course, make any kind of performance for reasonable problem sizes unachievable. An example of this memory mapping is below:

```
setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i,j -> t,i,j)", "(2,0,0)");
setMemoryMap(program, system, "Final", "Final",
    "(t,i,j -> t,i,j)", "(1,0,0)");
```

In this first statement, the user is indicating that as the array `Intermediate` is updated, all values in the dimensions `i` and `j` need to be saved, while in the `t`-dimension, only the immediately previous values needs to be saved for future reference. In the array `Final`, only one value is needed in the `t`-dimension.

AlphaZ implements this mapping using a `MOD` function. Since the `%` function is ambiguously defined in the C language, early versions of AlphaZ defined this `MOD` function for negative values as well:

```
#define MOD(i,j)  ( (i)>=0 ? (i)%(j) : (j-1)-((-(i))%(j)) )
```

However, since AlphaZ-generated code is guaranteed not to take the modulus of a negative number, this definition is unnecessary.* We requested that this function be updated; newer versions of AlphaZ use a different definition (which is in fact simply the C definition) and experience a corresponding improvement in performance.

If only one value of the array needs to be kept in any dimension, the following syntax can be used:

```
setMemoryMap(program, system, "A", "A", "(t,i,j -> )");
```

This produces an 'array' of zero dimensions in the resulting generated code.

For our Jacobi stencils, we allocated our `Final` array as follows:

```
setSpaceTimeMap(program, system, "Final", "(i -> T,i)");
```

* In fact, the definition given here is not only unnecessary, but also incorrect. Nevertheless, even a definition that matched the true mathematical definition would result in slowed performance.

Note that there is only one dimension in the array. However, AlphaZ mandates that there must be the same number of dimensions in all arrays (so that it is able to order them properly); therefore, we added the constant 'dimension' `T`, which is the last time-step of the `Intermediate` array in the same schedule. In the resulting generated code, misleading variables are used in the memory macros and `#define` statements (see, for example, `Final(i,i1)` rather than `Final(t,i)` on page 36); nevertheless, the program functions correctly.

Performance of AlphaZ-generated code

Sequential (untiled) code generated by AlphaZ runs at speeds comparable to, but somewhat slower than, analogous hand-written code. As mentioned above, changing the method of memory allocation often results in performance increases.

For our one-dimensional Jacobi stencil, the following were performance benchmarks we achieved.

	T = 10000000, N = 1000	T = 10000, N = 1000000
Hand-written code, local pointers*	4532 MFLOPS	1949 MFLOPS
AlphaZ-generated code, modified allocation	3893 MFLOPS	1869 MFLOPS
Hand-written code, standard version	4253 MFLOPS	1338 MFLOPS
AlphaZ-generated code, using two arrays†	4360 MFLOPS	1118 MFLOPS
AlphaZ-generated code, standard version	3429 MFLOPS	1713 MFLOPS

For small problem sizes, AlphaZ's `oneD` performs worse than the equivalent `oneD-copy`, which uses a whole second array to save values from the previous time-step rather than using modulus. However, our preliminary tests seemed to indicate that the performance of this version of AlphaZ code did not perform well when tiled (see page 9); many times execution was halted due to a segmentation fault.

As can also be seen in the table above, altering the memory allocation in the AlphaZ-generated code gives us some performance improvement. Although it does not bring us quite up to the speeds achieved by our highest-performing hand-written code, the modified version shows promise; for large problems, it is very close to the speeds achieved by the fastest-running hand-written code.

* Changing these local pointers to global pointers (see comment in hand-written code, page 17) results in a drastic drop in performance.

† This benchmark was obtained using unmodified AlphaZ-generated code; if we modify the memory allocation as we did for the standard AlphaZ version, we get performance of 4867 MFLOPS using static arrays for the small problem (10000000 time-steps and 1000 array size).

In both one-dimensional and two-dimensional stencils, our untiled AlphaZ-generated code gives better performance than the 'standard' hand-written C code for large problem sizes (this is the C code that we use for other research systems, such as the Pluto compiler). Like the standard hand-written code, however, the AlphaZ-generated code using two arrays gives worse performance for large problem sizes as well. The cause of this can be explained by the different patterns of memory reference. The standard hand-written codes, as well as the two-array version made with AlphaZ, compute the new values and store the results in a second array, after which those values are copied into the first array again. The more efficient methods, on the other hand, after reading from one array and storing the results of the computation in the other, treat the second array as the array to be read from, while the first takes the place of the locus of the results of the computation; thus the unnecessary copy operation is done away with.

AlphaZ performs much better with equations that involve division by 4 than those involving division by 3. More specifically, the following code, which involves division by 4 —

```
Intermediate[t,i] = case
    {| t>0 && 0<i<N-1} :
        (2*Intermediate[t-1,i] + Intermediate[t-1,i-1] +
        Intermediate[t-1,i+1]) / 4.0;
esac;
```

— achieves higher performance than the following analogous code using division by 3:

```
Intermediate[t,i] = case
    {| t>0 && 0<i<N-1} :
        (Intermediate[t-1,i] + Intermediate[t-1,i-1] +
        Intermediate[t-1,i+1]) / 3.0;
esac;
```

So far, this is unexplained to us, since the performance of similar code not generated by AlphaZ does not suffer to nearly the same extent with the change in divisor.

A similar, but different, issue we encountered was that of subnormal numbers.* Although, as mentioned above, a divisor of 4 is under normal conditions more efficient than a divisor of 3, having a divisor of 4 without doubling the weight of one of the terms to be averaged results in much slower performance. The reason for this is that under the conditions described above, the value of each element in the array approaches zero; as a result, the processor must do more work in accurately handling these numbers.

In the algorithms that we used with AlphaZ, using `double` ran somewhat slower than using `float`; however, we used `double` throughout all our experiments, since it is our research standard and because our other tests make use of `double`.

* *Subnormal numbers* (also called *denormal numbers*) are numbers that, though not equal to zero, have a low enough absolute value that they fall outside the capabilities of the system's ordinary system of representation, since the exponent of the floating-point representation is farther below zero than is possible. These numbers cause more work for the processor, thus reducing performance.

Optimizing persistent initial values

As Alphabets files are purely functional, certain commands taken for granted in imperative programming are absent. One example of this is actually what might be called the *lack* of a command. Certain algorithms — including all the algorithms we are tiling using AlphaZ — update values across an array with the exception of the first and last elements. In the imperative programming paradigm, the fact that these values are not updated means that some time is saved during each pass through the array. AlphaZ, on the other hand, demands that each point on the array be defined for every iteration, or 'time-step', that the array is updated. The result of this is that the edges are redefined as the same initial value an excessive number of times.

Below is an example of Alphabets code defining how an array is to be updated:

```
X[t,j,i] = case
    {| t==1} || {| j==1} || {| i==1} || {| j==N} || {| i==N} :
        X0[j,i];
    {| 2<=t<=ITACT && 2<=j<=N-1 && 2<=i<=N-1} :
        X[t-1,j,i] + RX2[t,j,i];
esac;
```

This code excerpt essentially communicates the following information: in the first iteration (i.e. when `t` = 1), or at the first or last element in either of the two dimensions in the matrix, `X` should be equal to that given originally as the parameter `X0`; otherwise, it will be updated based on itself and another matrix, `RX2`. While an imperative programmer would omit the redundant definition at each iteration, AlphaZ includes it in each time-step. These redundant definitions are not nested in loops as deeply as other code, so that actual performance is not usually significantly decreased, but it is not the case that the compiler takes care of these redundant definitions automatically.

For example, Tomcatv, running with the above code with a problem size of 1000000 iterations and a tiny 4 × 4 square matrix (one that, in fact, has more edges than non-edges), performed at 4816 MFLOPS on `gcc`. When the code was changed to the following code, which copies the redundant value from the previous iteration rather than from another array entirely, the performance increased by 21 %, giving a performance of 5822 MFLOPS:

```
X[t,j,i] = case
    {| t==1} :
        X0[j,i];
    {| j==1} || {| i==1} || {| j==N} || {| i==N} :
        X[t-1,j,i];
    {| 2<=t<=ITACT && 2<=j<=N-1 && 2<=i<=N-1} :
        X[t-1,j,i] + RX2[t,j,i];
esac;
```

However, even this does not give the same performance as hand-optimized code, which does not include any redundant assignments at all; using a simple alteration of the preprocessor definitions, we achieved 6256 MFLOPS.

Despite this minor missed performance enhancement, AlphaZ is generally quite good

at not generating redundant code. We were made keenly aware of this feature when things accidentally went awry in one version of the AlphaZ system: instead of putting the initialization (when `t` = 0) in its own block before the loop that iterated all the other values of `t`, it used a `?:` operation throughout the loop, checking the current value of `t` at each time-step to decide whether it should initialize or continue with the other operations. This constant checking naturally resulted in a significant decrease in performance.

Running tiled code sequentially

The performance of tiled code generated by AlphaZ is, oddly, poor for small array sizes. This is the case in all of our benchmarks. For example, in our one-dimensional stencil, an array size of 10000 (`T` = 1000000) gives us 2714 MFLOPS; an array size of 1000 (`T` = 10000000) gives us only 126 MFLOPS; and finally, an array size of 100 (`T` = 100000000) achieves little better than 1 MFLOPS. However, this shortcoming of AlphaZ is not a significant problem in practice, since tiling (especially when it is run in parallel) is not necessary for such problem sizes; even under good conditions, it still would not be expected to give significant improvements over untiled code.

Like the one-dimensional stencil, the untiled two-dimensional stencil drops off in performance as the size of the arrays increases; tiling the stencil increases memory locality, and so retains high-speed performance for large problem sizes. For example, our tiled one-dimensional stencil achieved 3406 MFLOPS on a problem size of `T` = 10000, `N` = 1000000 — faster than the corresponding untiled code by a factor of 3.2. Tiling our two-dimensional stencil allowed us to similarly maintain performance; with a very large problem size (`T` = 1000, `X` = 10000, `Y` = 10000) we achieved 3385 MFLOPS, which was 24 % faster than the corresponding untiled code.

Unlike the one-dimensional stencil, however, the two-dimensional problem does not give good performance when parallelized with AlphaZ. We suspect that this is due to AlphaZ's current lack of multi-level tiling; such tiling would allow a problem to first be broken up into chunks to feed each processor core or node, while these chunks would in turn be distributed into smaller pieces, thus increasing spatial locality. (Because AlphaZ uses pure affine functions — one needs to use the built-in tiling function — it is not possible to manually provide multi-level tiling.)

It is worth noting that, in our tests using the current release of AlphaZ, using the option `setTiledCGOptionOptimize()` often results in better performance for tiled code than not using the option, if the schedule was able to compile with the option at all (in some schedules, using the option would not work).

Regarding tile sizes, we found that the ideal tile size for our one-dimensional problem was 1250 in both the `T` and `N` dimensions; for our two-dimensional problem, we achieved our highest speeds using tiling of `T` = 48, `X` = 30, `Y` = 42.

Running tiled code in parallel

The parallel code generator for the most recent version of AlphaZ we tested (the version from 2011-07-29) does not work; the code compiles, but the run times are extremely slow. However, using an old version of the AlphaZ system (2011-06-24), we achieved 71 % of linear speedup on a single node (4 cores) on our one-dimensional stencil over the corresponding tiled but single-core run (`T` = 100000, `N` = 1000000). We reached 80 % of linear speedup with the slightly larger problem, which had an array size 10 times larger. (This is the largest order of magnitude that fits into our machines' memory.) However, if we use an even greater number of time-steps, performance drops again. This drop when the number of time-steps is large does not occur with our corresponding non-parallel tiled code; we suspect that the cause of this drop has to do with the inefficiency of the method of parcelling out work, as it appears that once a computer is assigned a portion of the problem, it does not receive any new work even when it has finished its first job.

Running AlphaZ-generated code (or any code) in parallel using OpenMP requires the use of special compile flags: if `gcc` is being used, the flag `-fopenmp` is needed, while for `icc`, `-openmp` is needed. By default, AlphaZ mentions `gcc` in the `Makefile` it generates; we used our script to change the compiler name as well as the flags mentioned, depending on the system we were running the compiler on.

Although Cluster OpenMP is not officially supported by the AlphaZ development team, we were able to get our AlphaZ-generated parallel code to run across multiple nodes in a cluster by merging the function file (e.g. `oneD.c`) and the wrapper (e.g. `oneD-wrapper.c`). This merge was performed by another script. (Cluster OpenMP also requires the use of `kmp_sharable_malloc()` rather than `malloc()` in the memory allocation; this modification is taken care of by the first script.) The `generateMerged.py` script produces a file ending in `-merged.c`, which can then be compiled without `Makefile` using a command such as the following (an example from our runs):

```
/opt/intel/cce/10.1.022/bin/icc oneD-merged.c -O3 -lm -cluster-openmp
-o oneD-merged
```

(We tried using the `-static` flag when compiling, since we do not have Cluster OpenMP or `icc` installed on all our machines, but an error resulted, saying that `dlopen` requires at runtime the shared libraries from the `glibc` version used for linking.)

In addition, there needs to be a file called `kmp_cluster.ini` in the same directory as the parallel code to be executed using Cluster OpenMP, containing a single line such as the following:

```
--process_threads=6 --processes=10 --hostfile=mpd-8.hosts
--startup_timeout=3600 --launch=ssh --sharable_heap=1000M
```

(Note that although this is printed on two lines for practical reasons, it is imperative that this appear as a single line in the actual file.) A `hostfile` should be included in the same directory that lists the names of the machines, one on a line, with the host computer's name as the first line; the name of this file is arbitrary, although we have

chosen a filename that contains the number of machines listed. The `--process-threads` parameter should be followed by a number not less than the number of cores on the processor; we used 6, which was two more than the number of cores on our processors, to avoid issues of load imbalance. Similarly, the `--processes` parameter would ideally be identical to the number of nodes listed in the `hostfile`; however, when this was the case during our runs, some of the machines finished their work faster and sat idle while the other machines were still working. Increasing this number, then, resulted in significant performance increases in at least one large run using Cluster OpenMP (the last run listed in the table below).

One last thing necessary for AlphaZ-generated code to run using Cluster OpenMP is to set the environment variable `LD_LIBRARY_PATH` appropriately (in our instance, we set it to the path `/opt/intel/cce/10.1.022/lib`).

The following table demonstrates our performance achieved for the one-dimensional stencil problem run in parallel using AlphaZ-generated code. For the unparallelized benchmarks, the time-step values were chosen so that the run would take about a half-minute to complete; using different time-step values resulted in very similar performance. For the parallel benchmarks, we recorded some of our best runs, although runs on different values in the same range do give similar speedup. Due to the nature of our code, our best runs were often those with the largest number of time-steps. However, it will be noted that the parallel code using static arrays experienced a drop in performance with 8000000 as opposed to 100000 time-steps; the exact reason for this is currently unknown to us.

	Total performance	Performance per core
Unmodified tiling `T` = 1000, `N` = 10000000	3369 MFLOPS	3369 MFLOPS
Tiling with static arrays `T` = 100, `N` = 10000000	3376 MFLOPS	3376 MFLOPS
Unmodified parallel `T` = 100000, `N` = 10000000 4 cores on 1 node	4348 MFLOPS	1087 MFLOPS
Unmodified parallel `T` = 1000000, `N` = 1000000 32 cores on 8 nodes	9935 MFLOPS	310 MFLOPS
Parallel with static arrays `T` = 100000, `N` = 10000000 4 cores on 1 node	10889 MFLOPS	2722 MFLOPS
Parallel with static arrays `T` = 100000, `N` = 8000000 32 cores on 8 nodes	40626 MFLOPS	1269 MFLOPS
Parallel with static arrays `T` = 8000000, `N` = 8000000 32 cores on 8 nodes	35294 MFLOPS	1102 MFLOPS

The above table demonstrates the ability of AlphaZ to generate code that, when modified, gives significant performance increases when run on a multi-core machine. Furthermore, we can continue to get significant performance increases over a cluster; while the increase is not as dramatic as that achieved by running code in parallel on a single node, it is still significant. The improvement in performance caused by the change in memory allocation shows that there is the potential for improvement in the AlphaZ system within a more-or-less easy reach.

We have found that when we run on a cluster using Cluster OpenMP, our problem size is still limited by the amount of memory available on the host machine. We hope that when the AlphaZ system makes the transition to MPI we will be able to make use of the memory on the entire cluster for the initial data set.

The Tomcatv benchmark

The Tomcatv benchmark (`http://www.spec.org/cpu95/CFP95/101.tomcatv/`) is a vectorized mesh generation program that is part of the SPEC95 benchmark suite; it makes use of many loops and temporary variables to calculate residuals, but, unlike the Jacobi stencils, it does not have many dependencies in dimensions other than the time dimension. We used `f2c` to compile a modified version of the Tomcatv code; this modification changed the initialization to fill the arrays with `1` (to match AlphaZ's initialization) and removes the 'break' from the program to make it more compatible with the polyhedral equational model. This Tomcatv-like program was run with parameters as constants in the code; the code was recompiled each time we changed the parameters.*

Our first working Tomcatv program made using AlphaZ used significantly fewer arrays for temporary variables than did the original. This resulted in excessive redundancy in many of the variable definitions (see page 73 and following), which the compiler was apparently not able to efficiently evaluate. We hypothesized that the code for Tomcatv generated by AlphaZ would be significantly faster if we replaced the redundant code with external functions. This was not the case, however; with the exception of a few `pow()` functions, the resulting source code ran even more slowly than the redundant code.

When compiled using the standard options in AlphaZ, this specific version of Tomcatv crashes the compiler; to remedy this problem, we needed to use the deprecated option `setCGOptionDisableNormalize_depreciated()`. Since this option turns off the normalization of reductions, it reduces performance; hence, this option is only a temporary workaround that will be removed in future versions of AlphaZ. In an effort to make a version of the program that would not crash the compiler with this option turned off, we rewrote the Alphabets definitions for the two arrays that used AlphaZ's built-in `reduce()` function so that they would not have to use that function. Rather than compiling without crashing, however, it still required the option; its performance was

* Hereafter the term *Tomcatv* will be used for the more precise *Tomcatv-like*; none of our tests made use of the original source code unchanged.

somewhat slower than the version using AlphaZ reductions.

We later completed another translation of the benchmark into AlphaZ source code, this time using all the temporary variables that were used in the original Fortran code; we termed this version the ‘obvious’ version of our code, since it more obviously paralleled the Fortran. We got this version of the program to compile only after substituting constant dimensions rather than using AlphaZ’s statement ordering function (see the difference in syntax between setting dimensions as ordering on page 32 and the use of the `setStatementOrdering()` function on page 35). However, even though the code inside the loops was quite similar, the AlphaZ-generated version ran slower by more than a factor of 10.

We tested the performance of each version of the code with each inner loop eliminated; each such shortening of the amount of work to be done resulted in a decrease in run times approximately proportional to the total run times of the two versions of code.* We thus concluded that it was not any one block in particular that was slower in the AlphaZ-generated code, but rather it was something more general that affected the entire program.

We then made an abbreviated version of the code — dubbed *Tomkitten*† — which contained only one of the loop blocks that displayed a performance difference. Our initial observations of this heavily abbreviated code seemed to indicate that a partial cause of the performance difference was the difference between the Fortran and C conventions for storing multi-dimensional arrays linearly: while Fortran uses a column-major order, C uses a row-major order. It does indeed appear to be the case that using `[j,i]` rather than `[i,j]` in AlphaZ generates somewhat faster-running code; however, in this case, the drastic performance difference seems to have been caused by the inconsistency in our own code, since we happened to be using both conventions simultaneously in the schedule’s target mapping:

```
setSpaceTimeMap(program, system, "XX", "(t,i,j -> 0,t,0,j,0,i,0)");
```

Note that the `i` and `j` dimensions are reversed on the right-hand side in the equation above. Although the code generated by this schedule is correct, it makes very inefficient use of locality, and performance plummets.

As mentioned above, memory allocation played a large rôle in the performance of this benchmark. Below is a summary of the performance achieved with different versions of the code (the AlphaZ code listed is the ‘obvious’ version):

* The exception to this was the code block containing the reductions; this block was actually faster on the AlphaZ-generated code than on the original Fortran code.

† Note the typographical distinction between *Tomkitten* (referring to abbreviated benchmark code) and *Tom Kitten* (a protagonist in the œuvre of Beatrix Potter).

	ITACT = 1000, N = 1000
Hand-written code, using `f2c`	3301 MFLOPS
AlphaZ-generated code	962 MFLOPS
AlphaZ code, modified `malloc()`	1199 MFLOPS
AlphaZ code, static array allocation	2428 MFLOPS

In the case of this particular benchmark, it appears that the memory allocation used in the code automatically generated by AlphaZ, which includes the use of triple pointers, takes a toll on performance; our modification memory allocation using `malloc()`, while still allowing for user-specified parameters at runtime, gives a speedup of 25 %. If we use static arrays rather than `malloc()`, however, we get a performance increase of a factor of 2.5 over the original AlphaZ-generated code. One possibility for this increase in performance is the compiler's vectorization of the loops: for the unmodified AlphaZ-generated code, `icc` gives no compiler messages about loop vectorization, while it returns 8 messages with the words `LOOP WAS VECTORIZED` when compiling the `f2c` code, and 22 such messages for our memory allocation with static arrays.

As was previously mentioned, AlphaZ will sometimes allocate an array of zero dimensions; this appears not to be as efficient for the compiler as actually using a scalar variable. Although the use of scalar variables instead of these zero-dimensional 'arrays' does indeed affect performance to some extent, it does not account for the large differences described above.

Future work on the Tomcatv benchmark will include tiling the code for increased locality as well as parallelization to test runs across a cluster; since the benchmark is highly vectorizable (as stated on the SPEC Benchmark Specifications website at `http://www.spec.org/cpu92/DESCR.047`), we believe that we will eventually be able to get results at least as good as those on the Jacobi stencils.

Conclusions

Our experiences using the AlphaZ system demonstrated the AlphaZ system's potential usefulness in parallelizing equational mathematical problems. Noted shortcomings in the system included the inefficient method of memory allocation as well as the lack of multi-level tiling. Despite these, however, AlphaZ gave significant speedup when running in parallel with our modified memory allocation, even when run across a cluster rather than on a single node. We noted a possible inefficiency in the method of distributing the work across a cluster. Finally, we realized the possibility of implementing of running a real-world, non-trivial problem: the Tomcatv benchmark. Future work should examine the possibility of tiling this code and running it across a cluster; such an attempt, if successful, would be notable, as our previous attempts to perform parallel time-skewing for this code using other tools have failed.

Potential improvement of the AlphaZ system

Though our research found that the AlphaZ system shows much promise, it also uncovered room for improvement. Some suggestions — some of which may be straightforward to implement and may appear in AlphaZ in the near future, and some of which may be more difficult — are given below.

- An option could be allowed for allocation using static arrays rather than `malloc()`. This does not necessarily remove the ability to specify parameters at runtime; the user could specify an upper bound on the array size in the Alphabets file, which could then be used to determine the amount of memory to allocate at compile time.
- Even when `malloc()` is used, it is more efficient to use array arithmetic in a one-dimensional array than to use a cascade of pointers, as the AlphaZ system currently does. This improvement should be straightforward and could result in significant performance gains.
- Our runs demonstrated consistently poor performance of tiled code with small array sizes; this could be investigated (and perhaps fixed).
- The ability to implement multi-level tiling, though perhaps difficult, is very important for some mathematical problems, such as the two-dimensional Jacobi stencil.
- The ability to tile selected loops (rather than the whole problem) would be useful for certain problems.
- It seems that AlphaZ-generated code is not as efficient as it could be with regards to distribution of work; this is something that could be investigated.
- AlphaZ is probably going to support MPI (either in addition to or instead of [Cluster] OpenMP); when this happens, it should be able to support large problem sizes that make use of all the available memory on the cluster, rather than needing the entire problem to fit on one node (as is the case with Cluster OpenMP)
- The depreciated option that disables loop normalization should be retained until the underlying problem is repaired; without the option, codes such as our Tomcatv algorithm cannot be compiled.

Our suggestion that the `MOD` function be updated for improved performance has already been implemented.

Acknowledgements

The hand-written C code for the heat stencils is borrowed from work by Tim Douglas and Solomon Lutze (both Haverford College). Alphabets source code and schedules were written by Micah Walter (Haverford College) with input from Tomofumi Yuki (Colorado State University) and David Wonnacott (Haverford College). The AlphaZ-generated code was produced using the AlphaZ compiler from Colorado State University, version 2011-07-29 (except for generated code marked ‘Parallel’, which was produced by version 2011-06-24).

References

[Dou11] Tim Douglas. “An empirical study of the performance of scalable locality on a distributed shared memory system.” Pages 6–7. Haverford College, May 2011.

Appendix

Examples of AlphaZ source code

Note that some of this code is generated; it is reproduced here as-is, with the exception that wrapped lines are reformatted. In addition, the filenames for many of the files are provided as comments at the beginning of those files for convenient reference.

The generated files take as their filename the name of the affine function in the Alphabets file that generated them. To run a file with modified memory allocation using the built-in `Makefile`, it is necessary to rename it accordingly (for example, `oneD.c` is a generated AlphaZ file, and `oneD-modified.c` is the corresponding file with modified memory allocation; when running the modified code, we renamed `oneD.c` to `oneD-generated.c` and made a copy of `oneD-modified.c` named `oneD.c`.)

The wrappers and makefiles generated by the AlphaZ system are provided in the digital copy of these codes, as they are necessary to run the benchmarks, but they are not provided here, as they are of little referential use.

Note that the Python scripts to modify files must be in an ancestor directory of the files to be edited. The Python script which compiles the original Fortran code using `f2c` is included, but the original benchmark itself is not, for reasons of copyright.

Hand-written code for one-dimensional stencil: fast

```
/* Filename: oneD_hand_fast.c */

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#if ! defined PAD_ROWS_WITHIN_ONE_MALLOC
#define PAD_ROWS_WITHIN_ONE_MALLOC (2 * sizeof(double))
#endif

#define OPS_PER_ITER 4
#define A(t,i) (A_mem[((N)*(t))+(i)])

double cur_time(void) {
    struct timeval tv;
    gettimeofday(&tv, 0);
    return tv.tv_sec + tv.tv_usec*1.0e-6;
}

int main(int argc, char *argv[]) {
    /* If the following line is put outside the main() function,
     * as described in the technical report, performance drops.
     */
    static double *A_mem;

    double run_time, mflops, total = 0, sum_err_sqr = 0;
    char chtotal = 0;
```

```
    long i, t, N = -1, T = -1;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s T N\n", argv[0]);
        exit(-1);
    }

    T = atol(argv[1]);
    N = atol(argv[2]);

    if (N <= 0 || T <= 0) {
        fprintf(stderr, "T and N must be positive integers.\n");
        exit(-1);
    }

    A_mem = (double *) malloc(2 * (N) * sizeof(double));

    /* Initializing by filling with 1, as AlphaZ does */
    for (i = 0; i < N; i++)
        A(1,i) = 1;

    run_time = cur_time();
    // The +2 won't hurt the A[1] above or A[T%2] below but avoids -1%2
    for (t = 0+2; t < T+2; t++) {
        for (i = 1; i < N-1; i++) {
            A(t%2, i) = (A((t-1)%2, i-1) + 2*A((t-1)%2, i) +
                A((t-1)%2, i+1)) * (1.0/4);
        }
    }

    run_time = cur_time() - run_time;
    mflops = (((double) OPS_PER_ITER * N * T) / run_time) / 1000000L;

    printf("%f MFLOPS, ", mflops);
    printf("%f s\n", run_time);

    return 0;
}
```

Hand-written code for one-dimensional stencil: standard

```
/* Filename: oneD_hand_st'd.c */

#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

#define MALLOC_ARRAYS 0
#define STACK_ARRAYS 0

/* Define the parameters for the current problem instance;
 * these values need to be changed and the code re-compiled
 * when performing benchmarks on a different set of parameters.
 */

#define NMAX (131071984L)
#define N (1000000L)
```

```
#define T (10000L)

double cur_time(void) {
    struct timeval tv;
    struct timezone tz;
    gettimeofday(&tv, &tz);
    return tv.tv_sec + tv.tv_usec*1.0e-6;
}

#if ! MALLOC_ARRAYS && ! STACK_ARRAYS
// These need to be global or we get too big a stack frame.
double A[NMAX], new[NMAX];
#endif

int main(int argc, char *argv[]) {
#if ! MALLOC_ARRAYS && STACK_ARRAYS
    double A[NMAX], new[NMAX];
#endif
    double run_time, mflops, total = 0, sum_err_sqr = 0;
    char chtotal = 0;
    long int i, t;
    long long n_fp_ops = 0;
    int ops_per_iter = 4; // constant

    if (NMAX < N) {
        fprintf(stderr, "COMPILE FLAG INCONSISTENCY: NMAX (%ld) < N"
            " (%ld)\n", (long) NMAX, (long) N);
        exit(1);
    }

#if MALLOC_ARRAYS
    double *A, *new;
    A = kmp_sharable_malloc(2 * NMAX * sizeof(double));
    new = &(A[NMAX]);
    if (A==0 || new==0) {
        fprintf(stderr, "Failed to malloc two arrays of size %ld",
            (long) NMAX);
        exit(2);
    }
#endif

    /* Initializing by filling with 1, as AlphaZ does */
    for (i = 0; i < N; i++)
        A[i] = 1;
    run_time = cur_time();

    for (t = 0; t < T; t++) {
        for (i = 1; i < N-1; i++) {
            new[i] = (A[i-1] + 2*A[i] + A[i+1]) * (1.0/4);
        }

        for (i = 1; i < N-1; i++) {
            A[i] = new[i];
        }
    }

    run_time = cur_time() - run_time;
    mflops = (((double) ops_per_iter * N * T) / run_time) / 1000000L;

    printf("%f MFLOPS, ", mflops);
    printf("%f s\n", run_time);
```

```
    return 0;
}
```

Hand-written code for two-dimensional stencil

```
/* Filename: twoD_hand_st'd.c */

#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

#define MALLOC_ARRAYS 0

/* Define the parameters for the current problem instance;
 * these values need to be changed and the code re-compiled
 * when performing benchmarks on a different set of parameters.
 */

#define XMAS (1250L)
#define YMAX (1250L)
#define X (150L)
#define Y (8L)
#define T (5000000L)

double A[XMAS][YMAX], new[XMAS][YMAX];

double cur_time(void) {
    struct timeval tv;
    struct timezone tz;
    gettimeofday(&tv, &tz);
    return tv.tv_sec + tv.tv_usec*1.0e-6;
}

int main(int argc, char *argv[]) {
    double init_start_time, start_time, end_time, run_time;
    double mflops, total = 0, sum_err_sqr = 0;
    long int i, j, t;
    int ops_per_iter = 6; // constant

    if (XMAS < X) {
        fprintf(stderr, "COMPILE FLAG INCONSISTENCY: XMAS (%ld) < X"
            "(%ld)\n", (long) XMAS, (long) X);
        exit(1);
    }

    if (YMAX < Y) {
        fprintf(stderr, "COMPILE FLAG INCONSISTENCY: YMAX (%ld) < Y"
            "(%ld)\n", (long) YMAX, (long) Y);
        exit(1);
    }

    init_start_time = cur_time();

#if MALLOC_ARRAYS
    A = kmp_sharable_malloc(XMAS * sizeof(double));
    new = kmp_sharable_malloc(XMAS * sizeof(double));
    if (A==0 || new==0) {
```

```
        fprintf(stderr, "Failed to malloc two arrays of size %ld",
            (long) XMAS);
        exit(2);
    }
#endif

    /* Initializing by filling with 1, as AlphaZ does */
    for (i = 0; i < X; i++) {
        for (j = 0; j < Y; j++) {
            A[i][j] = rand() % 700;}
    }

    start_time = cur_time();

    for (t = 0; t < T; t++) {
        for (i = 1; i < X-1; i++) {
            for (j = 1; j < Y-1; j++) {
              new[i][j] = (A[i+1][j] + A[i-1][j] + A[i][j] + A[i][j-1] +
                  A[i][j+1]) * 0.2;
            }
        }

        for (i = 1; i < X-1; i++) {
            for (j = 1; j < Y-1; j++) {
                A[i][j] = new[i][j];
            }
        }
    }

    end_time = cur_time();
    run_time = end_time - start_time;

    mflops = (((double) ops_per_iter * X * Y * T) / run_time) / 1000000L;

    printf("%f MFLOPS, ", mflops);
    printf("%f s\n", run_time);

    return 0;
}
```

Alphabets file for one-dimensional stencil

```
/* Filename: oneD_standard.ab */

affine oneD {T,N | T>0 && N>4}

given
    double Initial {i | 0<=i<=N-1};

returns
    double Final {i | 0<=i<=N-1};

using
    double Intermediate {t,i | 0<=t<=T && 0<=i<=N-1};

through
    Intermediate[t,i] = case
        {| t==0} : Initial[i];
        {| t>0 && i==0} : (Intermediate[t-1,i]);
        {| t>0 && i==N-1} : (Intermediate[t-1,i]);
```

```
        {| t>0 && 0<i<N-1} :
            (2*Intermediate[t-1,i] + Intermediate[t-1,i-1] +
            Intermediate[t-1,i+1]) / 4.0;
    esac;

    Final[i] = Intermediate[T,i];
.
```

Alphabets file for one-dimensional stencil: two arrays

```
/* Filename: oneD_twoarrays.ab */

affine oneD {T,N | T>0 && N>4}

given
    double Initial {i | 0<=i<=N-1};

returns
    double Final {i | 0<=i<=N-1};

using
    double A {t,i | 0<=t<T && 0<=i<=N-1};
    double new {t,i | 0<t<=T && 0<=i<=N-1};

through
    new[t,i] = case
        {| i==0} || {| i==N-1} : A[t-1,i];
        {| 0<i<N-1} : (2*A[t-1,i] + A[t-1,i-1] + A[t-1,i+1]) / 4.0;
    esac;

    A[t,i] = case
        {| t==0} : Initial[i];
        {| 0<t<T} : new[t,i];
    esac;

    Final[i] = new[T,i];
.
```

Alphabets file for two-dimensional stencil

```
/* Filename: twoD.ab */

affine twoD {T,X,Y | T>0 && (X,Y) > 4}

given
    double Initial {i,j | 0<=i<X && 0<=j<Y};

returns
    double Final {i,j | 0<=i<X && 0<=j<Y};

using
    double Intermediate {t,i,j | 0<=t<=T && 0<=i<X && 0<=j<Y};

through
    Intermediate[t,i,j] = case
        {| t==0} :
            Initial[i,j];
        {| t>0 && j==0} ||
        {| t>0 && i==0} ||
        {| t>0 && i==X-1} ||
```

```
        {| t>0 && j==Y-1} :
            Intermediate[t-1,i,j];
        {| t>0 && 0<i<X-1 && 0<j<Y-1} :
            (4*Intermediate[t-1,i,j] + Intermediate[t-1,i,j+1] +
            Intermediate[t-1,i,j-1] + Intermediate[t-1,i+1,j] +
            Intermediate[t-1,i-1,j]) / 8.0;
    esac;

    Final[i,j] = Intermediate[T,i,j];
.
```

Alphabets file for Tomcatv: obvious

```
/* Filename: tomcatv_obvious.ab */

double abs(double);

/* This file has array indices begin at 1 rather than 0
 * in accordance with the original FORTRAN code.
 * Loops usually cover all but the first and last elements,
 * hence an iteration from 2 through N-1 (there are exceptions).
 */

affine Tomcatv {ITACT,N | (ITACT,N)>=4}

given
    double X0  {  i,j |               1<=(i,j)<=N  };
    double Y0  {  i,j |               1<=(i,j)<=N  };

returns
    double RXM {t     | 1<=t<=ITACT                };
    double RYM {t     | 1<=t<=ITACT                };

using
    double XX  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double YX  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double XY  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double YY  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double A   {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double B   {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double C   {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double AA  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double DD  {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double PXX {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double QXX {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double PYY {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double QYY {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double PXY {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double QXY {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RX0 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RY0 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double R   {t,i,j | 1<=t<=ITACT && 2<=i<=N-1 && 3<=j<=N-1};
    double D   {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RX1 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RY1 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RX2 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double RY2 {t,i,j | 1<=t<=ITACT && 2<=(i,j)<=N-1};
    double X   {t,i,j | 0<=t<=ITACT && 1<=(i,j)<=N  };
    double Y   {t,i,j | 0<=t<=ITACT && 1<=(i,j)<=N  };
```

```
through
    XX[t,i,j] = X[t-1,i+1,j] - X[t-1,i-1,j];
    YX[t,i,j] = Y[t-1,i+1,j] - Y[t-1,i-1,j];
    XY[t,i,j] = X[t-1,i,j+1] - X[t-1,i,j-1];
    YY[t,i,j] = Y[t-1,i,j+1] - Y[t-1,i,j-1];
    A[t,i,j] = 0.25 * (XY[t,i,j] * XY[t,i,j] + YY[t,i,j] * YY[t,i,j]);
    B[t,i,j] = 0.25 * (XX[t,i,j] * XX[t,i,j] + YX[t,i,j] * YX[t,i,j]);
    C[t,i,j] = 0.125 * (XX[t,i,j] * XY[t,i,j] + YX[t,i,j] * YY[t,i,j]);
    AA[t,i,j] = (-1) * B[t,i,j];
    DD[t,i,j] = B[t,i,j] + B[t,i,j] + (A[t,i,j] * (2.0 / 0.98));
    PXX[t,i,j] = X[t-1,i+1,j] - (2.0 * X[t-1,i,j]) + X[t-1,i-1,j];
    QXX[t,i,j] = Y[t-1,i+1,j] - (2.0 * Y[t-1,i,j]) + Y[t-1,i-1,j];
    PYY[t,i,j] = X[t-1,i,j+1] - (2.0 * X[t-1,i,j]) + X[t-1,i,j-1];
    QYY[t,i,j] = Y[t-1,i,j+1] - (2.0 * Y[t-1,i,j]) + Y[t-1,i,j-1];
    PXY[t,i,j] = X[t-1,i+1,j+1] - X[t-1,i+1,j-1] - X[t-1,i-1,j+1] +
        X[t-1,i-1,j-1];
    QXY[t,i,j] = Y[t-1,i+1,j+1] - Y[t-1,i+1,j-1] - Y[t-1,i-1,j+1] +
        Y[t-1,i-1,j-1];
    RX0[t,i,j] = A[t,i,j] * PXX[t,i,j] + B[t,i,j] * PYY[t,i,j] -
        C[t,i,j] * PXY[t,i,j];
    RY0[t,i,j] = A[t,i,j] * QXX[t,i,j] + B[t,i,j] * QYY[t,i,j] -
        C[t,i,j] * QXY[t,i,j];
    R[t,i,j] = AA[t,i,j] * D[t,i,j-1];

    D[t,i,j] = case
        {| j==2} : 1 / DD[t,i,j];
        {| j>=3} : 1 / (DD[t,i,j] - AA[t,i,j-1] * R[t,i,j]);
    esac;

    RX1[t,i,j] = case
        {| j==2} : RX0[t,i,j];
        {| 3<=j<=N-1} : RX0[t,i,j] - RX0[t,i,j-1] * R[t,i,j];
    esac;

    RY1[t,i,j] = case
        {| j==2} : RY0[t,i,j];
        {| 3<=j<=N-1} : RY0[t,i,j] - RY0[t,i,j-1] * R[t,i,j];
    esac;

    RX2[t,i,j] = case
        {| j==N-1} : RX1[t,i,j] * D[t,i,j];
        {| 2<=j<N-2} : RX1[t,i,j] - AA[t,i,j] * RX1[t,i,j+1] * D[t,i,j];
    esac;

    RY2[t,i,j] = case
        {| j==N-1} : RY1[t,i,j] * D[t,i,j];
        {| 2<=j<N-2} : RY1[t,i,j] - AA[t,i,j] * RY1[t,i,j+1] * D[t,i,j];
    esac;

    X[t,i,j] = case
        {| t==1} : X0[i,j];
        {| i==1} || {| j==1} || {| i==N} || {| j==N} : X[t-1,i,j];
        {| 2<=t<=ITACT && 2<=(i,j)<=N-1} : X[t-1,i,j] + RX2[t,i,j];
    esac;

    Y[t,i,j] = case
        {| t==1} : Y0[i,j];
        {| i==1} || {| j==1} || {| i==N} || {| j==N} : Y[t-1,i,j];
        {| 2<=t<=ITACT && 2<=(i,j)<=N-1} : Y[t-1,i,j] + RY2[t,i,j];
    esac;
```

```
    RXM[t] = reduce(max, [k,l], {| 2<=(k,l)<=N-1} : abs(RX0[t,k,l]));
    RYM[t] = reduce(max, [k,l], {| 2<=(k,l)<=N-1} : abs(RY0[t,k,l]));
.
```

Alphabets file for Tomcatv: fewer arrays

```
/* Filename: tomcatv_fewer.ab */

double abs(double);
double pow(double);
double fmax(double, double);

affine Tomcatv {ITACT,N | (ITACT,N)>3}

given
    double X0  {  i,j |                0<=(i,j)< N  };
    double Y0  {  i,j |                0<=(i,j)< N  };

returns
    double RXM {t     | 1<=t<=ITACT                 };
    double RYM {t     | 1<=t<=ITACT                 };

using
    double D   {t,i,j | 1<=t<=ITACT && 0< (i,j)< N-1};
    double RX  {t,i,j | 1<=t<=ITACT && 0< (i,j)< N-1};
    double RY  {t,i,j | 1<=t<=ITACT && 0< (i,j)< N-1};
    double X   {t,i,j | 0<=t<=ITACT && 0<=(i,j)<=N-1};
    double Y   {t,i,j | 0<=t<=ITACT && 0<=(i,j)<=N-1};

through
    D[t,i,j] = case
        {| j==1} :
            1 / ((0.5 * pow((X[t-1,i+1,j] - X[t-1,i-1,j]), 2) +
            2 * pow((Y[t-1,i+1,j] - Y[t-1,i-1,j]), 2)) + (0.25 * pow((X[t-1,
            i,j+1] - X[t-1,i,j-1]), 2) + (2.0 / 0.98) * pow((Y[t-1,i,j+1] -
            Y[t-1,i,j-1]), 2)));
        {| j>1} :
            1 / ((2 * (0.25 * pow((X[t-1,i+1,j] - X[t-1,i-1,j]), 2) +
            pow((Y[t-1,i+1,j] - Y[t-1,i-1,j]), 2)) + (0.25 * pow((X[t-1,
            i,j+1] - X[t-1,i,j-1]), 2) + pow((Y[t-1,i,j+1] - Y[t-1,i,j-1]),
            2)) * (2.0 / 0.98)) - (0.25 * pow((X[t-1,i+1,j-1] - X[t-1,j-1,
            i-1]), 2) + pow((Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1]), 2)) * (0.25 *
            pow((X[t-1,i+1,j] - X[t-1,i-1,j]), 2) + pow((Y[t-1,i+1,j] -
            Y[t-1,i-1,j]), 2)) * D[t,i,j-1]);
    esac;

    RX[t,i,j] = case
        {| j==1} :
            ((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
            X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
            Y[t-1,i,j-1])) * (X[t-1,i+1,j] - (2.0 * X[t-1,i,j]) + X[t-1,j,
            i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
            X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
            Y[t-1,i-1,j])) * (X[t-1,i,j+1] - (2.0 * X[t-1,i,j]) + X[t-1,j-1,
            i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
            X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
            Y[t-1,i,j-1])) * (X[t-1,i+1,j+1] - X[t-1,i+1,j-1] - X[t-1,j+1,
            i-1] + X[t-1,i-1,j-1])) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) *
            (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) *
```

```
    (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * ((0.25 * (X[t-1,i,j+2] - X[t-1,
    i,j]) * (X[t-1,i,j+2] - X[t-1,i,j]) + (Y[t-1,i,j+2] - Y[t-1,i,j])
    * (Y[t-1,i,j+2] - Y[t-1,i,j])) * (X[t-1,i+1,j+1] - (2.0 * X[t-1,
    i,j+1]) + X[t-1,i-1,j+1]) + (0.25 * (X[t-1,i+1,j+1] - X[t-1,j+1,
    i-1]) * (X[t-1,i+1,j+1] - X[t-1,i-1,j+1]) + (Y[t-1,i+1,j+1] -
    Y[t-1,i-1,j+1]) * (Y[t-1,i+1,j+1] - Y[t-1,i-1,j+1])) * (X[t-1,i,
    j+2] - (2.0 * X[t-1,i,j+1]) + X[t-1,i,j]) - (0.125 * (X[t-1,j+1,
    i+1] - X[t-1,i-1,j+1]) * (X[t-1,i,j+2] - X[t-1,i,j]) + (Y[t-1,
    i+1,j+1] - Y[t-1,i-1,j+1]) * (Y[t-1,i,j+2] - Y[t-1,i,j])) *
    (X[t-1,i+1,j+2] - X[t-1,i+1,j] - X[t-1,i-1,j+2] + X[t-1,i-1,j]))
    * D[t,i,j];
{| j==N-2} :
    (((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
    X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
    Y[t-1,i,j-1])) * (X[t-1,i+1,j] - (2.0 * X[t-1,i,j]) + X[t-1,j,
    i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
    X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
    Y[t-1,i-1,j])) * (X[t-1,i,j+1] - (2.0 * X[t-1,i,j]) + X[t-1,j-1,
    i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
    X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
    Y[t-1,i,j-1])) * (X[t-1,i+1,j+1] - X[t-1,i+1,j-1] - X[t-1,j+1,
    i-1] + X[t-1,i-1,j-1])) + ((0.25 * (X[t-1,i,j] - X[t-1,i,j-2]) *
    (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i,j] - Y[t-1,i,j-2]) *
    (Y[t-1,i,j] - Y[t-1,i,j-2])) * (X[t-1,i+1,j-1] - (2.0 * X[t-1,
    i,j-1]) + X[t-1,i-1,j-1]) + (0.25 * (X[t-1,i+1,j-1] - X[t-1,j-1,
    i-1]) * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) + (Y[t-1,i+1,j-1] -
    Y[t-1,i-1,j-1]) * (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1])) * (X[t-1,j,
    i] - (2.0 * X[t-1,i,j-1]) + X[t-1,i,j-2]) - (0.125 * (X[t-1,j-1,
    i+1] - X[t-1,i-1,j-1]) * (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,
    i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) *
    (X[t-1,i+1,j] - X[t-1,i+1,j-2] - X[t-1,i-1,j] + X[t-1,i-1,j-2]))
    * (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] - X[t-1,
    i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] - Y[t-1,
    i-1,j])) * D[t,i,j-1]) * D[t,i,N-2];
{| 1<j<N-2} :
    (((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
    X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
    Y[t-1,i,j-1])) * (X[t-1,i+1,j] - (2.0 * X[t-1,i,j]) + X[t-1,j,
    i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
    X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
    Y[t-1,i-1,j])) * (X[t-1,i,j+1] - (2.0 * X[t-1,i,j]) + X[t-1,j-1,
    i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
    X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
    Y[t-1,i,j-1])) * (X[t-1,i+1,j+1] - X[t-1,i+1,j-1] - X[t-1,j+1,
    i-1] + X[t-1,i-1,j-1])) + ((0.25 * (X[t-1,i,j] - X[t-1,i,j-2]) *
    (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i,j] - Y[t-1,i,j-2]) *
    (Y[t-1,i,j] - Y[t-1,i,j-2])) * (X[t-1,i+1,j-1] - (2.0 * X[t-1,
    i,j-1]) + X[t-1,i-1,j-1]) + (0.25 * (X[t-1,i+1,j-1] - X[t-1,j-1,
    i-1]) * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) + (Y[t-1,i+1,j-1] -
    Y[t-1,i-1,j-1]) * (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1])) * (X[t-1,j,
    i] - (2.0 * X[t-1,i,j-1]) + X[t-1,i,j-2]) - (0.125 * (X[t-1,j-1,
    i+1] - X[t-1,i-1,j-1]) * (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,
    i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) *
    (X[t-1,i+1,j] - X[t-1,i+1,j-2] - X[t-1,i-1,j] + X[t-1,i-1,j-2]))
    * (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] - X[t-1,
    i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] - Y[t-1,
    i-1,j])) * D[t,i,j-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) *
    (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) *
    (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * (((0.25 * (X[t-1,i,j+1] - X[t-1,
    i,j-1]) * (X[t-1,i,j+1] - X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,
    i,j-1]) * (Y[t-1,i,j+1] - Y[t-1,i,j-1])) * (X[t-1,i+1,j] - (2.0 *
```

```
        X[t-1,i,j]) + X[t-1,i-1,j]) + (0.25 * (X[t-1,i+1,j] - X[t-1,j,
        i-1]) * (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,j,
        i-1]) * (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * (X[t-1,i,j+1] - (2.0 *
        X[t-1,i,j]) + X[t-1,i,j-1]) - (0.125 * (X[t-1,i+1,j] - X[t-1,j,
        i-1]) * (X[t-1,i,j+1] - X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,j,
        i-1]) * (Y[t-1,i,j+1] - Y[t-1,i,j-1])) * (X[t-1,i+1,j+1] - X[t-1,
        i+1,j-1] - X[t-1,i-1,j+1] + X[t-1,i-1,j-1])) + ((0.25 * (X[t-1,j,
        i] - X[t-1,i,j-2]) * (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i,j] -
        Y[t-1,i,j-2]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) * (X[t-1,i+1,j-1] -
        (2.0 * X[t-1,i,j-1]) + X[t-1,i-1,j-1]) + (0.25 * (X[t-1,i+1,j-1]
        - X[t-1,i-1,j-1]) * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) + (Y[t-1,
        i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1])) *
        (X[t-1,i,j] - (2.0 * X[t-1,i,j-1]) + X[t-1,i,j-2]) - (0.125 *
        (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) * (X[t-1,i,j] - X[t-1,i,j-2]) +
        (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i,j] - Y[t-1,i,j-2]))
        * (X[t-1,i+1,j] - X[t-1,i+1,j-2] - X[t-1,i-1,j] + X[t-1,i-1,
        j-2])) * (0.25 * (X[t-1,i+1,j+1] - X[t-1,i-1,j+1]) * (X[t-1,j+1,
        i+1] - X[t-1,i-1,j+1]) + (Y[t-1,i+1,j+1] - Y[t-1,i-1,j+1]) *
        (Y[t-1,i+1,j+1] - Y[t-1,i-1,j+1])) * D[t,i,j]) * D[t,i,j];
esac;

RY[t,i,j] = case
    {| j==1} :
        ((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
        X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
        Y[t-1,i,j-1])) * (Y[t-1,i+1,j] - (2.0 * Y[t-1,i,j]) + Y[t-1,j,
        i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
        X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
        Y[t-1,i-1,j])) * (Y[t-1,i,j+1] - (2.0 * Y[t-1,i,j]) + Y[t-1,j-1,
        i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
        X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
        Y[t-1,i,j-1])) * (Y[t-1,i+1,j+1] - Y[t-1,i+1,j-1] - Y[t-1,j+1,
        i-1] + Y[t-1,i-1,j-1])) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) *
        (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) *
        (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * ((0.25 * (X[t-1,i,j+2] - X[t-1,
        i,j]) * (X[t-1,i,j+2] - X[t-1,i,j]) + (Y[t-1,i,j+2] - Y[t-1,i,j])
        * (Y[t-1,i,j+2] - Y[t-1,i,j])) * (Y[t-1,i+1,j+1] - (2.0 * Y[t-1,
        i,j+1]) + Y[t-1,i-1,j+1]) + (0.25 * (X[t-1,i+1,j+1] - X[t-1,j+1,
        i-1]) * (X[t-1,i+1,j+1] - X[t-1,i-1,j+1]) + (Y[t-1,i+1,j+1] -
        Y[t-1,i-1,j+1]) * (Y[t-1,i+1,j+1] - Y[t-1,i-1,j+1])) * (Y[t-1,i,
        j+2] - (2.0 * Y[t-1,i,j+1]) + Y[t-1,i,j]) - (0.125 * (X[t-1,j+1,
        i+1] - X[t-1,i-1,j+1]) * (X[t-1,i,j+2] - X[t-1,i,j]) + (Y[t-1,
        i+1,j+1] - Y[t-1,i-1,j+1]) * (Y[t-1,i,j+2] - Y[t-1,i,j])) *
        (Y[t-1,i+1,j+2] - Y[t-1,i+1,j] - Y[t-1,i-1,j+2] + Y[t-1,i-1,j]))
        * D[t,i,j];
    {| j==N-2} :
        (((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
        X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
        Y[t-1,i,j-1])) * (Y[t-1,i+1,j] - (2.0 * Y[t-1,i,j]) + Y[t-1,j,
        i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
        X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
        Y[t-1,i-1,j])) * (Y[t-1,i,j+1] - (2.0 * Y[t-1,i,j]) + Y[t-1,j-1,
        i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
        X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
        Y[t-1,i,j-1])) * (Y[t-1,i+1,j+1] - Y[t-1,i+1,j-1] - Y[t-1,j+1,
        i-1] + Y[t-1,i-1,j-1])) + ((0.25 * (X[t-1,i,j] - X[t-1,i,j-2]) *
        (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i,j] - Y[t-1,i,j-2]) *
        (Y[t-1,i,j] - Y[t-1,i,j-2])) * (Y[t-1,i+1,j-1] - (2.0 * Y[t-1,
        i,j-1]) + Y[t-1,i-1,j-1]) + (0.25 * (X[t-1,i+1,j-1] - X[t-1,j-1,
        i-1]) * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) + (Y[t-1,i+1,j-1] -
```

```
            Y[t-1,i-1,j-1]) * (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1])) * (Y[t-1,j,
            i] - (2.0 * Y[t-1,i,j-1]) + Y[t-1,i,j-2]) - (0.125 * (X[t-1,j-1,
            i+1] - X[t-1,i-1,j-1]) * (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,
            i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) *
            (Y[t-1,i+1,j] - Y[t-1,i+1,j-2] - Y[t-1,i-1,j] + Y[t-1,i-1,j-2]))
            * (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] - X[t-1,
            i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] - Y[t-1,
            i-1,j])) * D[t,i,j-1]) * D[t,i,N-2];
        {| 1<j<N-2} :
            (((0.25 * (X[t-1,i,j+1] - X[t-1,i,j-1]) * (X[t-1,i,j+1] -
            X[t-1,i,j-1]) + (Y[t-1,i,j+1] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] -
            Y[t-1,i,j-1])) * (Y[t-1,i+1,j] - (2.0 * Y[t-1,i,j]) + Y[t-1,j,
            i-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] -
            X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] -
            Y[t-1,i-1,j])) * (Y[t-1,i,j+1] - (2.0 * Y[t-1,i,j]) + Y[t-1,j-1,
            i]) - (0.125 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i,j+1] -
            X[t-1,i,j-1]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i,j+1] -
            Y[t-1,i,j-1])) * (Y[t-1,i+1,j+1] - Y[t-1,i+1,j-1] - Y[t-1,j+1,
            i-1] + Y[t-1,i-1,j-1])) + ((0.25 * (X[t-1,i,j] - X[t-1,i,j-2]) *
            (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i,j] - Y[t-1,i,j-2]) *
            (Y[t-1,i,j] - Y[t-1,i,j-2])) * (Y[t-1,i+1,j-1] - (2.0 * Y[t-1,
            i,j-1]) + Y[t-1,i-1,j-1]) + (0.25 * (X[t-1,i+1,j-1] - X[t-1,j-1,
            i-1]) * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) + (Y[t-1,i+1,j-1] -
            Y[t-1,i-1,j-1]) * (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1])) * (Y[t-1,j,
            i] - (2.0 * Y[t-1,i,j-1]) + Y[t-1,i,j-2]) - (0.125 * (X[t-1,j-1,
            i+1] - X[t-1,i-1,j-1]) * (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,
            i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) *
            (Y[t-1,i+1,j] - Y[t-1,i+1,j-2] - Y[t-1,i-1,j] + Y[t-1,i-1,j-2]))
            * (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) * (X[t-1,i+1,j] - X[t-1,
            i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) * (Y[t-1,i+1,j] - Y[t-1,
            i-1,j])) * D[t,i,j-1]) + (0.25 * (X[t-1,i+1,j] - X[t-1,i-1,j]) *
            (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] - Y[t-1,i-1,j]) *
            (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * D[t,i,j] * (((0.25 * (X[t-1,j+1,
            i] - X[t-1,i,j-1]) * (X[t-1,i,j+1] - X[t-1,i,j-1]) + (Y[t-1,j+1,
            i] - Y[t-1,i,j-1]) * (Y[t-1,i,j+1] - Y[t-1,i,j-1])) * (Y[t-1,j,
            i+1] - (2.0 * Y[t-1,i,j]) + Y[t-1,i-1,j]) + (0.25 * (X[t-1,i+1,j]
            - X[t-1,i-1,j]) * (X[t-1,i+1,j] - X[t-1,i-1,j]) + (Y[t-1,i+1,j] -
            Y[t-1,i-1,j]) * (Y[t-1,i+1,j] - Y[t-1,i-1,j])) * (Y[t-1,i,j+1] -
            (2.0 * Y[t-1,i,j]) + Y[t-1,i,j-1]) - (0.125 * (X[t-1,i+1,j] -
            X[t-1,i-1,j]) * (X[t-1,i,j+1] - X[t-1,i,j-1]) + (Y[t-1,i+1,j] -
            Y[t-1,i-1,j]) * (Y[t-1,i,j+1] - Y[t-1,i,j-1])) * (Y[t-1,i+1,j+1]
            - Y[t-1,i+1,j-1] - Y[t-1,i-1,j+1] + Y[t-1,i-1,j-1])) + ((0.25 *
            (X[t-1,i,j] - X[t-1,i,j-2]) * (X[t-1,i,j] - X[t-1,i,j-2]) +
            (Y[t-1,i,j] - Y[t-1,i,j-2]) * (Y[t-1,i,j] - Y[t-1,i,j-2])) *
            (Y[t-1,i+1,j-1] - (2.0 * Y[t-1,i,j-1]) + Y[t-1,i-1,j-1]) +
            (0.25 * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) * (X[t-1,i+1,j-1] -
            X[t-1,i-1,j-1]) + (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1]) * (Y[t-1,
            i+1,j-1] - Y[t-1,i-1,j-1])) * (Y[t-1,i,j] - (2.0 * Y[t-1,i,j-1])
            + Y[t-1,i,j-2]) - (0.125 * (X[t-1,i+1,j-1] - X[t-1,i-1,j-1]) *
            (X[t-1,i,j] - X[t-1,i,j-2]) + (Y[t-1,i+1,j-1] - Y[t-1,i-1,j-1]) *
            (Y[t-1,i,j] - Y[t-1,i,j-2])) * (Y[t-1,i+1,j] - Y[t-1,i+1,j-2] -
            Y[t-1,i-1,j] + Y[t-1,i-1,j-2])) * (0.25 * (X[t-1,i+1,j+1] -
            X[t-1,i-1,j+1]) * (X[t-1,i+1,j+1] - X[t-1,i-1,j+1]) + (Y[t-1,j+1,
            i+1] - Y[t-1,i-1,j+1]) * (Y[t-1,i+1,j+1] - Y[t-1,i-1,j+1])) *
            D[t,i,j]);
    esac;

    X[t,i,j] = case
        {| t==0} || {| i==0} || {| j==0} || {| i==N-1} || {| j==N-1} :
            X0[i,j];
        {| 0<t<=ITACT && 1<=(i,j)<N-1 } :
```

```
            X[t-1,i,j] + RX[t,i,j];
    esac;

    Y[t,i,j] = case
        {| t==0} || {| i==0} || {| j==0} || {| i==N-1} || {| j==N-1} :
            Y0[i,j];
       {| 0<t<=ITACT && 1<=(i,j)<N-1 } :
            Y[t-1,i,j] + RY[t,i,j];
    esac;

    RXM[t] = reduce(max, [k,l], {| 1<=(k,l)<N-1} : abs(RX[t,k,l]));
    RYM[t] = reduce(max, [k,l], {| 1<=(k,l)<N-1} : abs(RY[t,k,l]));
.
```

Schedule for one-dimensional stencil: untiled

```
# Filename: oneD_untiled.cs

program = ReadAlphabets("../oneD_standard.ab");
system = "oneD";
outDir = ".";

setSpaceTimeMap(program, system, "Intermediate", "(t,i -> t,i)");
setSpaceTimeMap(program, system, "Final", "(i -> T,i)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i -> t,i)", "(2,0)");
setMemoryMap(program, system, "Final", "Final", "(i -> T,i)", "(1,0)");

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for one-dimensional stencil: tiling

```
# Filename: oneD_tiling.cs

program = ReadAlphabets("../oneD_standard.ab");
system = "oneD";
outDir = ".";

setSpaceTimeMap(program, system, "Intermediate", "(t,i -> t,i+t)");
setSpaceTimeMap(program, system, "Final", "(i -> T,i+T)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i -> t,i)", "(2,0)");
setMemoryMap(program, system, "Final", "Final", "(i -> T,i)", "(1,0)");

setTiling(program, system, 0);

toptions = createTiledCGOptionForScheduledC();
setTiledCGOptionOptimize(toptions, 1);

VerifyTargetMapping(program, system, "MIN");
```

```
generateScheduledCode(program, system, toptions, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, toptions, outDir);
```

Schedule for one-dimensional stencil: parallel

```
# Filename: oneD_parallel.cs

program = ReadAlphabets("../oneD_standard.ab");
system = "oneD";
outDir = ".";

setSpaceTimeMap(program, system, "Intermediate", "(t,i -> t,i+t)");
setSpaceTimeMap(program, system, "Final", "(i -> T,i+T)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i -> t,i)", "(2,0)");
setMemoryMap(program, system, "Final", "Final", "(i -> T,i)", "(1,0)");

setTiling(program, system, 0);
# The following line is the only one that makes this
# different from the tiling code.
setTilingType(program, system, "omp");

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for one-dimensional stencil: copy arrays

```
# Filename: oneD_copy.cs

program = ReadAlphabets("../oneD_twoarrays.ab");
system = "oneD";
outDir = ".";

setSpaceTimeMap(program, system, "A", "(t,i -> t,1,i)");
setSpaceTimeMap(program, system, "new", "(t,i -> t,0,i)");
setSpaceTimeMap(program, system, "Final", "(i -> T,1,i)");

setDimensionType(program, system, "A", 1, "O");
setDimensionType(program, system, "new", 1, "O");
setDimensionType(program, system, "Final", 1, "O");

# The statement ordering is not necessary because we have assigned
# a constant dimension 1 in each array to "O" for "Ordering" (above).

setMemoryMap(program, system, "A", "A", "(t,i -> i)");
setMemoryMap(program, system, "new", "new", "(t,i -> i)");
setMemoryMap(program, system, "Final", "Final", "(i -> i)");

options = createCGOptionForScheduledC();
setCGOptionFlattenArrays(options, 1);

VerifyTargetMapping(program, system, "MIN");
```

```
generateScheduledCode(program, system, options, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, options, outDir);
```

Schedule for two-dimensional stencil: untiled

```
# Filename: twoD_untiled.cs

program = ReadAlphabets("../twoD.ab");
system = "twoD";
outDir = ".";

setSpaceTimeMap(program, system, "Intermediate", "(t,i,j -> t,i,j)");
setSpaceTimeMap(program, system, "Final", "(i,j -> T,i,j)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i,j -> t,i,j)", "(2,0,0)");
setMemoryMap(program, system, "Final", "Final", "(i,j -> T,i,j)", "(1,0,0)");

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for two-dimensional stencil: tiling

```
# Filename: twoD_tiling.cs

program = ReadAlphabets("../twoD.ab");
system = "twoD";
outDir = ".";

setSpaceTimeMap(program, system, "Intermediate", "(t,i,j -> t,i+t,j+t)");
setSpaceTimeMap(program, system, "Final", "(i,j -> T,i+T,j+T)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i,j -> t,i,j)", "(2,0,0)");
setMemoryMap(program, system, "Final", "Final", "(i,j -> T,i,j)", "(1,0,0)");

setTiling(program, system, 0);

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for two-dimensional stencil: parallel

```
# Filename: twoD_parallel.cs

program = ReadAlphabets("../twoD.ab");
system = "twoD";
outDir = ".";
```

```
setSpaceTimeMap(program, system, "Final", "(i,j -> T,i+T,j+T)");

setStatementOrdering(program, system, "Intermediate", "Final");

setMemoryMap(program, system, "Intermediate", "Intermediate",
    "(t,i,j -> t,i,j)", "(2,0,0)");
setMemoryMap(program, system, "Final", "Final", "(i,j -> T,i,j)", "(1,0,0)");

setTiling(program, system, 0);
# The following line is the only one that makes this
# different from the tiling code.
setTilingType(program, system, "omp");

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for Tomcatv: obvious

```
# Filename: Tomcatv_obvious.cs

program = ReadAlphabets("../tomcatv_obvious.ab");
system = "Tomcatv";
outDir = ".";

# The two arrays marked with a comment sign involve countdown loops
setSpaceTimeMap(program, system, "XX",  "(t,i,j -> 0,t,0,i,0,j,0)");
setSpaceTimeMap(program, system, "YX",  "(t,i,j -> 0,t,0,i,0,j,1)");
setSpaceTimeMap(program, system, "XY",  "(t,i,j -> 0,t,0,i,0,j,2)");
setSpaceTimeMap(program, system, "YY",  "(t,i,j -> 0,t,0,i,0,j,3)");
setSpaceTimeMap(program, system, "A",   "(t,i,j -> 0,t,0,i,0,j,4)");
setSpaceTimeMap(program, system, "B",   "(t,i,j -> 0,t,0,i,0,j,5)");
setSpaceTimeMap(program, system, "C",   "(t,i,j -> 0,t,0,i,0,j,6)");
setSpaceTimeMap(program, system, "AA",  "(t,i,j -> 0,t,0,i,0,j,7)");
setSpaceTimeMap(program, system, "DD",  "(t,i,j -> 0,t,0,i,0,j,8)");
setSpaceTimeMap(program, system, "PXX", "(t,i,j -> 0,t,0,i,0,j,9)");
setSpaceTimeMap(program, system, "QXX", "(t,i,j -> 0,t,0,i,0,j,10)");
setSpaceTimeMap(program, system, "PYY", "(t,i,j -> 0,t,0,i,0,j,11)");
setSpaceTimeMap(program, system, "QYY", "(t,i,j -> 0,t,0,i,0,j,12)");
setSpaceTimeMap(program, system, "PXY", "(t,i,j -> 0,t,0,i,0,j,13)");
setSpaceTimeMap(program, system, "QXY", "(t,i,j -> 0,t,0,i,0,j,14)");
setSpaceTimeMap(program, system, "RX0", "(t,i,j -> 0,t,0,i,0,j,15)");
setSpaceTimeMap(program, system, "RY0", "(t,i,j -> 0,t,0,i,0,j,16)");
setSpaceTimeMap(program, system, "RXM", "(t     -> 0,t,1,N,0,N,0)");
setSpaceTimeMap(program, system, "RYM", "(t     -> 0,t,1,N,0,N,1)");
setSpaceTimeMap(program, system, "R",   "(t,i,j -> 0,t,2,i,0,j,0)");
setSpaceTimeMap(program, system, "D",   "(t,i,j -> 0,t,2,i,0,j,1)");
setSpaceTimeMap(program, system, "RX1", "(t,i,j -> 0,t,2,i,0,j,2)");
setSpaceTimeMap(program, system, "RY1", "(t,i,j -> 0,t,2,i,0,j,3)");
setSpaceTimeMap(program, system, "RX2", "(t,i,j -> 0,t,3,i,0,N-j,0)"); #
setSpaceTimeMap(program, system, "RY2", "(t,i,j -> 0,t,3,i,0,N-j,1)"); #
setSpaceTimeMap(program, system, "X",   "(t,i,j -> 0,t,4,i,0,j,0)");
setSpaceTimeMap(program, system, "Y",   "(t,i,j -> 0,t,4,i,0,j,1)");

# We set some of the constant dimensions as ordering

setDimensionType(program, system, "XX", 0, "O");
setDimensionType(program, system, "XX", 2, "O");
```

```
setDimensionType(program, system, "XX", 4, "O");
setDimensionType(program, system, "XX", 6, "O");
setDimensionType(program, system, "YX", 0, "O");
setDimensionType(program, system, "YX", 2, "O");
setDimensionType(program, system, "YX", 4, "O");
setDimensionType(program, system, "YX", 6, "O");
setDimensionType(program, system, "XY", 0, "O");
setDimensionType(program, system, "XY", 2, "O");
setDimensionType(program, system, "XY", 4, "O");
setDimensionType(program, system, "XY", 6, "O");
setDimensionType(program, system, "YY", 0, "O");
setDimensionType(program, system, "YY", 2, "O");
setDimensionType(program, system, "YY", 4, "O");
setDimensionType(program, system, "YY", 6, "O");
setDimensionType(program, system, "A", 0, "O");
setDimensionType(program, system, "A", 2, "O");
setDimensionType(program, system, "A", 4, "O");
setDimensionType(program, system, "A", 6, "O");
setDimensionType(program, system, "B", 0, "O");
setDimensionType(program, system, "B", 2, "O");
setDimensionType(program, system, "B", 4, "O");
setDimensionType(program, system, "B", 6, "O");
setDimensionType(program, system, "C", 0, "O");
setDimensionType(program, system, "C", 2, "O");
setDimensionType(program, system, "C", 4, "O");
setDimensionType(program, system, "C", 6, "O");
setDimensionType(program, system, "AA", 0, "O");
setDimensionType(program, system, "AA", 2, "O");
setDimensionType(program, system, "AA", 4, "O");
setDimensionType(program, system, "AA", 6, "O");
setDimensionType(program, system, "DD", 0, "O");
setDimensionType(program, system, "DD", 2, "O");
setDimensionType(program, system, "DD", 4, "O");
setDimensionType(program, system, "DD", 6, "O");
setDimensionType(program, system, "PXX", 0, "O");
setDimensionType(program, system, "PXX", 2, "O");
setDimensionType(program, system, "PXX", 4, "O");
setDimensionType(program, system, "PXX", 6, "O");
setDimensionType(program, system, "QXX", 0, "O");
setDimensionType(program, system, "QXX", 2, "O");
setDimensionType(program, system, "QXX", 4, "O");
setDimensionType(program, system, "QXX", 6, "O");
setDimensionType(program, system, "PYY", 0, "O");
setDimensionType(program, system, "PYY", 2, "O");
setDimensionType(program, system, "PYY", 4, "O");
setDimensionType(program, system, "PYY", 6, "O");
setDimensionType(program, system, "QYY", 0, "O");
setDimensionType(program, system, "QYY", 2, "O");
setDimensionType(program, system, "QYY", 4, "O");
setDimensionType(program, system, "QYY", 6, "O");
setDimensionType(program, system, "PXY", 0, "O");
setDimensionType(program, system, "PXY", 2, "O");
setDimensionType(program, system, "PXY", 4, "O");
setDimensionType(program, system, "PXY", 6, "O");
setDimensionType(program, system, "QXY", 0, "O");
setDimensionType(program, system, "QXY", 2, "O");
setDimensionType(program, system, "QXY", 4, "O");
setDimensionType(program, system, "QXY", 6, "O");
setDimensionType(program, system, "RX0", 0, "O");
setDimensionType(program, system, "RX0", 2, "O");
setDimensionType(program, system, "RX0", 4, "O");
```

```
setDimensionType(program, system, "RX0", 6, "O");
setDimensionType(program, system, "RY0", 0, "O");
setDimensionType(program, system, "RY0", 2, "O");
setDimensionType(program, system, "RY0", 4, "O");
setDimensionType(program, system, "RY0", 6, "O");
setDimensionType(program, system, "RXM", 0, "O");
setDimensionType(program, system, "RXM", 2, "O");
setDimensionType(program, system, "RXM", 4, "O");
setDimensionType(program, system, "RXM", 6, "O");
setDimensionType(program, system, "RYM", 0, "O");
setDimensionType(program, system, "RYM", 2, "O");
setDimensionType(program, system, "RYM", 4, "O");
setDimensionType(program, system, "RYM", 6, "O");
setDimensionType(program, system, "R", 0, "O");
setDimensionType(program, system, "R", 2, "O");
setDimensionType(program, system, "R", 4, "O");
setDimensionType(program, system, "R", 6, "O");
setDimensionType(program, system, "D", 0, "O");
setDimensionType(program, system, "D", 2, "O");
setDimensionType(program, system, "D", 4, "O");
setDimensionType(program, system, "D", 6, "O");
setDimensionType(program, system, "RX1", 0, "O");
setDimensionType(program, system, "RX1", 2, "O");
setDimensionType(program, system, "RX1", 4, "O");
setDimensionType(program, system, "RX1", 6, "O");
setDimensionType(program, system, "RY1", 0, "O");
setDimensionType(program, system, "RY1", 2, "O");
setDimensionType(program, system, "RY1", 4, "O");
setDimensionType(program, system, "RY1", 6, "O");
setDimensionType(program, system, "RX2", 0, "O");
setDimensionType(program, system, "RX2", 2, "O");
setDimensionType(program, system, "RX2", 4, "O");
setDimensionType(program, system, "RX2", 6, "O");
setDimensionType(program, system, "RY2", 0, "O");
setDimensionType(program, system, "RY2", 2, "O");
setDimensionType(program, system, "RY2", 4, "O");
setDimensionType(program, system, "RY2", 6, "O");
setDimensionType(program, system, "X", 0, "O");
setDimensionType(program, system, "X", 2, "O");
setDimensionType(program, system, "X", 4, "O");
setDimensionType(program, system, "X", 6, "O");
setDimensionType(program, system, "Y", 0, "O");
setDimensionType(program, system, "Y", 2, "O");
setDimensionType(program, system, "Y", 4, "O");
setDimensionType(program, system, "Y", 6, "O");

setMemoryMap(program, system, "XX",  "XX",  "(t,i,j -> )");
setMemoryMap(program, system, "YX",  "YX",  "(t,i,j -> )");
setMemoryMap(program, system, "XY",  "XY",  "(t,i,j -> )");
setMemoryMap(program, system, "YY",  "YY",  "(t,i,j -> )");
setMemoryMap(program, system, "A",   "A",   "(t,i,j -> )");
setMemoryMap(program, system, "B",   "B",   "(t,i,j -> )");
setMemoryMap(program, system, "C",   "C",   "(t,i,j -> )");
setMemoryMap(program, system, "AA",  "AA",  "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "DD",  "DD",  "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "PXX", "PXX", "(t,i,j -> )");
setMemoryMap(program, system, "QXX", "QXX", "(t,i,j -> )");
setMemoryMap(program, system, "PYY", "PYY", "(t,i,j -> )");
setMemoryMap(program, system, "QYY", "QYY", "(t,i,j -> )");
setMemoryMap(program, system, "PXY", "PXY", "(t,i,j -> )");
setMemoryMap(program, system, "QXY", "QXY", "(t,i,j -> )");
```

```
setMemoryMap(program, system, "RX0", "RX0", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RY0", "RY0", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "R",   "R",   "(t,i,j -> )");
setMemoryMap(program, system, "D",   "D",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RX1", "RX1", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RY1", "RY1", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RX2", "RX2", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RY2", "RY2", "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "X",   "X",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "Y",   "Y",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RXM", "RXM", "(t     -> t,0,0)"       );
setMemoryMap(program, system, "RYM", "RYM", "(t     -> t,0,0)"       );

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, outDir);
```

Schedule for Tomcatv: fewer arrays

```
# Filename: Tomcatv_fewer.cs

program = ReadAlphabets("../tomcatv_fewer.ab");
system = "Tomcatv";
outDir = ".";

setSpaceTimeMap (program, system, "D",   "(t,i,j -> t,i,j)");
setSpaceTimeMap (program, system, "RX",  "(t,i,j -> t,i,j)");
setSpaceTimeMap (program, system, "RY",  "(t,i,j -> t,i,j)");
setSpaceTimeMap (program, system, "X",   "(t,i,j -> t,i,j)");
setSpaceTimeMap (program, system, "Y",   "(t,i,j -> t,i,j)");
setSpaceTimeMap (program, system, "RXM", "(t     -> t,N,N)");
setSpaceTimeMap (program, system, "RYM", "(t     -> t,N,N)");

setStatementOrdering(program, system, "D", "RX");
setStatementOrdering(program, system, "D", "RY");
setStatementOrdering(program, system, "D", "X");
setStatementOrdering(program, system, "D", "Y");
setStatementOrdering(program, system, "D", "RXM");
setStatementOrdering(program, system, "D", "RYM");
setStatementOrdering(program, system, "RX", "RY");
setStatementOrdering(program, system, "RX", "X");
setStatementOrdering(program, system, "RX", "Y");
setStatementOrdering(program, system, "RX", "RXM");
setStatementOrdering(program, system, "RX", "RYM");
setStatementOrdering(program, system, "RY", "X");
setStatementOrdering(program, system, "RY", "Y");
setStatementOrdering(program, system, "RY", "RXM");
setStatementOrdering(program, system, "RY", "RYM");
setStatementOrdering(program, system, "X", "Y");
setStatementOrdering(program, system, "X", "RXM");
setStatementOrdering(program, system, "X", "RYM");
setStatementOrdering(program, system, "Y", "RXM");
setStatementOrdering(program, system, "Y", "RYM");
setStatementOrdering(program, system, "RXM", "RYM");

setMemoryMap(program, system, "D",   "D",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RX",  "RX",  "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RY",  "RY",  "(t,i,j -> t,i,j)", "(1,0,0)");
```

```
setMemoryMap(program, system, "X",   "X",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "Y",   "Y",   "(t,i,j -> t,i,j)", "(1,0,0)");
setMemoryMap(program, system, "RXM", "RXM", "(t     -> t,0,0)"       );
setMemoryMap(program, system, "RYM", "RYM", "(t     -> t,0,0)"       );

options = createCGOptionForScheduledC();
setCGOptionDisableNormalize_depreciated(options);

VerifyTargetMapping(program, system, "MIN");

generateScheduledCode(program, system, options, outDir);
generateMakefile(program, system, outDir);
generateWrapper(program, system, options, outDir);
```

Generated code for one-dimensional stencil: untiled

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]
//Function bodies
void oneD(long T,long N,double* Initial,double** Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
    printf("The value of parameters are not vaild.\n");
    exit(-1);
    }
    //Loop indices for malloc
```

```
    int mz1, mz2;
    double** Intermediate = malloc(sizeof(double*)*(2));
    if (Intermediate == NULL) {
    printf("Failed to allocate memory for Intermediate : size=%ld\n",
        (2) * sizeof(double*));
    exit(-1);
    }
    for (mz1=0; mz1<2; mz1++){
    Intermediate[mz1] = malloc(sizeof(double)*(N));
    if (Intermediate[mz1] == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (N) * sizeof(double));
        exit(-1);
    }
    }
    #define S0(t,i) Intermediate(t,i) = Initial(i);
    #define S1(t,i) Intermediate(t,i) = Intermediate(t-1,i);
    #define S2(t,i) Intermediate(t,i) = Intermediate(t-1,i);
    #define S3(t,i) Intermediate(t,i) = ((((2)*(Intermediate(t-1,i)))+\
    (Intermediate(t-1,i-1)))+(Intermediate(t-1,i+1)))/(4.0);
    #define S4(i,i1) Final(T,i1) = Intermediate(T,i1);
    {
    //Domain
    //{t,i|t== 0 && T-1>= 0 && N-5>= 0 && N-i-1>= 0 && i>= 0}
    //{t,i|i== 0 && T-1>= 0 && N-5>= 0 && t-1>= 0 && T-t>= 0}
    //{t,i|-N+i+1== 0 && T-1>= 0 && N-5>= 0 && t-1>= 0 && T-t>= 0}
    //{t,i|T-1>= 0 && N-5>= 0 && t-1>= 0 && i-1>= 0 && N-i-2>= 0 &&
    //T-t>= 0}
    //{i,i1|-T+i== 0 && T-1>= 0 && N-5>= 0 && N-i1-1>= 0 && i1>= 0}
    int c1,c2;
    for(c2=0;c2 <= N + -1;c2+=1){
        S0((0),(c2));
    }
    for(c1=1;c1 <= T + -1;c1+=1){
        S1((c1),(0));
        for(c2=1;c2 <= N + -2;c2+=1){
        S3((c1),(c2));
        }
        S2((c1),(N + -1));
    }
    S1((T),(0));
    S4((T),(0));
    for(c2=1;c2 <= N + -2;c2+=1){
        S3((T),(c2));
        S4((T),(c2));
    }
    S2((T),(N + -1));
    S4((T),(N + -1));
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
    //Memory Free
    for (mz1=0; mz1<2; mz1++){
    free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
```

```
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for one-dimensional stencil: tiling

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]
//Function bodies
void oneD(long T,long N,int ts1,int ts2,double* Initial,double** Final){
```

```
// Parameter checking
if (!((T-1>= 0 && N-5>= 0))) {
    printf("The value of parameters are not vaild.\n");
    exit(-1);
}
//Loop indices for malloc
int mz1, mz2;
double** Intermediate = malloc(sizeof(double*)*(2));
if (Intermediate == NULL) {
    printf("Failed to allocate memory for Intermediate : size=%ld\n",
        (2) * sizeof(double*));
    exit(-1);
}
for (mz1=0; mz1<2; mz1++){
    Intermediate[mz1] = malloc(sizeof(double)*(N));
    if (Intermediate[mz1] == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (N) * sizeof(double));
        exit(-1);
    }
}
#define S0(t,i) Intermediate(t,-t+i) = Initial(-t+i);
#define S1(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
#define S2(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
#define S3(t,i) Intermediate(t,-t+i) = ((((2)*(Intermediate(t-1,-t+i)))+\
    (Intermediate(t-1,-t+i-1)))+(Intermediate(t-1,-t+i+1)))/(4.0);
#define S4(i,i1) Final(T,i1-T) = Intermediate(T,i1-T);
{
    int c1,c2,ti1,ti2;
    for(ti1=ceild(-ts1 + 1, ts1)*ts1;ti1 <= T;ti1+=ts1){
        for(ti2=ceild(min(0,min(ti1,T)) + -ts2 + 1, ts2)*ts2;ti2 <=
            max(N + -1,max(N + ti1 + -1,N + T + -1)) +
            max(0,ts1 + -1);ti2+=ts2){
            /** guard that isolates selected statements for generic point
                loops **//** point loops with optimizations **/
            if((0 < ti1) && (ti1 + ts1 + -1 < ti2) && (ti1 < T + -ts1 +
                1) && (ti2 < ti1 + N + -ts2 + 0)){
                /** full-tile guard **/
                if((1 <= ti1) && (ti1 <= T + -ts1 + 0) && (ti1 + ts1 +
                    0 <= ti2) && (ti2 <= ti1 + N + -ts2 + -1)){
                    for(c1=ti1;c1 <= ti1 + ts1 + -1;c1+=1){
                        for(c2=ti2;c2 <= ti2 + ts2 + -1;c2+=1){
                            S3((c1),(c2));
                        }
                    }
                } else {
                    for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
                        T + -1);c1+=1){
                        for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
                            N + c1 + -2);c2+=1){
                            S3((c1),(c2));
                        }
                    }
                }
            } else {
                {
                    for(c1=max(ti1,0);c1 <= min(ti1 + ts1 + -1,0);c1+=1){
                        for(c2=max(ti2,0);c2 <= min(ti2 + ts2 + -1,
                            N + -1);c2+=1){
                            S0((0),(c2));
                        }
```

```
                        }
                        for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
                            T + -1);c1+=1){
                            for(c2=max(ti2,c1);c2 <= min(ti2 + ts2 + -1,
                                c1);c2+=1){
                                S1((c1),(c1));
                            }
                            for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
                                N + c1 + -2);c2+=1){
                                S3((c1),(c2));
                            }
                            for(c2=max(ti2,N + c1 + -1);c2 <= min(ti2 + ts2 +
                                -1,N + c1 + -1);c2+=1){
                                S2((c1),(N + c1 + -1));
                            }
                        }
                        for(c1=max(ti1,T);c1 <= min(ti1 + ts1 + -1,T);c1+=1){
                            for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,
                                T);c2+=1){
                                S1((T),(T));
                            }
                            for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,T);
                                c2+=1){
                                S4((T),(T));
                            }
                            for(c2=max(ti2,T + 1);c2 <= min(ti2 + ts2 + -1,
                                N + T + -2);c2+=1){
                                S3((T),(c2));
                                S4((T),(c2));
                            }
                            for(c2=max(ti2,N + T + -1);c2 <= min(ti2 + ts2 +
                                -1,N + T + -1);c2+=1){
                                S2((T),(N + T + -1));
                                S4((T),(N + T + -1));
                            }
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
    //Memory Free
    for (mz1=0; mz1<2; mz1++){
        free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
```

```
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for one-dimensional stencil: parallel

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ( (i)>=0 ? (i)%(j) : (j-1)-((-(i))%(j)) )
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]
//Function bodies
void oneD(long T,long N,int T1,int T2,double* Initial,double** Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2;
    double** Intermediate = malloc(sizeof(double*)*(2));
```

```
if (Intermediate == NULL) {
    printf("Failed to allocate memory for Intermediate : size=%ld\n",
        (2) * sizeof(double*));
    exit(-1);
}
for (mz1=0; mz1<2; mz1++){
    Intermediate[mz1] = malloc(sizeof(double)*(N));
    if (Intermediate[mz1] == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (N) * sizeof(double));
        exit(-1);
    }
}
#define S0(t,i) Intermediate(t,-t+i) = Initial(-t+i);
#define S1(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
#define S2(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
#define S3(t,i) Intermediate(t,-t+i) = ((((2)*(Intermediate(t-1,-t+i)))+\
    (Intermediate(t-1,-t+i-1)))+(Intermediate(t-1,-t+i+1)))/(4.0);
#define S4(i,i1) Final(T,i1-T) = Intermediate(T,i1-T);
{
    int c1,c2,end,ss1,ss2,start,time;
    ss1=CDIV(1-T1,T1)*T1;
    ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;
    start=ss1/T1+ss2/T2;
    ss1=T;
    ss2=MAX(N-1,MAX(ss1+N-1,T+N-1))+MAX(0,T1-1);
    end=ss1/T1+ss2/T2;
    for(time=start;time <= end;time+=1){
        #pragma omp parallel for private(c1,c2,ss1,ss2)
        for(ss1=CDIV(1-T1,T1)*T1;ss1 <= T;ss1+=T1){
            ss2=(time-ss1/T1)*T2;
            if(CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2<=ss1 && ss2<=MAX(N-1,
                MAX(ss1+N-1,T+N-1))+MAX(0,T1-1)){
                for(c1=MAX(0,ss1);c1 <= MIN(0,ss1+T1-1);c1+=1){
                    for(c2=MAX(0,ss2);c2 <= MIN(N-1,ss2+T2-1);c2+=1){
                        S0(0,c2);
                    }
                }
                for(c1=MAX(1,ss1);c1 <= MIN(T-1,ss1+T1-1);c1+=1){
                    for(c2=MAX(c1,ss2);c2 <= MIN(c1,ss2+T2-1);c2+=1){
                        S1(c1,c1);
                    }
                    for(c2=MAX(c1+1,ss2);c2 <= MIN(c1+N-2,ss2+T2-1);
                        c2+=1){
                        S3(c1,c2);
                    }
                    for(c2=MAX(c1+N-1,ss2);c2 <= MIN(c1+N-1,ss2+T2-1);
                        c2+=1){
                        S2(c1,c1+N-1);
                    }
                }
                for(c1=MAX(T,ss1);c1 <= MIN(T,ss1+T1-1);c1+=1){
                    for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                        S1(T,T);
                    }
                    for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                        S4(T,T);
                    }
                    for(c2=MAX(T+1,ss2);c2 <= MIN(T+N-2,ss2+T2-1);c2+=1){
                        S3(T,c2);
                        S4(T,c2);
```

```
                        }
                        for(c2=MAX(T+N-1,ss2);c2 <= MIN(T+N-1,ss2+T2-1);
                            c2+=1){
                            S2(T,T+N-1);
                            S4(T,T+N-1);
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
    //Memory Free
    for (mz1=0; mz1<2; mz1++){
        free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for one-dimensional stencil: copy arrays

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
```

```
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i) Initial[i]
#define A(t) A[t]
#define new(t) new[t]
#define Final(i) Final[i]
//Function bodies
void oneD(long T,long N,double* Initial,double* Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    double* A = malloc(sizeof(double)*(N));
    if (A == NULL) {
        printf("Failed to allocate memory for A : size=%ld\n",
            (N) * sizeof(double));
        exit(-1);
    }
    double* new = malloc(sizeof(double)*(N));
    if (new == NULL) {
        printf("Failed to allocate memory for new : size=%ld\n",
            (N) * sizeof(double));
        exit(-1);
    }
    #define S0(t,i,i2) new(i2) = A(i2);
    #define S1(t,i,i2) new(i2) = ((((2)*(A(i2)))+(A(i2-1)))+(A(i2+1)))/(4.0);
    #define S2(t,i,i2) A(i2) = Initial(i2);
    #define S3(t,i,i2) A(i2) = new(i2);
    #define S4(i,i1,i2) Final(i2) = new(i2);
    {
        //Domain
//{t,i,i2|i2== 0 && i== 0 && T-t>= 0 && N-5>= 0 && t-1>= 0} ||
    //{t,i,i2|-N+i2+1== 0 && i== 0 && T-t>= 0 && N-5>= 0 && t-1>= 0}
    //{t,i,i2|i== 0 && T-1>= 0 && N-5>= 0 && i2-1>= 0 && N-i2-2>= 0 &&
        //T-t>= 0 && t-1>= 0}
    //{t,i,i2|i-1== 0 && t== 0 && T-1>= 0 && N-5>= 0 && N-i2-1>= 0 && i2>= 0}
    //{t,i,i2|i-1== 0 && T-1>= 0 && N-5>= 0 && t-1>= 0 && T-t-1>= 0 &&
        //N-i2-1>= 0 && i2>= 0}
    //{i,i1,i2|i1-1== 0 && -T+i== 0 && T-1>= 0 && N-5>= 0 && N-i2-1>= 0 &&
        //i2>= 0}
        int c1,c3;
        for(c3=0;c3 <= N + -1;c3+=1){
            S2((0),(1),(c3));
        }
```

```
        for(c1=1;c1 <= T + -1;c1+=1){
            S0((c1),(0),(0));
            for(c3=1;c3 <= N + -2;c3+=1){
                S1((c1),(0),(c3));
            }
            S0((c1),(0),(N + -1));
            for(c3=0;c3 <= N + -1;c3+=1){
                S3((c1),(1),(c3));
            }
        }
        S0((T),(0),(0));
        for(c3=1;c3 <= N + -2;c3+=1){
            S1((T),(0),(c3));
        }
        S0((T),(0),(N + -1));
        for(c3=0;c3 <= N + -1;c3+=1){
            S4((T),(1),(c3));
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
    //Memory Free
    free(A);
    free(new);
}
#undef Initial
#undef A
#undef new
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for two-dimensional stencil: untiled

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]
//Function bodies
void twoD(long T,long X,long Y,double** Initial,double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2, mz3;
    double*** Intermediate = malloc(sizeof(double**)*(2));
    if (Intermediate == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (2) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<2; mz1++){
        Intermediate[mz1] = malloc(sizeof(double*)*(X));
        if (Intermediate[mz1] == NULL) {
            printf("Failed to allocate memory for Intermediate : size=%ld\n",
                (X) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<X; mz2++){
            Intermediate[mz1][mz2] = malloc(sizeof(double)*(Y));
            if (Intermediate[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for Intermediate :"
                    "size=%ld\n", (Y) * sizeof(double));
                exit(-1);
            }
        }
    }
    #define S0(t,i,j) Intermediate(t,i,j) = Initial(i,j);
    #define S1(t,i,j) Intermediate(t,i,j) = Intermediate(t-1,i,j);
```

```
    #define S2(t,i,j) Intermediate(t,i,j) = ((((((4)*(Intermediate(t-1,i,\
        j)))+(Intermediate(t-1,i,j+1)))+(Intermediate(t-1,i,j-1)))+\
        Intermediate(t-1,i+1,j)))+(Intermediate(t-1,i-1,j)))/(8.0);
    #define S3(i,j,i2) Final(T,j,i2) = Intermediate(T,j,i2);
    {
        //Domain
    //{t,i,j|t== 0 && T-1>= 0 && X-5>= 0 && Y-5>= 0 && Y-j-1>= 0 && j>= 0 &&
        //X-i-1>= 0 && i>= 0}
//{t,i,j|j== 0 && i>= 0 && X-5>= 0 && Y-5>= 0 && t-1>= 0 && T-t>= 0 &&
    //X-i-1>= 0} || {t,i,j|i== 0 && T-t>= 0 && X-5>= 0 && Y-5>= 0 &&
    //t-1>= 0 && Y-j-1>= 0 && j>= 0} || {t,i,j|-X+i+1== 0 && T-t>= 0 &&
    //X-5>= 0 && Y-5>= 0 && t-1>= 0 && Y-j-1>= 0 && j>= 0} ||
    //{t,i,j|-Y+j+1== 0 && i>= 0 && X-5>= 0 && Y-5>= 0 && t-1>= 0 &&
    //T-t>= 0 && X-i-1>= 0}
    //{t,i,j|T-1>= 0 && X-5>= 0 && Y-5>= 0 && t-1>= 0 && i-1>= 0 &&
        //X-i-2>= 0 && j-1>= 0 && Y-j-2>= 0 && T-t>= 0}
    //{i,j,i2|-T+i== 0 && T-1>= 0 && X-5>= 0 && Y-5>= 0 && Y-i2-1>= 0 &&
        //i2>= 0 && j>= 0 && X-j-1>= 0}
        int c1,c2,c3;
        for(c2=0;c2 <= X + -1;c2+=1){
            for(c3=0;c3 <= Y + -1;c3+=1){
                S0((0),(c2),(c3));
            }
        }
        for(c1=1;c1 <= T + -1;c1+=1){
            for(c3=0;c3 <= Y + -1;c3+=1){
                S1((c1),(0),(c3));
            }
            for(c2=1;c2 <= X + -2;c2+=1){
                S1((c1),(c2),(0));
                for(c3=1;c3 <= Y + -2;c3+=1){
                    S2((c1),(c2),(c3));
                }
                S1((c1),(c2),(Y + -1));
            }
            for(c3=0;c3 <= Y + -1;c3+=1){
                S1((c1),(X + -1),(c3));
            }
        }
        for(c3=0;c3 <= Y + -1;c3+=1){
            S1((T),(0),(c3));
            S3((T),(0),(c3));
        }
        for(c2=1;c2 <= X + -2;c2+=1){
            S1((T),(c2),(0));
            S3((T),(c2),(0));
            for(c3=1;c3 <= Y + -2;c3+=1){
                S2((T),(c2),(c3));
                S3((T),(c2),(c3));
            }
            S1((T),(c2),(Y + -1));
            S3((T),(c2),(Y + -1));
        }
        for(c3=0;c3 <= Y + -1;c3+=1){
            S1((T),(X + -1),(c3));
            S3((T),(X + -1),(c3));
        }
    }
    #undef S0
    #undef S1
    #undef S2
```

```
    #undef S3
    //Memory Free
    for (mz1=0; mz1<2; mz1++){
        for (mz2=0; mz2<X; mz2++){
            free(Intermediate[mz1][mz2]);
        }
        free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for two-dimensional stencil: tiling

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
```

```
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]
//Function bodies
void twoD(long T,long X,long Y,int ts1,int ts2,int ts3,double** Initial,
    double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2, mz3;
    double*** Intermediate = malloc(sizeof(double**)*(2));
    if (Intermediate == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (2) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<2; mz1++){
        Intermediate[mz1] = malloc(sizeof(double*)*(X));
        if (Intermediate[mz1] == NULL) {
            printf("Failed to allocate memory for Intermediate : size=%ld\n",
                (X) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<X; mz2++){
            Intermediate[mz1][mz2] = malloc(sizeof(double)*(Y));
            if (Intermediate[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for Intermediate :"
                    "size=%ld\n", (Y) * sizeof(double));
                exit(-1);
            }
        }
    }
    #define S0(t,i,j) Intermediate(t,-t+i,-t+j) = Initial(-t+i,-t+j);
    #define S1(t,i,j) Intermediate(t,-t+i,-t+j) = Intermediate(t-1,-t+i,
        -t+j);
    #define S2(t,i,j) Intermediate(t,-t+i,-t+j) = ((((((4)*(Intermediate(\
        t-1,-t+i,-t+j)))+(Intermediate(t-1,-t+i,-t+j+1)))+(Intermediate(t-1,\
        -t+i,-t+j-1)))+(Intermediate(t-1,-t+i+1,-t+j)))+(Intermediate(t-1,\
        -t+i-1,-t+j)))/(8.0);
    #define S3(i,j,i2) Final(T,j-T,i2-T) = Intermediate(T,j-T,i2-T);
    {
        int c1,c2,c3,ti1,ti2,ti3;
        for(ti1=ceild(-ts1 + 1, ts1)*ts1;ti1 <= T;ti1+=ts1){
            for(ti2=ceild(min(0,min(ti1,T)) + -ts2 + 1, ts2)*ts2;
                ti2 <= max(X + -1,max(X + ti1 + -1,T + X + -1)) +
                max(0,ts1 + -1);ti2+=ts2){
                for(ti3=ceild(min(T,min(ti1,0)) + -ts3 + 1, ts3)*ts3;
                    ti3 <= max(T + Y + -1,max(Y + ti1 + -1,Y + -1)) +
                        max(0,ts1 + -1);ti3+=ts3){
                    /** point loops for generic case **/
                    {
```

```
for(c1=max(ti1,0);c1 <= min(ti1 + ts1 + -1,0);c1+=1){
    for(c2=max(ti2,0);c2 <= min(ti2 + ts2 + -1,
        X + -1);c2+=1){
        for(c3=max(ti3,0);c3 <= min(ti3 + ts3 + -1,
            Y + -1);c3+=1){
            S0((0),(c2),(c3));
        }
    }
}
for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
    T + -1);c1+=1){
    for(c2=max(ti2,c1);c2 <= min(ti2 + ts2 + -1,c1);
        c2+=1){
        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
            Y + c1 + -1);c3+=1){
            S1((c1),(c1),(c3));
        }
    }
    for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
        X + c1 + -2);c2+=1){
        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
            c1);c3+=1){
            S1((c1),(c2),(c1));
        }
        for(c3=max(ti3,c1 + 1);c3 <= min(ti3 + ts3 +
            -1,Y + c1 + -2);c3+=1){
            S2((c1),(c2),(c3));
        }
        for(c3=max(ti3,Y + c1 + -1);c3 <= min(ti3 +
            ts3 + -1,Y + c1 + -1);c3+=1){
            S1((c1),(c2),(Y + c1 + -1));
        }
    }
    for(c2=max(ti2,X + c1 + -1);c2 <= min(ti2 + ts2 +
        -1,X + c1 + -1);c2+=1){
        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
            Y + c1 + -1);c3+=1){
            S1((c1),(X + c1 + -1),(c3));
        }
    }
}
for(c1=max(ti1,T);c1 <= min(ti1 + ts1 + -1,T);c1+=1){
    for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,T);
        c2+=1){
        for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
            T + Y + -1);c3+=1){
            S1((T),(T),(c3));
            S3((T),(T),(c3));
        }
    }
    for(c2=max(ti2,T + 1);c2 <= min(ti2 + ts2 + -1,
        T + X + -2);c2+=1){
        for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
            T);c3+=1){
            S1((T),(c2),(T));
            S3((T),(c2),(T));
        }
        for(c3=max(ti3,T + 1);c3 <= min(ti3 + ts3 +
            -1,T + Y + -2);c3+=1){
            S2((T),(c2),(c3));
            S3((T),(c2),(c3));
```

```
                                }
                                for(c3=max(ti3,T + Y + -1);c3 <= min(ti3 +
                                    ts3 + -1,T + Y + -1);c3+=1){
                                    S1((T),(c2),(T + Y + -1));
                                    S3((T),(c2),(T + Y + -1));
                                }
                            }
                            for(c2=max(ti2,T + X + -1);c2 <= min(ti2 + ts2 +
                                -1,T + X + -1);c2+=1){
                                for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
                                    T + Y + -1);c3+=1){
                                    S1((T),(T + X + -1),(c3));
                                    S3((T),(T + X + -1),(c3));
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    //Memory Free
    for (mz1=0; mz1<2; mz1++){
        for (mz2=0; mz2<X; mz2++){
            free(Intermediate[mz1][mz2]);
        }
        free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for two-dimensional stencil: parallel

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ( (i)>=0 ? (i)%(j) : (j-1)-((-(i))%(j)) )
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Memory Macros
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]
//Function bodies
void twoD(long T,long X,long Y,int T1,int T2,int T3,double** Initial,
    double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2, mz3;
    double*** Intermediate = malloc(sizeof(double**)*(2));
    if (Intermediate == NULL) {
        printf("Failed to allocate memory for Intermediate : size=%ld\n",
            (2) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<2; mz1++){
        Intermediate[mz1] = malloc(sizeof(double*)*(X));
        if (Intermediate[mz1] == NULL) {
            printf("Failed to allocate memory for Intermediate : size=%ld\n",
                (X) * sizeof(double*));
            exit(-1);
        }
```

```
    for (mz2=0; mz2<X; mz2++){
        Intermediate[mz1][mz2] = malloc(sizeof(double)*(Y));
        if (Intermediate[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for Intermediate :"
                "size=%ld\n", (Y) * sizeof(double));
            exit(-1);
        }
    }
}
#define S0(t,i,j) Intermediate(t,-t+i,-t+j) = Initial(-t+i,-t+j);
#define S1(t,i,j) Intermediate(t,-t+i,-t+j) = Intermediate(t-1,-t+i,\
    -t+j);
#define S2(t,i,j) Intermediate(t,-t+i,-t+j) = ((((((4)*(Intermediate(\
    t-1,-t+i,-t+j)))+(Intermediate(t-1,-t+i,-t+j+1)))+(Intermediate(t-1,\
    -t+i,-t+j-1)))+(Intermediate(t-1,-t+i+1,-t+j)))+(Intermediate(t-1,\
    -t+i-1,-t+j)))/(8.0);
#define S3(i,j,i2) Final(T,j-T,i2-T) = Intermediate(T,j-T,i2-T);
{
    int c1,c2,c3,end,ss1,ss2,ss3,start,time;
    ss1=CDIV(1-T1,T1)*T1;
    ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;
    ss3=CDIV(1+MIN(0,MIN(ss2,T))-T3,T3)*T3;
    start=ss1/T1+ss2/T2+ss3/T3;
    ss1=T;
    ss2=MAX(X-1,MAX(ss1+X-1,T+X-1))+MAX(0,T1-1);
    ss3=MAX(Y-1,MAX(ss2+Y-1,T+Y-1))+MAX(0,T1-1);
    end=ss1/T1+ss2/T2+ss3/T3;
    for(time=start;time <= end;time+=1){
        #pragma omp parallel for private(c1,c2,c3,ss1,ss2,ss3)
        for(ss1=CDIV(1-T1,T1)*T1;ss1 <= T;ss1+=T1){
            for(ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;ss2 <=
                MAX(X-1,MAX(ss1+X-1,T+X-1))+MAX(0,T1-1);ss2+=T2){
                ss3=(time-ss1/T1-ss2/T2)*T3;
                if(CDIV(1+MIN(0,MIN(ss2,T))-T3,T3)*T3<=ss2 &&
                    ss3<=MAX(Y-1,MAX(ss2+Y-1,T+Y-1))+MAX(0,T1-1)){
                    for(c1=MAX(0,ss1);c1 <= MIN(0,ss1+T1-1);c1+=1){
                        for(c2=MAX(0,ss2);c2 <= MIN(X-1,ss2+T2-1);c2+=1){
                            for(c3=MAX(0,ss3);c3 <= MIN(Y-1,ss3+T3-1);
                                c3+=1){
                                S0(0,c2,c3);
                            }
                        }
                    }
                    for(c1=MAX(1,ss1);c1 <= MIN(T-1,ss1+T1-1);c1+=1){
                        for(c2=MAX(c1,ss2);c2 <= MIN(c1,ss2+T2-1);c2+=1){
                            for(c3=MAX(c1,ss3);c3 <= MIN(c1+Y-1,
                                ss3+T3-1);c3+=1){
                                S1(c1,c1,c3);
                            }
                        }
                        for(c2=MAX(c1+1,ss2);c2 <= MIN(c1+X-2,ss2+T2-1);
                            c2+=1){
                            for(c3=MAX(c1,ss3);c3 <= MIN(c1,ss3+T3-1);
                                c3+=1){
                                S1(c1,c2,c1);
                            }
                            for(c3=MAX(c1+1,ss3);c3 <= MIN(c1+Y-2,
                                ss3+T3-1);c3+=1){
                                S2(c1,c2,c3);
                            }
```

```
                                for(c3=MAX(c1+Y-1,ss3);c3 <= MIN(c1+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(c1,c2,c1+Y-1);
                                }
                            }
                            for(c2=MAX(c1+X-1,ss2);c2 <= MIN(c1+X-1,
                                ss2+T2-1);c2+=1){
                                for(c3=MAX(c1,ss3);c3 <= MIN(c1+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(c1,c1+X-1,c3);
                                }
                            }
                        }
                        for(c1=MAX(T,ss1);c1 <= MIN(T,ss1+T1-1);c1+=1){
                            for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T+Y-1,ss3+T3-1);
                                    c3+=1){
                                    S1(T,T,c3);
                                    S3(T,T,c3);
                                }
                            }
                            for(c2=MAX(T+1,ss2);c2 <= MIN(T+X-2,ss2+T2-1);
                                c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T,ss3+T3-1);
                                    c3+=1){
                                    S1(T,c2,T);
                                    S3(T,c2,T);
                                }
                                for(c3=MAX(T+1,ss3);c3 <= MIN(T+Y-2,
                                    ss3+T3-1);c3+=1){
                                    S2(T,c2,c3);
                                    S3(T,c2,c3);
                                }
                                for(c3=MAX(T+Y-1,ss3);c3 <= MIN(T+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(T,c2,T+Y-1);
                                    S3(T,c2,T+Y-1);
                                }
                            }
                            for(c2=MAX(T+X-1,ss2);c2 <= MIN(T+X-1,ss2+T2-1);
                                c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T+Y-1,ss3+T3-1);
                                    c3+=1){
                                    S1(T,T+X-1,c3);
                                    S3(T,T+X-1,c3);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
#undef S0
#undef S1
#undef S2
#undef S3
//Memory Free
for (mz1=0; mz1<2; mz1++){
    for (mz2=0; mz2<X; mz2++){
        free(Intermediate[mz1][mz2]);
```

```
        }
        free(Intermediate[mz1]);
    }
    free(Intermediate);
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for Tomcatv: obvious

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include "external_functions.h"
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))
```

```
//Local Function Declarations
double Tomcatv_RXM_reduce_1(long,long,int,double**,double**,double*,double*,
    double*,double*,double*,double*,double*,double***,double***,double*,
    double*,double*,double*,double*,double*,double***,double***,double*,
    double***,double***,double***,double***,double***,double***,double***,
    double***,double***);
double Tomcatv_RYM_reduce_1(long,long,int,double**,double**,double*,double*,
    double*,double*,double*,double*,double*,double***,double***,double*,
    double*,double*,double*,double*,double*,double***,double***,double*,
    double***,double***,double***,double***,double***,double***,double***,
    double***,double***);
//Memory Macros
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define XX() XX[0]
#define YX() YX[0]
#define XY() XY[0]
#define YY() YY[0]
#define A() A[0]
#define B() B[0]
#define C() C[0]
#define AA(t,i,j) AA[MOD(t, 1)][i][j]
#define DD(t,i,j) DD[MOD(t, 1)][i][j]
#define PXX() PXX[0]
#define QXX() QXX[0]
#define PYY() PYY[0]
#define QYY() QYY[0]
#define PXY() PXY[0]
#define QXY() QXY[0]
#define RX0(t,i,j) RX0[MOD(t, 1)][i][j]
#define RY0(t,i,j) RY0[MOD(t, 1)][i][j]
#define R() R[0]
#define D(t,i,j) D[MOD(t, 1)][i][j]
#define RX1(t,i,j) RX1[MOD(t, 1)][i][j]
#define RY1(t,i,j) RY1[MOD(t, 1)][i][j]
#define RX2(t,i,j) RX2[MOD(t, 1)][i][j]
#define RY2(t,i,j) RY2[MOD(t, 1)][i][j]
#define X(t,i,j) X[MOD(t, 1)][i][j]
#define Y(t,i,j) Y[MOD(t, 1)][i][j]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]
//Function bodies
void Tomcatv(long ITACT,long N,double** X0,double** Y0,double*** RXM,
    double*** RYM){
    // Parameter checking
    if (!((ITACT-4>= 0 && N-4>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2, mz3;
    double* XX = malloc(sizeof(double)*(1));
    if (XX == NULL) {
        printf("Failed to allocate memory for XX : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* YX = malloc(sizeof(double)*(1));
    if (YX == NULL) {
        printf("Failed to allocate memory for YX : size=%ld\n",
            (1) * sizeof(double));
```

```
    exit(-1);
}
double* XY = malloc(sizeof(double)*(1));
if (XY == NULL) {
    printf("Failed to allocate memory for XY : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double* YY = malloc(sizeof(double)*(1));
if (YY == NULL) {
    printf("Failed to allocate memory for YY : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double* A = malloc(sizeof(double)*(1));
if (A == NULL) {
    printf("Failed to allocate memory for A : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double* B = malloc(sizeof(double)*(1));
if (B == NULL) {
    printf("Failed to allocate memory for B : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double* C = malloc(sizeof(double)*(1));
if (C == NULL) {
    printf("Failed to allocate memory for C : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double*** AA = malloc(sizeof(double**)*(1));
if (AA == NULL) {
    printf("Failed to allocate memory for AA : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    AA[mz1] = malloc(sizeof(double*)*(N));
    if (AA[mz1] == NULL) {
        printf("Failed to allocate memory for AA : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        AA[mz1][mz2] = malloc(sizeof(double)*(N));
        if (AA[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for AA : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double*** DD = malloc(sizeof(double**)*(1));
if (DD == NULL) {
    printf("Failed to allocate memory for DD : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
```

```
        DD[mz1] = malloc(sizeof(double*)*(N));
        if (DD[mz1] == NULL) {
            printf("Failed to allocate memory for DD : size=%ld\n",
                (N) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N; mz2++){
            DD[mz1][mz2] = malloc(sizeof(double)*(N));
            if (DD[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for DD : size=%ld\n",
                    (N) * sizeof(double));
                exit(-1);
            }
        }
    }
    double* PXX = malloc(sizeof(double)*(1));
    if (PXX == NULL) {
        printf("Failed to allocate memory for PXX : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* QXX = malloc(sizeof(double)*(1));
    if (QXX == NULL) {
        printf("Failed to allocate memory for QXX : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* PYY = malloc(sizeof(double)*(1));
    if (PYY == NULL) {
        printf("Failed to allocate memory for PYY : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* QYY = malloc(sizeof(double)*(1));
    if (QYY == NULL) {
        printf("Failed to allocate memory for QYY : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* PXY = malloc(sizeof(double)*(1));
    if (PXY == NULL) {
        printf("Failed to allocate memory for PXY : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double* QXY = malloc(sizeof(double)*(1));
    if (QXY == NULL) {
        printf("Failed to allocate memory for QXY : size=%ld\n",
            (1) * sizeof(double));
        exit(-1);
    }
    double*** RX0 = malloc(sizeof(double**)*(1));
    if (RX0 == NULL) {
        printf("Failed to allocate memory for RX0 : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        RX0[mz1] = malloc(sizeof(double*)*(N));
        if (RX0[mz1] == NULL) {
```

```
        printf("Failed to allocate memory for RX0 : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        RX0[mz1][mz2] = malloc(sizeof(double)*(N));
        if (RX0[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for RX0 : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double*** RY0 = malloc(sizeof(double**)*(1));
if (RY0 == NULL) {
    printf("Failed to allocate memory for RY0 : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    RY0[mz1] = malloc(sizeof(double*)*(N));
    if (RY0[mz1] == NULL) {
        printf("Failed to allocate memory for RY0 : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        RY0[mz1][mz2] = malloc(sizeof(double)*(N));
        if (RY0[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for RY0 : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double* R = malloc(sizeof(double)*(1));
if (R == NULL) {
    printf("Failed to allocate memory for R : size=%ld\n",
        (1) * sizeof(double));
    exit(-1);
}
double*** D = malloc(sizeof(double**)*(1));
if (D == NULL) {
    printf("Failed to allocate memory for D : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    D[mz1] = malloc(sizeof(double*)*(N));
    if (D[mz1] == NULL) {
        printf("Failed to allocate memory for D : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        D[mz1][mz2] = malloc(sizeof(double)*(N));
        if (D[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for D : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
```

```
        }
    }
    double*** RX1 = malloc(sizeof(double**)*(1));
    if (RX1 == NULL) {
        printf("Failed to allocate memory for RX1 : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        RX1[mz1] = malloc(sizeof(double*)*(N));
        if (RX1[mz1] == NULL) {
            printf("Failed to allocate memory for RX1 : size=%ld\n",
                (N) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N; mz2++){
            RX1[mz1][mz2] = malloc(sizeof(double)*(N));
            if (RX1[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for RX1 : size=%ld\n",
                    (N) * sizeof(double));
                exit(-1);
            }
        }
    }
    double*** RY1 = malloc(sizeof(double**)*(1));
    if (RY1 == NULL) {
        printf("Failed to allocate memory for RY1 : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        RY1[mz1] = malloc(sizeof(double*)*(N));
        if (RY1[mz1] == NULL) {
            printf("Failed to allocate memory for RY1 : size=%ld\n",
                (N) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N; mz2++){
            RY1[mz1][mz2] = malloc(sizeof(double)*(N));
            if (RY1[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for RY1 : size=%ld\n",
                    (N) * sizeof(double));
                exit(-1);
            }
        }
    }
    double*** RX2 = malloc(sizeof(double**)*(1));
    if (RX2 == NULL) {
        printf("Failed to allocate memory for RX2 : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        RX2[mz1] = malloc(sizeof(double*)*(N));
        if (RX2[mz1] == NULL) {
            printf("Failed to allocate memory for RX2 : size=%ld\n",
                (N) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N; mz2++){
            RX2[mz1][mz2] = malloc(sizeof(double)*(N));
```

```
        if (RX2[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for RX2 : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double*** RY2 = malloc(sizeof(double**)*(1));
if (RY2 == NULL) {
    printf("Failed to allocate memory for RY2 : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    RY2[mz1] = malloc(sizeof(double*)*(N));
    if (RY2[mz1] == NULL) {
        printf("Failed to allocate memory for RY2 : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        RY2[mz1][mz2] = malloc(sizeof(double)*(N));
        if (RY2[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for RY2 : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double*** X = malloc(sizeof(double**)*(1));
if (X == NULL) {
    printf("Failed to allocate memory for X : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    X[mz1] = malloc(sizeof(double*)*(N+1));
    if (X[mz1] == NULL) {
        printf("Failed to allocate memory for X : size=%ld\n",
            (N+1) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N+1; mz2++){
        X[mz1][mz2] = malloc(sizeof(double)*(N+1));
        if (X[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for X : size=%ld\n",
                (N+1) * sizeof(double));
            exit(-1);
        }
    }
}
double*** Y = malloc(sizeof(double**)*(1));
if (Y == NULL) {
    printf("Failed to allocate memory for Y : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    Y[mz1] = malloc(sizeof(double*)*(N+1));
    if (Y[mz1] == NULL) {
```

```
            printf("Failed to allocate memory for Y : size=%ld\n",
                (N+1) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N+1; mz2++){
            Y[mz1][mz2] = malloc(sizeof(double)*(N+1));
            if (Y[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for Y : size=%ld\n",
                    (N+1) * sizeof(double));
                exit(-1);
            }
        }
    }
    #define S0(t,i,j,i3,i4,i5,i6) XX() = (X(i-1,i3+1,i5))-\
        (X(i-1,i3-1,i5));
    #define S1(t,i,j,i3,i4,i5,i6) YX() = (Y(i-1,i3+1,i5))-(Y(i-1,i3-1,i5));
    #define S2(t,i,j,i3,i4,i5,i6) XY() = (X(i-1,i3,i5+1))-(X(i-1,i3,i5-1));
    #define S3(t,i,j,i3,i4,i5,i6) YY() = (Y(i-1,i3,i5+1))-(Y(i-1,i3,i5-1));
    #define S4(t,i,j,i3,i4,i5,i6) A() = (0.25)*(((XY())*(XY()))+\
        ((YY())*(YY())));
    #define S5(t,i,j,i3,i4,i5,i6) B() = (0.25)*(((XX())*(XX()))+\
        ((YX())*(YX())));
    #define S6(t,i,j,i3,i4,i5,i6) C() = (0.125)*(((XX())*(XY()))+\
        ((YX())*(YY())));
    #define S7(t,i,j,i3,i4,i5,i6) AA(i,i3,i5) = (-(1))*(B());
    #define S8(t,i,j,i3,i4,i5,i6) DD(i,i3,i5) = ((B())+(B()))+\
        ((A())*((2.0)/(0.98)));
    #define S9(t,i,j,i3,i4,i5,i6) PXX() = ((X(i-1,i3+1,i5))-\
        ((2.0)*(X(i-1,i3,i5))))+(X(i-1,i3-1,i5));
    #define S10(t,i,j,i3,i4,i5,i6) QXX() = ((Y(i-1,i3+1,i5))-\
        ((2.0)*(Y(i-1,i3,i5))))+(Y(i-1,i3-1,i5));
    #define S11(t,i,j,i3,i4,i5,i6) PYY() = ((X(i-1,i3,i5+1))-\
        ((2.0)*(X(i-1,i3,i5))))+(X(i-1,i3,i5-1));
    #define S12(t,i,j,i3,i4,i5,i6) QYY() = ((Y(i-1,i3,i5+1))-\
        ((2.0)*(Y(i-1,i3,i5))))+(Y(i-1,i3,i5-1));
    #define S13(t,i,j,i3,i4,i5,i6) PXY() = (((X(i-1,i3+1,i5+1))-\
        (X(i-1,i3+1,i5-1)))-(X(i-1,i3-1,i5+1)))+(X(i-1,i3-1,i5-1));
    #define S14(t,i,j,i3,i4,i5,i6) QXY() = (((Y(i-1,i3+1,i5+1))-\
        (Y(i-1,i3+1,i5-1)))-(Y(i-1,i3-1,i5+1)))+(Y(i-1,i3-1,i5-1));
    #define S15(t,i,j,i3,i4,i5,i6) RX0(i,i3,i5) = (((A())*(PXX()))+\
        ((B())*(PYY())))-((C())*(PXY()));
    #define S16(t,i,j,i3,i4,i5,i6) RY0(i,i3,i5) = (((A())*(QXX()))+
        (B())*(QYY())))-((C())*(QXY()));
    #define S17(t,i,j,i3,i4,i5,i6) R() = (AA(i,i3,i5))*(D(i,i3,i5-1));
    #define S18(t,i,j,i3,i4,i5,i6) D(i,i3,i5) = (1)/(DD(i,i3,i5));
    #define S19(t,i,j,i3,i4,i5,i6) D(i,i3,i5) = (1)/((DD(i,i3,i5))-\
        ((AA(i,i3,i5-1))*(R())));
    #define S20(t,i,j,i3,i4,i5,i6) RX1(i,i3,i5) = RX0(i,i3,i5);
    #define S21(t,i,j,i3,i4,i5,i6) RX1(i,i3,i5) = (RX0(i,i3,i5))-\
        ((RX0(i,i3,i5-1))*(R()));
    #define S22(t,i,j,i3,i4,i5,i6) RY1(i,i3,i5) = RY0(i,i3,i5);
    #define S23(t,i,j,i3,i4,i5,i6) RY1(i,i3,i5) = (RY0(i,i3,i5))-\
        ((RY0(i,i3,i5-1))*(R()));
    #define S24(t,i,j,i3,i4,i5,i6) RX2(i,i3,-i5+N) = (RX1(i,i3,-i5+N))*\
        (D(i,i3,-i5+N));
    #define S25(t,i,j,i3,i4,i5,i6) RX2(i,i3,-i5+N) = (RX1(i,i3,-i5+N))-\
        (((AA(i,i3,-i5+N))*(RX1(i,i3,-i5+N+1)))*(D(i,i3,-i5+N)));
    #define S26(t,i,j,i3,i4,i5,i6) RY2(i,i3,-i5+N) = (RY1(i,i3,-i5+N))*\
        (D(i,i3,-i5+N));
    #define S27(t,i,j,i3,i4,i5,i6) RY2(i,i3,-i5+N) = (RY1(i,i3,-i5+N))-\
        (((AA(i,i3,-i5+N))*(RY1(i,i3,-i5+N+1)))*(D(i,i3,-i5+N)));
```

```
#define S28(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = X0(i3,i5);
#define S29(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = X(i-1,i3,i5);
#define S30(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = (X(i-1,i3,i5))+\
    (RX2(i,i3,i5));
#define S31(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = Y0(i3,i5);
#define S32(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = Y(i-1,i3,i5);
#define S33(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = (Y(i-1,i3,i5))+\
    (RY2(i,i3,i5));
#define S34(t,i1,i2,i3,i4,i5,i6) RXM(i1,0,0) = Tomcatv_RXM_reduce_1(\
    ITACT,N,i1,X0,Y0,XX,YX,XY,YY,A,B,C,AA,DD,PXX,QXX,PYY,QYY,PXY,QXY,\
    RX0,RY0,R,D,RX1,RY1,RX2,RY2,X,Y,RXM,RYM);
#define S35(t,i1,i2,i3,i4,i5,i6) RYM(i1,0,0) = Tomcatv_RYM_reduce_1(
    ITACT,N,i1,X0,Y0,XX,YX,XY,YY,A,B,C,AA,DD,PXX,QXX,PYY,QYY,PXY,QXY,\
    RX0,RY0,R,D,RX1,RY1,RX2,RY2,X,Y,RXM,RYM);
{
    //Domain
//{t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-2== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-3== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-4== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && N-i5-1>= 0 && i5-2>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0 && i3-2>= 0 && N-i3-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-5== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && N-i5-1>= 0 && i5-2>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0 && i3-2>= 0 && N-i3-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-6== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && N-i5-1>= 0 && i5-2>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0 && i3-2>= 0 && N-i3-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-7== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && N-i5-1>= 0 && i5-2>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0 && i3-2>= 0 && N-i3-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-8== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-9== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-10== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-11== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-12== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-13== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i,j,i3,i4,i5,i6|i6-14== 0 && i4== 0 && j== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
```

```
    //{t,i,j,i3,i4,i5,i6|i6-15== 0 && i4== 0 && j== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
        //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-16== 0 && i4== 0 && j== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
        //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && j-2== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i-1>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
        //N-i3-1>= 0 && i5-3>= 0 && N-i5-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-1== 0 && i5-2== 0 && i4== 0 && j-2== 0 &&\
        //t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i3-1>= 0 && i3-2>= 0 &&\
        //ITACT-i>= 0 && i-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && j-2== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i5-3>= 0 && N-i3-1>= 0 && i3-2>= 0 &&\
        //N-i5-1>= 0 && ITACT-i>= 0 && i-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-2== 0 && i5-2== 0 && i4== 0 && j-2== 0 &&\
        //t== 0 && ITACT-4>= 0 && N-4>= 0 && i3-2>= 0 && N-i3-1>= 0 &&\
        //i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-2== 0 && i4== 0 && j-2== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i5-3>= 0 && N-i5-1>= 0 && N-i3-1>= 0 &&\
        //i3-2>= 0 && i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-3== 0 && i5-2== 0 && i4== 0 && j-2== 0 &&\
        //t== 0 && ITACT-4>= 0 && N-4>= 0 && i3-2>= 0 && N-i3-1>= 0 &&\
        //i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-3== 0 && i4== 0 && j-2== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i5-3>= 0 && N-i5-1>= 0 && N-i3-1>= 0 &&\
        //i3-2>= 0 && i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6== 0 && i5-1== 0 && i4== 0 && j-3== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && i3-2>= 0 && N-i3-1>= 0 && i-1>= 0 &&\
        //ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && j-3== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && N-i5-2>= 0 && i5-3>= 0 && N-i3-1>= 0 &&\
        //i3-2>= 0 && i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-1== 0 && i5-1== 0 && i4== 0 && j-3== 0 &&\
        //t== 0 && ITACT-4>= 0 && N-4>= 0 && i3-2>= 0 && N-i3-1>= 0 &&\
        //i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && j-3== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && N-i5-2>= 0 && i5-3>= 0 && N-i3-1>= 0 &&\
        //i3-2>= 0 && i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && j-4== 0 && i-1== 0 && t== 0 &&\
        //ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 && i5-1>= 0 && i3-1>= 0 &&\
        //N-i3>= 0}
//{t,i,j,i3,i4,i5,i6|i6== 0 && -N+i5== 0 && i4== 0 && j-4== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i3-1>= 0 && N-i3>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && i3-1== 0 &&\
    //j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 && i5-1>= 0 &&\
    //i-1>= 0 && ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 &&\
    //-N+i3== 0 && j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 &&\
    //i5-1>= 0 && i-1>= 0 && ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6== 0 &&\
    //i5-1== 0 && i4== 0 && j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 &&\
    //i3-1>= 0 && N-i3>= 0 && i-1>= 0 && ITACT-i>= 0}
    //{t,i,j,i3,i4,i5,i6|i6== 0 && i4== 0 && j-4== 0 && t== 0 && ITACT-4>= 0\
        //&& N-4>= 0 && i-2>= 0 && ITACT-i>= 0 && i3-2>= 0 && N-i3-1>= 0 &&\
        //i5-2>= 0 && N-i5-1>= 0}
    //{t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && j-4== 0 && i-1== 0 &&\
        //t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 && i5-1>= 0 &&\
        //i3-1>= 0 && N-i3>= 0}
//{t,i,j,i3,i4,i5,i6|i6-1== 0 && -N+i5== 0 && i4== 0 && j-4== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i3-1>= 0 && N-i3>= 0 && i-1>= 0 &&\
    //ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && i3-1== 0 &&\
    //j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 && i5-1>= 0 &&\
```

```
//i-1>= 0 && ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 &&\
//-N+i3== 0 && j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && N-i5>= 0 &&\
//i5-1>= 0 && i-1>= 0 && ITACT-i>= 0} || {t,i,j,i3,i4,i5,i6|i6-1== 0 &&\
//i5-1== 0 && i4== 0 && j-4== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 &&\
//i3-1>= 0 && N-i3>= 0 && i-1>= 0 && ITACT-i>= 0}
//{t,i,j,i3,i4,i5,i6|i6-1== 0 && i4== 0 && j-4== 0 && t== 0 &&\
    //ITACT-4>= 0 && N-4>= 0 && i-2>= 0 && ITACT-i>= 0 && i3-2>= 0 &&\
    //N-i3-1>= 0 && i5-2>= 0 && N-i5-1>= 0}
//{t,i1,i2,i3,i4,i5,i6|i6== 0 && -N+i5== 0 && i4== 0 && -N+i3== 0 &&\
    //i2-1== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-i1>= 0 &&\
    //i1-1>= 0}
//{t,i1,i2,i3,i4,i5,i6|i6-1== 0 && -N+i5== 0 && i4== 0 && -N+i3== 0 &&\
    //i2-1== 0 && t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-i1>= 0 &&\
    //i1-1>= 0}
    int c2,c4,c6;
    if(N >= 5){
        for(c4=2;c4 <= N + -1;c4+=1){
            for(c6=2;c6 <= N + -1;c6+=1){
                S0((0),(1),(0),(c4),(0),(c6),(0));
                S1((0),(1),(0),(c4),(0),(c6),(1));
                S2((0),(1),(0),(c4),(0),(c6),(2));
                S3((0),(1),(0),(c4),(0),(c6),(3));
                S4((0),(1),(0),(c4),(0),(c6),(4));
                S5((0),(1),(0),(c4),(0),(c6),(5));
                S6((0),(1),(0),(c4),(0),(c6),(6));
                S7((0),(1),(0),(c4),(0),(c6),(7));
                S8((0),(1),(0),(c4),(0),(c6),(8));
                S9((0),(1),(0),(c4),(0),(c6),(9));
                S10((0),(1),(0),(c4),(0),(c6),(10));
                S11((0),(1),(0),(c4),(0),(c6),(11));
                S12((0),(1),(0),(c4),(0),(c6),(12));
                S13((0),(1),(0),(c4),(0),(c6),(13));
                S14((0),(1),(0),(c4),(0),(c6),(14));
                S15((0),(1),(0),(c4),(0),(c6),(15));
                S16((0),(1),(0),(c4),(0),(c6),(16));
            }
        }
        S34((0),(1),(1),(N),(0),(N),(0));
        S35((0),(1),(1),(N),(0),(N),(1));
        for(c4=2;c4 <= N + -1;c4+=1){
            S18((0),(1),(2),(c4),(0),(2),(1));
            S20((0),(1),(2),(c4),(0),(2),(2));
            S22((0),(1),(2),(c4),(0),(2),(3));
            for(c6=3;c6 <= N + -1;c6+=1){
                S17((0),(1),(2),(c4),(0),(c6),(0));
                S19((0),(1),(2),(c4),(0),(c6),(1));
                S21((0),(1),(2),(c4),(0),(c6),(2));
                S23((0),(1),(2),(c4),(0),(c6),(3));
            }
        }
        for(c4=2;c4 <= N + -1;c4+=1){
            S24((0),(1),(3),(c4),(0),(1),(0));
            S26((0),(1),(3),(c4),(0),(1),(1));
            for(c6=3;c6 <= N + -2;c6+=1){
                S25((0),(1),(3),(c4),(0),(c6),(0));
                S27((0),(1),(3),(c4),(0),(c6),(1));
            }
        }
        for(c6=1;c6 <= N + -1;c6+=1){
            S28((0),(1),(4),(1),(0),(c6),(0));
            S29((0),(1),(4),(1),(0),(c6),(0));
```

```
            S31((0),(1),(4),(1),(0),(c6),(1));
            S32((0),(1),(4),(1),(0),(c6),(1));
        }
        S28((0),(1),(4),(1),(0),(N),(0));
        S29((0),(1),(4),(1),(0),(N),(0));
        S31((0),(1),(4),(1),(0),(N),(1));
        S32((0),(1),(4),(1),(0),(N),(1));
        for(c4=2;c4 <= N + -1;c4+=1){
            S28((0),(1),(4),(c4),(0),(1),(0));
            S29((0),(1),(4),(c4),(0),(1),(0));
            S31((0),(1),(4),(c4),(0),(1),(1));
            S32((0),(1),(4),(c4),(0),(1),(1));
            for(c6=2;c6 <= N + -1;c6+=1){
                S28((0),(1),(4),(c4),(0),(c6),(0));
                S31((0),(1),(4),(c4),(0),(c6),(1));
            }
            S28((0),(1),(4),(c4),(0),(N),(0));
            S29((0),(1),(4),(c4),(0),(N),(0));
            S31((0),(1),(4),(c4),(0),(N),(1));
            S32((0),(1),(4),(c4),(0),(N),(1));
        }
        for(c6=1;c6 <= N;c6+=1){
            S28((0),(1),(4),(N),(0),(c6),(0));
            S29((0),(1),(4),(N),(0),(c6),(0));
            S31((0),(1),(4),(N),(0),(c6),(1));
            S32((0),(1),(4),(N),(0),(c6),(1));
        }
    }
    if(N >= 5){
        for(c2=2;c2 <= ITACT;c2+=1){
            for(c4=2;c4 <= N + -1;c4+=1){
                for(c6=2;c6 <= N + -1;c6+=1){
                    S0((0),(c2),(0),(c4),(0),(c6),(0));
                    S1((0),(c2),(0),(c4),(0),(c6),(1));
                    S2((0),(c2),(0),(c4),(0),(c6),(2));
                    S3((0),(c2),(0),(c4),(0),(c6),(3));
                    S4((0),(c2),(0),(c4),(0),(c6),(4));
                    S5((0),(c2),(0),(c4),(0),(c6),(5));
                    S6((0),(c2),(0),(c4),(0),(c6),(6));
                    S7((0),(c2),(0),(c4),(0),(c6),(7));
                    S8((0),(c2),(0),(c4),(0),(c6),(8));
                    S9((0),(c2),(0),(c4),(0),(c6),(9));
                    S10((0),(c2),(0),(c4),(0),(c6),(10));
                    S11((0),(c2),(0),(c4),(0),(c6),(11));
                    S12((0),(c2),(0),(c4),(0),(c6),(12));
                    S13((0),(c2),(0),(c4),(0),(c6),(13));
                    S14((0),(c2),(0),(c4),(0),(c6),(14));
                    S15((0),(c2),(0),(c4),(0),(c6),(15));
                    S16((0),(c2),(0),(c4),(0),(c6),(16));
                }
            }
            S34((0),(c2),(1),(N),(0),(N),(0));
            S35((0),(c2),(1),(N),(0),(N),(1));
            for(c4=2;c4 <= N + -1;c4+=1){
                S18((0),(c2),(2),(c4),(0),(2),(1));
                S20((0),(c2),(2),(c4),(0),(2),(2));
                S22((0),(c2),(2),(c4),(0),(2),(3));
                for(c6=3;c6 <= N + -1;c6+=1){
                    S17((0),(c2),(2),(c4),(0),(c6),(0));
                    S19((0),(c2),(2),(c4),(0),(c6),(1));
                    S21((0),(c2),(2),(c4),(0),(c6),(2));
```

```
                S23((0),(c2),(2),(c4),(0),(c6),(3));
            }
        }
        for(c4=2;c4 <= N + -1;c4+=1){
            S24((0),(c2),(3),(c4),(0),(1),(0));
            S26((0),(c2),(3),(c4),(0),(1),(1));
            for(c6=3;c6 <= N + -2;c6+=1){
                S25((0),(c2),(3),(c4),(0),(c6),(0));
                S27((0),(c2),(3),(c4),(0),(c6),(1));
            }
        }
        for(c6=1;c6 <= N;c6+=1){
            S29((0),(c2),(4),(1),(0),(c6),(0));
            S32((0),(c2),(4),(1),(0),(c6),(1));
        }
        for(c4=2;c4 <= N + -1;c4+=1){
            S29((0),(c2),(4),(c4),(0),(1),(0));
            S32((0),(c2),(4),(c4),(0),(1),(1));
            for(c6=2;c6 <= N + -1;c6+=1){
                S30((0),(c2),(4),(c4),(0),(c6),(0));
                S33((0),(c2),(4),(c4),(0),(c6),(1));
            }
            S29((0),(c2),(4),(c4),(0),(N),(0));
            S32((0),(c2),(4),(c4),(0),(N),(1));
        }
        for(c6=1;c6 <= N;c6+=1){
            S29((0),(c2),(4),(N),(0),(c6),(0));
            S32((0),(c2),(4),(N),(0),(c6),(1));
        }
    }
}
if(N == 4){
    for(c4=2;c4 <= 3;c4+=1){
        for(c6=2;c6 <= 3;c6+=1){
            S0((0),(1),(0),(c4),(0),(c6),(0));
            S1((0),(1),(0),(c4),(0),(c6),(1));
            S2((0),(1),(0),(c4),(0),(c6),(2));
            S3((0),(1),(0),(c4),(0),(c6),(3));
            S4((0),(1),(0),(c4),(0),(c6),(4));
            S5((0),(1),(0),(c4),(0),(c6),(5));
            S6((0),(1),(0),(c4),(0),(c6),(6));
            S7((0),(1),(0),(c4),(0),(c6),(7));
            S8((0),(1),(0),(c4),(0),(c6),(8));
            S9((0),(1),(0),(c4),(0),(c6),(9));
            S10((0),(1),(0),(c4),(0),(c6),(10));
            S11((0),(1),(0),(c4),(0),(c6),(11));
            S12((0),(1),(0),(c4),(0),(c6),(12));
            S13((0),(1),(0),(c4),(0),(c6),(13));
            S14((0),(1),(0),(c4),(0),(c6),(14));
            S15((0),(1),(0),(c4),(0),(c6),(15));
            S16((0),(1),(0),(c4),(0),(c6),(16));
        }
    }
    S34((0),(1),(1),(4),(0),(4),(0));
    S35((0),(1),(1),(4),(0),(4),(1));
    for(c4=2;c4 <= 3;c4+=1){
        S18((0),(1),(2),(c4),(0),(2),(1));
        S20((0),(1),(2),(c4),(0),(2),(2));
        S22((0),(1),(2),(c4),(0),(2),(3));
        S17((0),(1),(2),(c4),(0),(3),(0));
        S19((0),(1),(2),(c4),(0),(3),(1));
```

```
            S21((0),(1),(2),(c4),(0),(3),(2));
            S23((0),(1),(2),(c4),(0),(3),(3));
        }
        for(c4=2;c4 <= 3;c4+=1){
            S24((0),(1),(3),(c4),(0),(1),(0));
            S26((0),(1),(3),(c4),(0),(1),(1));
        }
        for(c6=1;c6 <= 4;c6+=1){
            S28((0),(1),(4),(1),(0),(c6),(0));
            S29((0),(1),(4),(1),(0),(c6),(0));
            S31((0),(1),(4),(1),(0),(c6),(1));
            S32((0),(1),(4),(1),(0),(c6),(1));
        }
        for(c4=2;c4 <= 3;c4+=1){
            S28((0),(1),(4),(c4),(0),(1),(0));
            S29((0),(1),(4),(c4),(0),(1),(0));
            S31((0),(1),(4),(c4),(0),(1),(1));
            S32((0),(1),(4),(c4),(0),(1),(1));
            for(c6=2;c6 <= 3;c6+=1){
                S28((0),(1),(4),(c4),(0),(c6),(0));
                S31((0),(1),(4),(c4),(0),(c6),(1));
            }
            S28((0),(1),(4),(c4),(0),(4),(0));
            S29((0),(1),(4),(c4),(0),(4),(0));
            S31((0),(1),(4),(c4),(0),(4),(1));
            S32((0),(1),(4),(c4),(0),(4),(1));
        }
        S28((0),(1),(4),(4),(0),(1),(0));
        S29((0),(1),(4),(4),(0),(1),(0));
        S31((0),(1),(4),(4),(0),(1),(1));
        S32((0),(1),(4),(4),(0),(1),(1));
        for(c6=2;c6 <= 4;c6+=1){
            S28((0),(1),(4),(4),(0),(c6),(0));
            S29((0),(1),(4),(4),(0),(c6),(0));
            S31((0),(1),(4),(4),(0),(c6),(1));
            S32((0),(1),(4),(4),(0),(c6),(1));
        }
    }
    if(N == 4){
        for(c2=2;c2 <= ITACT;c2+=1){
            for(c4=2;c4 <= 3;c4+=1){
                for(c6=2;c6 <= 3;c6+=1){
                    S0((0),(c2),(0),(c4),(0),(c6),(0));
                    S1((0),(c2),(0),(c4),(0),(c6),(1));
                    S2((0),(c2),(0),(c4),(0),(c6),(2));
                    S3((0),(c2),(0),(c4),(0),(c6),(3));
                    S4((0),(c2),(0),(c4),(0),(c6),(4));
                    S5((0),(c2),(0),(c4),(0),(c6),(5));
                    S6((0),(c2),(0),(c4),(0),(c6),(6));
                    S7((0),(c2),(0),(c4),(0),(c6),(7));
                    S8((0),(c2),(0),(c4),(0),(c6),(8));
                    S9((0),(c2),(0),(c4),(0),(c6),(9));
                    S10((0),(c2),(0),(c4),(0),(c6),(10));
                    S11((0),(c2),(0),(c4),(0),(c6),(11));
                    S12((0),(c2),(0),(c4),(0),(c6),(12));
                    S13((0),(c2),(0),(c4),(0),(c6),(13));
                    S14((0),(c2),(0),(c4),(0),(c6),(14));
                    S15((0),(c2),(0),(c4),(0),(c6),(15));
                    S16((0),(c2),(0),(c4),(0),(c6),(16));
                }
            }
```

```
                S34((0),(c2),(1),(4),(0),(4),(0));
                S35((0),(c2),(1),(4),(0),(4),(1));
                for(c4=2;c4 <= 3;c4+=1){
                    S18((0),(c2),(2),(c4),(0),(2),(1));
                    S20((0),(c2),(2),(c4),(0),(2),(2));
                    S22((0),(c2),(2),(c4),(0),(2),(3));
                    S17((0),(c2),(2),(c4),(0),(3),(0));
                    S19((0),(c2),(2),(c4),(0),(3),(1));
                    S21((0),(c2),(2),(c4),(0),(3),(2));
                    S23((0),(c2),(2),(c4),(0),(3),(3));
                }
                for(c4=2;c4 <= 3;c4+=1){
                    S24((0),(c2),(3),(c4),(0),(1),(0));
                    S26((0),(c2),(3),(c4),(0),(1),(1));
                }
                for(c6=1;c6 <= 4;c6+=1){
                    S29((0),(c2),(4),(1),(0),(c6),(0));
                    S32((0),(c2),(4),(1),(0),(c6),(1));
                }
                for(c4=2;c4 <= 3;c4+=1){
                    S29((0),(c2),(4),(c4),(0),(1),(0));
                    S32((0),(c2),(4),(c4),(0),(1),(1));
                    for(c6=2;c6 <= 3;c6+=1){
                        S30((0),(c2),(4),(c4),(0),(c6),(0));
                        S33((0),(c2),(4),(c4),(0),(c6),(1));
                    }
                    S29((0),(c2),(4),(c4),(0),(4),(0));
                    S32((0),(c2),(4),(c4),(0),(4),(1));
                }
                for(c6=1;c6 <= 4;c6+=1){
                    S29((0),(c2),(4),(4),(0),(c6),(0));
                    S32((0),(c2),(4),(4),(0),(c6),(1));
                }
            }
        }
    }
#undef S0
#undef S1
#undef S2
#undef S3
#undef S4
#undef S5
#undef S6
#undef S7
#undef S8
#undef S9
#undef S10
#undef S11
#undef S12
#undef S13
#undef S14
#undef S15
#undef S16
#undef S17
#undef S18
#undef S19
#undef S20
#undef S21
#undef S22
#undef S23
#undef S24
```

```
#undef S25
#undef S26
#undef S27
#undef S28
#undef S29
#undef S30
#undef S31
#undef S32
#undef S33
#undef S34
#undef S35
//Memory Free
free(XX);
free(YX);
free(XY);
free(YY);
free(A);
free(B);
free(C);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(AA[mz1][mz2]);
    }
    free(AA[mz1]);
}
free(AA);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(DD[mz1][mz2]);
    }
    free(DD[mz1]);
}
free(DD);
free(PXX);
free(QXX);
free(PYY);
free(QYY);
free(PXY);
free(QXY);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(RX0[mz1][mz2]);
    }
    free(RX0[mz1]);
}
free(RX0);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(RY0[mz1][mz2]);
    }
    free(RY0[mz1]);
}
free(RY0);
free(R);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(D[mz1][mz2]);
    }
    free(D[mz1]);
}
free(D);
```

```
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N; mz2++){
            free(RX1[mz1][mz2]);
        }
        free(RX1[mz1]);
    }
    free(RX1);
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N; mz2++){
            free(RY1[mz1][mz2]);
        }
        free(RY1[mz1]);
    }
    free(RY1);
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N; mz2++){
            free(RX2[mz1][mz2]);
        }
        free(RX2[mz1]);
    }
    free(RX2);
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N; mz2++){
            free(RY2[mz1][mz2]);
        }
        free(RY2[mz1]);
    }
    free(RY2);
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N+1; mz2++){
            free(X[mz1][mz2]);
        }
        free(X[mz1]);
    }
    free(X);
    for (mz1=0; mz1<1; mz1++){
        for (mz2=0; mz2<N+1; mz2++){
            free(Y[mz1][mz2]);
        }
        free(Y[mz1]);
    }
    free(Y);
}
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double* XX,double* YX,double* XY,double* YY,double* A,
    double* B,double* C,double*** AA,double*** DD,double* PXX,double* QXX,
    double* PYY,double* QYY,double* PXY,double* QXY,double*** RX0,
    double*** RY0,double* R,double*** D,double*** RX1,double*** RY1,
    double*** RX2,double*** RY2,double*** X,double*** Y,double*** RXM,
    double*** RYM){
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RX0(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        //Domain
    //{t,k,l|-t_p+t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-t_p>= 0 && t_p-1>=
0 && N-l-1>= 0 && l-2>= 0 && k-2>= 0 && N-k-1>= 0}
        int c2,c3;
        for(c2=2;c2 <= N + -1;c2+=1){
            for(c3=2;c3 <= N + -1;c3+=1){
                S0((t_p),(c2),(c3));
```

```
            }
        }
    }
    #undef S0
    return reduceVar;
}
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double* XX,double* YX,double* XY,double* YY,double* A,
    double* B,double* C,double*** AA,double*** DD,double* PXX,double* QXX,
    double* PYY,double* QYY,double* PXY,double* QXY,double*** RX0,
    double*** RY0,double* R,double*** D,double*** RX1,double*** RY1,
    double*** RX2,double*** RY2,double*** X,double*** Y,double*** RXM,
    double*** RYM){
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RY0(t,k,l));
    reduceVar = max(reduceVar,__temp__); }
    {
        //Domain
    //{t,k,l|-t_p+t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-t_p>= 0 && t_p-1>=
0 && N-l-1>= 0 && l-2>= 0 && k-2>= 0 && N-k-1>= 0}
        int c2,c3;
        for(c2=2;c2 <= N + -1;c2+=1){
            for(c3=2;c3 <= N + -1;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}
#undef X0
#undef Y0
#undef XX
#undef YX
#undef XY
#undef YY
#undef A
#undef B
#undef C
#undef AA
#undef DD
#undef PXX
#undef QXX
#undef PYY
#undef QYY
#undef PXY
#undef QXY
#undef RX0
#undef RY0
#undef R
#undef D
#undef RX1
#undef RY1
#undef RX2
#undef RY2
#undef X
#undef Y
#undef RXM
#undef RYM
//Postamble
#undef MAX
```

```
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Generated code for Tomcatv: fewer arrays

```
//Preamble
// This file is generated from test alphabets program by code generator in
    // alphaz
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include "external_functions.h"
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

//Local Function Declarations
double Tomcatv_RXM_reduce_1(long,long,int,double**,double**,double***,
    double***,double***,double***,double***,double***);
double Tomcatv_RYM_reduce_1(long,long,int,double**,double**,double***,
    double***,double***,double***,double***,double***);
//Memory Macros
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define D(t,i,j) D[MOD(t, 1)][i][j]
```

```
#define RX(t,i,j) RX[MOD(t, 1)][i][j]
#define RY(t,i,j) RY[MOD(t, 1)][i][j]
#define X(t,i,j) X[MOD(t, 1)][i][j]
#define Y(t,i,j) Y[MOD(t, 1)][i][j]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]
//Function bodies
void Tomcatv(long ITACT,long N,double** X0,double** Y0,double*** RXM,
    double*** RYM){
    // Parameter checking
    if (!((ITACT-4>= 0 && N-4>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
    //Loop indices for malloc
    int mz1, mz2, mz3;
    double*** D = malloc(sizeof(double**)*(1));
    if (D == NULL) {
        printf("Failed to allocate memory for D : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        D[mz1] = malloc(sizeof(double*)*(N-1));
        if (D[mz1] == NULL) {
            printf("Failed to allocate memory for D : size=%ld\n",
                (N-1) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N-1; mz2++){
            D[mz1][mz2] = malloc(sizeof(double)*(N-1));
            if (D[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for D : size=%ld\n",
                    (N-1) * sizeof(double));
                exit(-1);
            }
        }
    }
    double*** RX = malloc(sizeof(double**)*(1));
    if (RX == NULL) {
        printf("Failed to allocate memory for RX : size=%ld\n",
            (1) * sizeof(double**));
        exit(-1);
    }
    for (mz1=0; mz1<1; mz1++){
        RX[mz1] = malloc(sizeof(double*)*(N-1));
        if (RX[mz1] == NULL) {
            printf("Failed to allocate memory for RX : size=%ld\n",
                (N-1) * sizeof(double*));
            exit(-1);
        }
        for (mz2=0; mz2<N-1; mz2++){
            RX[mz1][mz2] = malloc(sizeof(double)*(N-1));
            if (RX[mz1][mz2] == NULL) {
                printf("Failed to allocate memory for RX : size=%ld\n",
                    (N-1) * sizeof(double));
                exit(-1);
            }
        }
    }
    double*** RY = malloc(sizeof(double**)*(1));
```

```
if (RY == NULL) {
    printf("Failed to allocate memory for RY : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    RY[mz1] = malloc(sizeof(double*)*(N-1));
    if (RY[mz1] == NULL) {
        printf("Failed to allocate memory for RY : size=%ld\n",
            (N-1) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N-1; mz2++){
        RY[mz1][mz2] = malloc(sizeof(double)*(N-1));
        if (RY[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for RY : size=%ld\n",
                (N-1) * sizeof(double));
            exit(-1);
        }
    }
}
double*** X = malloc(sizeof(double**)*(1));
if (X == NULL) {
    printf("Failed to allocate memory for X : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    X[mz1] = malloc(sizeof(double*)*(N));
    if (X[mz1] == NULL) {
        printf("Failed to allocate memory for X : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        X[mz1][mz2] = malloc(sizeof(double)*(N));
        if (X[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for X : size=%ld\n",
                (N) * sizeof(double));
            exit(-1);
        }
    }
}
double*** Y = malloc(sizeof(double**)*(1));
if (Y == NULL) {
    printf("Failed to allocate memory for Y : size=%ld\n",
        (1) * sizeof(double**));
    exit(-1);
}
for (mz1=0; mz1<1; mz1++){
    Y[mz1] = malloc(sizeof(double*)*(N));
    if (Y[mz1] == NULL) {
        printf("Failed to allocate memory for Y : size=%ld\n",
            (N) * sizeof(double*));
        exit(-1);
    }
    for (mz2=0; mz2<N; mz2++){
        Y[mz1][mz2] = malloc(sizeof(double)*(N));
        if (Y[mz1][mz2] == NULL) {
            printf("Failed to allocate memory for Y : size=%ld\n",
                (N) * sizeof(double));
```

```
            exit(-1);
        }
    }
}
#define S0(t,i,j) D(t,i,j) = (((j-1== 0))?(1)/((((0.5)*(pow((X(t-1,i+1,\
    j))-(X(t-1,i-1,j)),2)))+((2)*(pow((Y(t-1,i+1,j))-(Y(t-1,i-1,j)),\
    2))))+(((0.25)*(pow((X(t-1,i,j+1))-(X(t-1,i,j-1)),2)))+(((2.0)/\
    (0.98))*(pow((Y(t-1,i,j+1))-(Y(t-1,i,j-1)),2))))):((1)/((((2)*\
    (((0.25)*(pow((X(t-1,i+1,j))-(X(t-1,i-1,j)),2)))+(pow((Y(t-1,i+1,\
    j))-(Y(t-1,i-1,j)),2))))+((((0.25)*(pow((X(t-1,i,j+1))-(X(t-1,i,\
    j-1)),2)))+(pow((Y(t-1,i,j+1))-(Y(t-1,i,j-1)),2)))*((2.0)/(0.98))))-\
    (((((0.25)*(pow((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1)),2)))+(pow((Y(t-1,\
    i+1,j-1))-(Y(t-1,i-1,j-1)),2)))*(((0.25)*(pow((X(t-1,i+1,j))-(X(t-1,\
    i-1,j)),2)))+(pow((Y(t-1,i+1,j))-(Y(t-1,i-1,j)),2))))*(D(t,i,\
    j-1))))));
#define S1(t,i,j) RX(t,i,j) = (((j-1== 0))?(((((((0.25)*((X(t-1,i,j+1))-\
    (X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
    (Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-\
    ((2.0)*(X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-\
    ((2.0)*(X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,\
    j+1))-(X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1)))))+\
    ((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))))*(((((((0.25)*((X(t-1,i,j+2))-(X(t-1,i,j))))*((X(t-1,i,\
    j+2))-(X(t-1,i,j))))+(((Y(t-1,i,j+2))-(Y(t-1,i,j)))*((Y(t-1,i,j+2))-\
    (Y(t-1,i,j)))))*(((X(t-1,i+1,j+1))-((2.0)*(X(t-1,i,j+1))))+(X(t-1,\
    i-1,j+1))))+(((((0.25)*((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,\
    i+1,j+1))-(X(t-1,i-1,j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*\
    ((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))))*(((X(t-1,i,j+2))-((2.0)*\
    (X(t-1,i,j+1))))+(X(t-1,i,j)))))-(((((0.125)*((X(t-1,i+1,j+1))-\
    (X(t-1,i-1,j+1))))*((X(t-1,i,j+2))-(X(t-1,i,j))))+(((Y(t-1,i+1,\
    j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i,j+2))-(Y(t-1,i,j)))))*((((X(t-1,\
    i+1,j+2))-(X(t-1,i+1,j)))-(X(t-1,i-1,j+2)))+(X(t-1,i-1,j))))))*\
    (D(t,i,j))):((((-N+j+2== 0))?(((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1)))))+\
    ((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
    (((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*\
    (D(t,i,j-1))))*(D(t,i,N-2)):((((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
```

```
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1)))))+\
    (((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
    (((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*\
    (D(t,i,j-1))))+((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,\
    i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,\
    i+1,j))-(Y(t-1,i-1,j)))))*(((((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1)))))+\
    (((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,i+1,j+1))-(X(t-1,i-1,\
    j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i+1,j+1))-\
    (Y(t-1,i-1,j+1))))))*(D(t,i,j)))))*(D(t,i,j))))));
#define S2(t,i,j) RY(t,i,j) = (((j-1== 0))?(((((((0.25)*((X(t-1,i,j+1))-\
    (X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
    (Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-\
    ((2.0)*(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-\
    ((2.0)*(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,\
    j+1))-(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
    ((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))))*(((((((0.25)*((X(t-1,i,j+2))-(X(t-1,i,j))))*((X(t-1,i,\
    j+2))-(X(t-1,i,j))))+(((Y(t-1,i,j+2))-(Y(t-1,i,j)))*((Y(t-1,i,j+2))-\
    (Y(t-1,i,j)))))*(((Y(t-1,i+1,j+1))-((2.0)*(Y(t-1,i,j+1))))+(Y(t-1,\
    i-1,j+1))))+(((((0.25)*((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,\
```

```
i+1,j+1))-(X(t-1,i-1,j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*\
((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))))*(((Y(t-1,i,j+2))-((2.0)*\
(Y(t-1,i,j+1))))+(Y(t-1,i,j)))))-(((((0.125)*((X(t-1,i+1,j+1))-\
(X(t-1,i-1,j+1))))*((X(t-1,i,j+2))-(X(t-1,i,j))))+(((Y(t-1,i+1,\
j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i,j+2))-(Y(t-1,i,j)))))*((((Y(t-1,\
i+1,j+2))-(Y(t-1,i+1,j)))-(Y(t-1,i-1,j+2)))+(Y(t-1,i-1,j))))))*\
(D(t,i,j))):(((-N+j+2== 0))?(((((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-((2.0)*\
(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-((2.0)*\
(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,j+1))-\
(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1))))+\
(((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
j-2)))))*(((Y(t-1,i+1,j-1))-((2.0)*(Y(t-1,i,j-1))))+(Y(t-1,i-1,\
j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((Y(t-1,i,j))-((2.0)*(Y(t-1,\
i,j-1))))+(Y(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
(Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((Y(t-1,i+1,j))-\
(Y(t-1,i+1,j-2)))-(Y(t-1,i-1,j)))+(Y(t-1,i-1,j-2)))))*((((0.25)*\
((((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1))))+\
(((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j))))))*\
(D(t,i,j-1))))+((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,\
i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,\
i+1,j))-(Y(t-1,i-1,j)))))*(D(t,i,j)))*(((((((((0.25)*((X(t-1,i,j+1))-\
(X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
(Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,\
```

```
        j+1))-(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
        ((((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
        i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
        j-2)))))*(((Y(t-1,i+1,j-1))-((2.0)*(Y(t-1,i,j-1))))+(Y(t-1,i-1,\
        j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
        j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
        ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((Y(t-1,i,j))-((2.0)*(Y(t-1,\
        i,j-1))))+(Y(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
        i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
        (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((Y(t-1,i+1,j))-\
        (Y(t-1,i+1,j-2)))-(Y(t-1,i-1,j)))+(Y(t-1,i-1,j-2)))))*((((0.25)*\
        ((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,i+1,j+1))-(X(t-1,i-1,\
        j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i+1,j+1))-\
        (Y(t-1,i-1,j+1))))))*(D(t,i,j)))))))));
    #define S3(t,i,j) X(t,i,j) = (((t== 0) || (-N+j+1== 0) || (i== 0) ||\
        (-N+i+1== 0) || (j== 0))?X0(i,j):((X(t-1,i,j))+(RX(t,i,j))));
    #define S4(t,i,j) Y(t,i,j) = (((t== 0) || (-N+j+1== 0) || (i== 0) ||\
        (-N+i+1== 0) || (j== 0))?Y0(i,j):((Y(t-1,i,j))+(RY(t,i,j))));
    #define S5(t,i1,i2) RXM(t,0,0) = Tomcatv_RXM_reduce_1(ITACT,N,t,X0,Y0,D,\
        RX,RY,X,Y,RXM,RYM);
    #define S6(t,i1,i2) RYM(t,0,0) = Tomcatv_RYM_reduce_1(ITACT,N,t,X0,Y0,D,\
        RX,RY,X,Y,RXM,RYM);
    {
        //Domain
    //{t,i,j|ITACT-4>= 0 && N-j-2>= 0 && i-1>= 0 && ITACT-t>= 0 &&\
        t-1>= 0 && j-1>= 0 && N-i-2>= 0 && N-4>= 0}
    //{t,i,j|ITACT-4>= 0 && N-j-2>= 0 && N-i-2>= 0 && i-1>= 0 &&\
        ITACT-t>= 0 && j-1>= 0 && t-1>= 0 && N-4>= 0}
    //{t,i,j|ITACT-4>= 0 && N-j-2>= 0 && N-i-2>= 0 && i-1>= 0 &&\
        ITACT-t>= 0 && j-1>= 0 && t-1>= 0 && N-4>= 0}
//{t,i,j|t== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 && j>= 0 && i>= 0 &&\
    N-i-1>= 0} || {t,i,j|i== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 &&\
    j>= 0 && t>= 0 && ITACT-t>= 0} || {t,i,j|j== 0 && ITACT-4>= 0 &&\
    N-4>= 0 && i>= 0 && N-i-1>= 0 && t>= 0 && ITACT-t>= 0} ||\
    {t,i,j|-N+i+1== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 && j>= 0 &&\
    t>= 0 && ITACT-t>= 0} || {t,i,j|-N+j+1== 0 && ITACT-4>= 0 && N-4>= 0 &&\
    i>= 0 && N-i-1>= 0 && t>= 0 && ITACT-t>= 0} || {t,i,j|ITACT-4>= 0 &&\
    N-4>= 0 && t-1>= 0 && ITACT-t>= 0 && i-1>= 0 && N-i-2>= 0 && j-1>= 0 &&\
    N-j-2>= 0}
//{t,i,j|t== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 && j>= 0 && i>= 0 &&\
    N-i-1>= 0} || {t,i,j|i== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 &&\
    j>= 0 && t>= 0 && ITACT-t>= 0} || {t,i,j|j== 0 && ITACT-4>= 0 &&\
    N-4>= 0 && i>= 0 && N-i-1>= 0 && t>= 0 && ITACT-t>= 0} ||\
    {t,i,j|-N+i+1== 0 && ITACT-4>= 0 && N-4>= 0 && N-j-1>= 0 && j>= 0 &&\
    t>= 0 && ITACT-t>= 0} || {t,i,j|-N+j+1== 0 && ITACT-4>= 0 && N-4>= 0 &&\
    i>= 0 && N-i-1>= 0 && t>= 0 && ITACT-t>= 0} || {t,i,j|ITACT-4>= 0 &&\
    N-4>= 0 && t-1>= 0 && ITACT-t>= 0 && i-1>= 0 && N-i-2>= 0 && j-1>= 0 &&\
    N-j-2>= 0}
    //{t,i1,i2|-N+i2== 0 && -N+i1== 0 && ITACT-4>= 0 && N-4>= 0 &&\
        ITACT-t>= 0 && t-1>= 0}
    //{t,i1,i2|-N+i2== 0 && -N+i1== 0 && ITACT-4>= 0 && N-4>= 0 &&\
        ITACT-t>= 0 && t-1>= 0}
        int c1,c2,c3;
        for(c2=0;c2 <= N + -1;c2+=1){
            for(c3=0;c3 <= N + -1;c3+=1){
                S3((0),(c2),(c3));
                S4((0),(c2),(c3));
            }
        }
        for(c1=1;c1 <= ITACT;c1+=1){
            for(c3=0;c3 <= N + -1;c3+=1){
```

```
            S3((c1),(0),(c3));
            S4((c1),(0),(c3));
        }
        for(c2=1;c2 <= N + -2;c2+=1){
            S3((c1),(c2),(0));
            S4((c1),(c2),(0));
            for(c3=1;c3 <= N + -2;c3+=1){
                S0((c1),(c2),(c3));
                S1((c1),(c2),(c3));
                S2((c1),(c2),(c3));
                S3((c1),(c2),(c3));
                S4((c1),(c2),(c3));
            }
            S3((c1),(c2),(N + -1));
            S4((c1),(c2),(N + -1));
        }
        for(c3=0;c3 <= N + -1;c3+=1){
            S3((c1),(N + -1),(c3));
            S4((c1),(N + -1),(c3));
        }
        S5((c1),(N),(N));
        S6((c1),(N),(N));
    }
}
#undef S0
#undef S1
#undef S2
#undef S3
#undef S4
#undef S5
#undef S6
//Memory Free
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N-1; mz2++){
        free(D[mz1][mz2]);
    }
    free(D[mz1]);
}
free(D);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N-1; mz2++){
        free(RX[mz1][mz2]);
    }
    free(RX[mz1]);
}
free(RX);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N-1; mz2++){
        free(RY[mz1][mz2]);
    }
    free(RY[mz1]);
}
free(RY);
for (mz1=0; mz1<1; mz1++){
    for (mz2=0; mz2<N; mz2++){
        free(X[mz1][mz2]);
    }
    free(X[mz1]);
}
free(X);
for (mz1=0; mz1<1; mz1++){
```

```
        for (mz2=0; mz2<N; mz2++){
            free(Y[mz1][mz2]);
        }
        free(Y[mz1]);
    }
    free(Y);
}
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double*** D,double*** RX,double*** RY,double*** X,
    double*** Y,double*** RXM,double*** RYM){
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RX(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        //Domain
    //{t,k,l|-t_p+t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-t_p>= 0 &&\
        t_p-1>= 0 && N-l-2>= 0 && l-1>= 0 && k-1>= 0 && N-k-2>= 0}
        int c2,c3;
        for(c2=1;c2 <= N + -2;c2+=1){
            for(c3=1;c3 <= N + -2;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double*** D,double*** RX,double*** RY,double*** X,
    double*** Y,double*** RXM,double*** RYM){
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RY(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        //Domain
    //{t,k,l|-t_p+t== 0 && ITACT-4>= 0 && N-4>= 0 && ITACT-t_p>= 0 &&\
        t_p-1>= 0 && N-l-2>= 0 && l-1>= 0 && k-1>= 0 && N-k-2>= 0}
        int c2,c3;
        for(c2=1;c2 <= N + -2;c2+=1){
            for(c3=1;c3 <= N + -2;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}
#undef X0
#undef Y0
#undef D
#undef RX
#undef RY
#undef X
#undef Y
#undef RXM
#undef RYM
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
```

```
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for one-dimensional stencil: untiled

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define NMAX (131071984L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION STATIC_ARY_ARY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2) * N + i]
#define Final(i,i1) Final[MOD(i, 1)][i1]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i) Initial[i]
```

```
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]

#endif
//Function bodies
void oneD(long T,long N,double* Initial,double** Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*N * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
    static double Intermediate[2][NMAX];
#endif

    #define S0(t,i) Intermediate(t,i) = Initial(i);
    #define S1(t,i) Intermediate(t,i) = Intermediate(t-1,i);
    #define S2(t,i) Intermediate(t,i) = Intermediate(t-1,i);
    #define S3(t,i) Intermediate(t,i) = ((((2)*(Intermediate(t-1,i)))+\
        (Intermediate(t-1,i-1)))+(Intermediate(t-1,i+1)))/(4.0);
    #define S4(t,i) Final(T,i) = Intermediate(T,i);
    {
        int c1,c2;
        for(c2=0;c2 <= N + -1;c2+=1){
            S0((0),(c2));
        }
        for(c1=1;c1 <= T + -1;c1+=1){
            S1((c1),(0));
            for(c2=1;c2 <= N + -2;c2+=1){
                S3((c1),(c2));
            }
            S2((c1),(N + -1));
        }
        S1((T),(0));
        S4((T),(0));
        for(c2=1;c2 <= N + -2;c2+=1){
            S3((T),(c2));
            S4((T),(c2));
        }
        S2((T),(N + -1));
        S4((T),(N + -1));
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
```

```
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for one-dimensional stencil: tiling

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define NMAX (131071984L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION STATIC_ARY_ARY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2) * N + i]
#define Final(i,i1) Final[MOD(i, 1)][i1]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]
```

```
#endif

//Function bodies
void oneD(long T,long N,int ts1,int ts2,double* Initial,double** Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*N * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
    static double Intermediate[2][NMAX];
#endif

    #define S0(t,i) Intermediate(t,-t+i) = Initial(-t+i);
    #define S1(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
    #define S2(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
    #define S3(t,i) Intermediate(t,-t+i) = ((((2)*(Intermediate(t-1,-t+i)))+\
        (Intermediate(t-1,-t+i-1)))+(Intermediate(t-1,-t+i+1)))/(4.0);
    #define S4(i,i1) Final(T,i1-T) = Intermediate(T,i1-T);
    {
        int c1,c2,ti1,ti2;
        for(ti1=ceild(-ts1 + 1, ts1)*ts1;ti1 <= T;ti1+=ts1){
            for(ti2=ceild(min(0,min(ti1,T)) + -ts2 + 1, ts2)*ts2;ti2 <=
                max(N + -1,max(N + ti1 + -1,N + T + -1)) +
                max(0,ts1 + -1);ti2+=ts2){
                /** guard that isolates selected statements for generic point
                    loops **//** point loops with optimizations **/
                if((0 < ti1) && (ti1 + ts1 + -1 < ti2) && (ti1 < T + -ts1 +
                    1) && (ti2 < ti1 + N + -ts2 + 0)){
                    /** full-tile guard **/
                    if((1 <= ti1) && (ti1 <= T + -ts1 + 0) && (ti1 + ts1 +
                        0 <= ti2) && (ti2 <= ti1 + N + -ts2 + -1)){
                        for(c1=ti1;c1 <= ti1 + ts1 + -1;c1+=1){
                            for(c2=ti2;c2 <= ti2 + ts2 + -1;c2+=1){
                                S3((c1),(c2));
                            }
                        }
                    } else {
                        for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
                            T + -1);c1+=1){
                            for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
                                N + c1 + -2);c2+=1){
                                S3((c1),(c2));
                            }
                        }
                    }
                } else {
                    {
                        for(c1=max(ti1,0);c1 <= min(ti1 + ts1 + -1,0);c1+=1){
                            for(c2=max(ti2,0);c2 <= min(ti2 + ts2 + -1,
                                N + -1);c2+=1){
                                S0((0),(c2));
                            }
                        }
                        for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
                            T + -1);c1+=1){
                            for(c2=max(ti2,c1);c2 <= min(ti2 + ts2 + -1,
                                c1);c2+=1){
```

```
                                S1((c1),(c1));
                            }
                            for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
                                N + c1 + -2);c2+=1){
                                S3((c1),(c2));
                            }
                            for(c2=max(ti2,N + c1 + -1);c2 <= min(ti2 + ts2 +
                                -1,N + c1 + -1);c2+=1){
                                S2((c1),(N + c1 + -1));
                            }
                        }
                        for(c1=max(ti1,T);c1 <= min(ti1 + ts1 + -1,T);c1+=1){
                            for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,
                                T);c2+=1){
                                S1((T),(T));
                            }
                            for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,T);
                                c2+=1){
                                S4((T),(T));
                            }
                            for(c2=max(ti2,T + 1);c2 <= min(ti2 + ts2 + -1,
                                N + T + -2);c2+=1){
                                S3((T),(c2));
                                S4((T),(c2));
                            }
                            for(c2=max(ti2,N + T + -1);c2 <= min(ti2 + ts2 +
                                -1,N + T + -1);c2+=1){
                                S2((T),(N + T + -1));
                                S4((T),(N + T + -1));
                            }
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
```

```
#undef RMIN
```

Modified AlphaZ code for one-dimensional stencil: parallel

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ( (i)>=0 ? (i)%(j) : (j-1)-((-(i))%(j)) )

#define NMAX (131071984L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION STATIC_ARY_ARY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2) * N + i]
#define Final(i,i1) Final[MOD(i, 1)][i1]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i) Initial[i]
#define Intermediate(t,i) Intermediate[MOD(t, 2)][i]
#define Final(i,i1) Final[MOD(i, 1)][i1]

#endif

//Function bodies
void oneD(long T,long N,int T1,int T2,double* Initial,double** Final){
    // Parameter checking
    if (!((T-1>= 0 && N-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*N * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
```

```
    static double Intermediate[2][NMAX];
#endif

    #define S0(t,i) Intermediate(t,-t+i) = Initial(-t+i);
    #define S1(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
    #define S2(t,i) Intermediate(t,-t+i) = Intermediate(t-1,-t+i);
    #define S3(t,i) Intermediate(t,-t+i) = ((((2)*(Intermediate(t-1,-t+i)))+\
        (Intermediate(t-1,-t+i-1)))+(Intermediate(t-1,-t+i+1)))/(4.0);
    #define S4(i,i1) Final(T,i1-T) = Intermediate(T,i1-T);
    {
        int c1,c2,end,ss1,ss2,start,time;
        ss1=CDIV(1-T1,T1)*T1;
        ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;
        start=ss1/T1+ss2/T2;
        ss1=T;
        ss2=MAX(N-1,MAX(ss1+N-1,T+N-1))+MAX(0,T1-1);
        end=ss1/T1+ss2/T2;
        for(time=start;time <= end;time+=1){
            #pragma omp parallel for private(c1,c2,ss1,ss2)
            for(ss1=CDIV(1-T1,T1)*T1;ss1 <= T;ss1+=T1){
                ss2=(time-ss1/T1)*T2;
                if(CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2<=ss1 && ss2<=MAX(N-1,
                    MAX(ss1+N-1,T+N-1))+MAX(0,T1-1)){
                    for(c1=MAX(0,ss1);c1 <= MIN(0,ss1+T1-1);c1+=1){
                        for(c2=MAX(0,ss2);c2 <= MIN(N-1,ss2+T2-1);c2+=1){
                            S0(0,c2);
                        }
                    }
                    for(c1=MAX(1,ss1);c1 <= MIN(T-1,ss1+T1-1);c1+=1){
                        for(c2=MAX(c1,ss2);c2 <= MIN(c1,ss2+T2-1);c2+=1){
                            S1(c1,c1);
                        }
                        for(c2=MAX(c1+1,ss2);c2 <= MIN(c1+N-2,ss2+T2-1);
                            c2+=1){
                            S3(c1,c2);
                        }
                        for(c2=MAX(c1+N-1,ss2);c2 <= MIN(c1+N-1,ss2+T2-1);
                            c2+=1){
                            S2(c1,c1+N-1);
                        }
                    }
                    for(c1=MAX(T,ss1);c1 <= MIN(T,ss1+T1-1);c1+=1){
                        for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                            S1(T,T);
                        }
                        for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                            S4(T,T);
                        }
                        for(c2=MAX(T+1,ss2);c2 <= MIN(T+N-2,ss2+T2-1);c2+=1){
                            S3(T,c2);
                            S4(T,c2);
                        }
                        for(c2=MAX(T+N-1,ss2);c2 <= MIN(T+N-1,ss2+T2-1);
                            c2+=1){
                            S2(T,T+N-1);
                            S4(T,T+N-1);
                        }
                    }
                }
            }
        }
```

```
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for two-dimensional stencil: copy arrays

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))
```

```
#define NMAX (131071984L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2
//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Memory Macros
#define Initial(i) Initial[i]
#define A(t) A[t]
#define new(t) new[t]
#define Final(i) Final[i]

//Function bodies
void oneD(long T,long N,double* Initial,double* Final){
       // Parameter checking
       if (!((T-1>= 0 && N-5>= 0))) {
              printf("The value of parameters are not vaild.\n");
              exit(-1);
       }

#if ALLOCATION == MALLOC_ARRAY
       double *A = (double*) malloc(N * sizeof(double));
       double *new = (double*) malloc(N * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
       static double A[NMAX];
       static double new[NMAX];
#endif

       #define S0(t,i,i2) new(i2) = A(i2);
       #define S1(t,i,i2) new(i2) = ((((2)*(A(i2)))+(A(i2-1)))+(A(i2+1)))/\
           (4.0);
       #define S2(t,i,i2) A(i2) = Initial(i2);
       #define S3(t,i,i2) A(i2) = new(i2);
       #define S4(i,i1,i2) Final(i2) = new(i2);
       {
              int c1,c3;
              for(c3=0;c3 <= N + -1;c3+=1){
                     S2((0),(1),(c3));
              }
              for(c1=1;c1 <= T + -1;c1+=1){
                     S0((c1),(0),(0));
                     for(c3=1;c3 <= N + -2;c3+=1){
                            S1((c1),(0),(c3));
                     }
                     S0((c1),(0),(N + -1));
                     for(c3=0;c3 <= N + -1;c3+=1){
                            S3((c1),(1),(c3));
                     }
              }
              S0((T),(0),(0));
              for(c3=1;c3 <= N + -2;c3+=1){
                     S1((T),(0),(c3));
              }
              S0((T),(0),(N + -1));
              for(c3=0;c3 <= N + -1;c3+=1){
                     S4((T),(1),(c3));
              }
       }
       #undef S0
       #undef S1
       #undef S2
```

```
      #undef S3
      #undef S4
}
#undef Initial
#undef A
#undef new
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for two-dimensional stencil: untiled

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define XMAS (1250L)
#define YMAX (1250L)
#define MALLOC_ARRAY      1
```

```
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2) * (X * Y) + i * Y + j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#endif

//Function bodies
void twoD(long T,long X,long Y,double** Initial,double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*X*Y * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
    static double Intermediate[2][XMAS][YMAX];
#endif

    #define S0(t,i,j) Intermediate(t,i,j) = Initial(i,j);
    #define S1(t,i,j) Intermediate(t,i,j) = Intermediate(t-1,i,j);
    #define S2(t,i,j) Intermediate(t,i,j) = ((((((4)*(Intermediate(t-1,i,\
        j)))+(Intermediate(t-1,i,j+1)))+(Intermediate(t-1,i,j-1)))+\
        (Intermediate(t-1,i+1,j)))+(Intermediate(t-1,i-1,j)))/(8.0);
    #define S3(i,j,i2) Final(T,j,i2) = Intermediate(T,j,i2);
    {
        int c1,c2,c3;
        for(c2=0;c2 <= X + -1;c2+=1){
            for(c3=0;c3 <= Y + -1;c3+=1){
                S0((0),(c2),(c3));
            }
        }
        for(c1=1;c1 <= T + -1;c1+=1){
            for(c3=0;c3 <= Y + -1;c3+=1){
                S1((c1),(0),(c3));
            }
            for(c2=1;c2 <= X + -2;c2+=1){
                S1((c1),(c2),(0));
                for(c3=1;c3 <= Y + -2;c3+=1){
                    S2((c1),(c2),(c3));
                }
                S1((c1),(c2),(Y + -1));
            }
            for(c3=0;c3 <= Y + -1;c3+=1){
                S1((c1),(X + -1),(c3));
            }
        }
        for(c3=0;c3 <= Y + -1;c3+=1){
```

```
            S1((T),(0),(c3));
            S3((T),(0),(c3));
        }
        for(c2=1;c2 <= X + -2;c2+=1){
            S1((T),(c2),(0));
            S3((T),(c2),(0));
            for(c3=1;c3 <= Y + -2;c3+=1){
                S2((T),(c2),(c3));
                S3((T),(c2),(c3));
            }
            S1((T),(c2),(Y + -1));
            S3((T),(c2),(Y + -1));
        }
        for(c3=0;c3 <= Y + -1;c3+=1){
            S1((T),(X + -1),(c3));
            S3((T),(X + -1),(c3));
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for two-dimensional stencil: tiling

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
```

```
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define XMAS (1250L)
#define YMAX (1250L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2) * (X * Y) + i * Y + j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#endif

//Function bodies
void twoD(long T,long X,long Y,int ts1,int ts2,int ts3,double** Initial,
    double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*X*Y * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
    static double Intermediate[2][XMAS][YMAX];
#endif

    #define S0(t,i,j) Intermediate(t,-t+i,-t+j) = Initial(-t+i,-t+j);
    #define S1(t,i,j) Intermediate(t,-t+i,-t+j) = Intermediate(t-1,-t+i,
        -t+j);
    #define S2(t,i,j) Intermediate(t,-t+i,-t+j) = ((((((4)*(Intermediate(\
        t-1,-t+i,-t+j)))+(Intermediate(t-1,-t+i,-t+j+1)))+(Intermediate(t-1,\
        -t+i,-t+j-1)))+(Intermediate(t-1,-t+i+1,-t+j)))+(Intermediate(t-1,\
        -t+i-1,-t+j)))/(8.0);
    #define S3(i,j,i2) Final(T,j-T,i2-T) = Intermediate(T,j-T,i2-T);
    {
```

```
int c1,c2,c3,ti1,ti2,ti3;
for(ti1=ceild(-ts1 + 1, ts1)*ts1;ti1 <= T;ti1+=ts1){
    for(ti2=ceild(min(0,min(ti1,T)) + -ts2 + 1, ts2)*ts2;
        ti2 <= max(X + -1,max(X + ti1 + -1,T + X + -1)) +
        max(0,ts1 + -1);ti2+=ts2){
        for(ti3=ceild(min(T,min(ti1,0)) + -ts3 + 1, ts3)*ts3;
            ti3 <= max(T + Y + -1,max(Y + ti1 + -1,Y + -1)) +
                max(0,ts1 + -1);ti3+=ts3){
            /** point loops for generic case **/
            {
                for(c1=max(ti1,0);c1 <= min(ti1 + ts1 + -1,0);c1+=1){
                    for(c2=max(ti2,0);c2 <= min(ti2 + ts2 + -1,
                        X + -1);c2+=1){
                        for(c3=max(ti3,0);c3 <= min(ti3 + ts3 + -1,
                            Y + -1);c3+=1){
                            S0((0),(c2),(c3));
                        }
                    }
                }
                for(c1=max(ti1,1);c1 <= min(ti1 + ts1 + -1,
                    T + -1);c1+=1){
                    for(c2=max(ti2,c1);c2 <= min(ti2 + ts2 + -1,c1);
                        c2+=1){
                        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
                            Y + c1 + -1);c3+=1){
                            S1((c1),(c1),(c3));
                        }
                    }
                    for(c2=max(ti2,c1 + 1);c2 <= min(ti2 + ts2 + -1,
                        X + c1 + -2);c2+=1){
                        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
                            c1);c3+=1){
                            S1((c1),(c2),(c1));
                        }
                        for(c3=max(ti3,c1 + 1);c3 <= min(ti3 + ts3 +
                            -1,Y + c1 + -2);c3+=1){
                            S2((c1),(c2),(c3));
                        }
                        for(c3=max(ti3,Y + c1 + -1);c3 <= min(ti3 +
                            ts3 + -1,Y + c1 + -1);c3+=1){
                            S1((c1),(c2),(Y + c1 + -1));
                        }
                    }
                    for(c2=max(ti2,X + c1 + -1);c2 <= min(ti2 + ts2 +
                        -1,X + c1 + -1);c2+=1){
                        for(c3=max(ti3,c1);c3 <= min(ti3 + ts3 + -1,
                            Y + c1 + -1);c3+=1){
                            S1((c1),(X + c1 + -1),(c3));
                        }
                    }
                }
                for(c1=max(ti1,T);c1 <= min(ti1 + ts1 + -1,T);c1+=1){
                    for(c2=max(ti2,T);c2 <= min(ti2 + ts2 + -1,T);
                        c2+=1){
                        for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
                            T + Y + -1);c3+=1){
                            S1((T),(T),(c3));
                            S3((T),(T),(c3));
                        }
                    }
```

```
                            for(c2=max(ti2,T + 1);c2 <= min(ti2 + ts2 + -1,
                                T + X + -2);c2+=1){
                                for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
                                    T);c3+=1){
                                    S1((T),(c2),(T));
                                    S3((T),(c2),(T));
                                }
                                for(c3=max(ti3,T + 1);c3 <= min(ti3 + ts3 +
                                    -1,T + Y + -2);c3+=1){
                                    S2((T),(c2),(c3));
                                    S3((T),(c2),(c3));
                                }
                                for(c3=max(ti3,T + Y + -1);c3 <= min(ti3 +
                                    ts3 + -1,T + Y + -1);c3+=1){
                                    S1((T),(c2),(T + Y + -1));
                                    S3((T),(c2),(T + Y + -1));
                                }
                            }
                            for(c2=max(ti2,T + X + -1);c2 <= min(ti2 + ts2 +
                            -1,T + X + -1);c2+=1){
                                for(c3=max(ti3,T);c3 <= min(ti3 + ts3 + -1,
                                    T + Y + -1);c3+=1){
                                    S1((T),(T + X + -1),(c3));
                                    S3((T),(T + X + -1),(c3));
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for two-dimensional stencil: parallel

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ( (i)>=0 ? (i)%(j) : (j-1)-((-(i))%(j)) )
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define XMAS (1250L)
#define YMAX (1250L)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2) * (X * Y) + i * Y + j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#elif ALLOCATION == STATIC_ARY_ARY
#define Initial(i,j) Initial[i][j]
#define Intermediate(t,i,j) Intermediate[MOD(t, 2)][i][j]
#define Final(i,j,i2) Final[MOD(i, 1)][j][i2]

#endif

//Function bodies
void twoD(long T,long X,long Y,int T1,int T2,int T3,double** Initial,
    double*** Final){
    // Parameter checking
    if (!((T-1>= 0 && X-5>= 0 && Y-5>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
```

```
#if ALLOCATION == MALLOC_ARRAY
    double *Intermediate = (double *) malloc(2*X*Y * sizeof(double));
#elif ALLOCATION == STATIC_ARY_ARY
    static double Intermediate[2][XMAS][YMAX];
#endif

    #define S0(t,i,j) Intermediate(t,-t+i,-t+j) = Initial(-t+i,-t+j);
    #define S1(t,i,j) Intermediate(t,-t+i,-t+j) = Intermediate(t-1,-t+i,\
        -t+j);
    #define S2(t,i,j) Intermediate(t,-t+i,-t+j) = ((((((4)*(Intermediate(\
        t-1,-t+i,-t+j)))+(Intermediate(t-1,-t+i,-t+j+1)))+(Intermediate(t-1,\
        -t+i,-t+j-1)))+(Intermediate(t-1,-t+i+1,-t+j)))+(Intermediate(t-1,\
        -t+i-1,-t+j)))/(8.0);
    #define S3(i,j,i2) Final(T,j-T,i2-T) = Intermediate(T,j-T,i2-T);
    {
        int c1,c2,c3,end,ss1,ss2,ss3,start,time;
        ss1=CDIV(1-T1,T1)*T1;
        ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;
        ss3=CDIV(1+MIN(0,MIN(ss2,T))-T3,T3)*T3;
        start=ss1/T1+ss2/T2+ss3/T3;
        ss1=T;
        ss2=MAX(X-1,MAX(ss1+X-1,T+X-1))+MAX(0,T1-1);
        ss3=MAX(Y-1,MAX(ss2+Y-1,T+Y-1))+MAX(0,T1-1);
        end=ss1/T1+ss2/T2+ss3/T3;
        for(time=start;time <= end;time+=1){
            #pragma omp parallel for private(c1,c2,c3,ss1,ss2,ss3)
            for(ss1=CDIV(1-T1,T1)*T1;ss1 <= T;ss1+=T1){
                for(ss2=CDIV(1+MIN(0,MIN(ss1,T))-T2,T2)*T2;ss2 <=
                    MAX(X-1,MAX(ss1+X-1,T+X-1))+MAX(0,T1-1);ss2+=T2){
                    ss3=(time-ss1/T1-ss2/T2)*T3;
                    if(CDIV(1+MIN(0,MIN(ss2,T))-T3,T3)*T3<=ss2 &&
                        ss3<=MAX(Y-1,MAX(ss2+Y-1,T+Y-1))+MAX(0,T1-1)){
                        for(c1=MAX(0,ss1);c1 <= MIN(0,ss1+T1-1);c1+=1){
                            for(c2=MAX(0,ss2);c2 <= MIN(X-1,ss2+T2-1);c2+=1){
                                for(c3=MAX(0,ss3);c3 <= MIN(Y-1,ss3+T3-1);
                                    c3+=1){
                                    S0(0,c2,c3);
                                }
                            }
                        }
                        for(c1=MAX(1,ss1);c1 <= MIN(T-1,ss1+T1-1);c1+=1){
                            for(c2=MAX(c1,ss2);c2 <= MIN(c1,ss2+T2-1);c2+=1){
                                for(c3=MAX(c1,ss3);c3 <= MIN(c1+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(c1,c1,c3);
                                }
                            }
                            for(c2=MAX(c1+1,ss2);c2 <= MIN(c1+X-2,ss2+T2-1);
                                c2+=1){
                                for(c3=MAX(c1,ss3);c3 <= MIN(c1,ss3+T3-1);
                                    c3+=1){
                                    S1(c1,c2,c1);
                                }
                                for(c3=MAX(c1+1,ss3);c3 <= MIN(c1+Y-2,
                                    ss3+T3-1);c3+=1){
                                    S2(c1,c2,c3);
                                }
                                for(c3=MAX(c1+Y-1,ss3);c3 <= MIN(c1+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(c1,c2,c1+Y-1);
                                }
```

```
                            }
                            for(c2=MAX(c1+X-1,ss2);c2 <= MIN(c1+X-1,
                                ss2+T2-1);c2+=1){
                                for(c3=MAX(c1,ss3);c3 <= MIN(c1+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(c1,c1+X-1,c3);
                                }
                            }
                        }
                        for(c1=MAX(T,ss1);c1 <= MIN(T,ss1+T1-1);c1+=1){
                            for(c2=MAX(T,ss2);c2 <= MIN(T,ss2+T2-1);c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T+Y-1,ss3+T3-1);
                                    c3+=1){
                                    S1(T,T,c3);
                                    S3(T,T,c3);
                                }
                            }
                            for(c2=MAX(T+1,ss2);c2 <= MIN(T+X-2,ss2+T2-1);
                                c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T,ss3+T3-1);
                                    c3+=1){
                                    S1(T,c2,T);
                                    S3(T,c2,T);
                                }
                                for(c3=MAX(T+1,ss3);c3 <= MIN(T+Y-2,
                                    ss3+T3-1);c3+=1){
                                    S2(T,c2,c3);
                                    S3(T,c2,c3);
                                }
                                for(c3=MAX(T+Y-1,ss3);c3 <= MIN(T+Y-1,
                                    ss3+T3-1);c3+=1){
                                    S1(T,c2,T+Y-1);
                                    S3(T,c2,T+Y-1);
                                }
                            }
                            for(c2=MAX(T+X-1,ss2);c2 <= MIN(T+X-1,ss2+T2-1);
                                c2+=1){
                                for(c3=MAX(T,ss3);c3 <= MIN(T+Y-1,ss3+T3-1);
                                    c3+=1){
                                    S1(T,T+X-1,c3);
                                    S3(T,T+X-1,c3);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
}
#undef Initial
#undef Intermediate
#undef Final
//Postamble
#undef MAX
#undef max
#undef MIN
```

```
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Modified AlphaZ code for Tomcatv: obvious

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include "external_functions.h"
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define NMAX (2048 + 17)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Local Function Declarations
#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RXM_reduce_1(long,long,int,double**,double**,double,double,
    double,double,double,double,double,double*,double*,double,double,double,
    double,double,double,double*,double*,double,double*,double*,double*,
    double*,double*,double*,double*,double***,double***);
```

```
double Tomcatv_RYM_reduce_1(long,long,int,double**,double**,double,double,
    double,double,double,double,double,double*,double*,double,double,double,
    double,double,double,double*,double*,double,double*,double*,double*,
    double*,double*,double*,double*,double***,double***);

#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A,double B,
    double C,double (*AA)[NMAX],double (*DD)[NMAX],double PXX,double QXX,
    double PYY,double QYY,double PXY,double QXY,double (*RX0)[NMAX],
    double (*RY0)[NMAX],double R,double (*D)[NMAX],double (*RX1)[NMAX],
    double (*RY1)[NMAX],double (*RX2)[NMAX],double (*RY2)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM);
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A,double B,
    double C,double (*AA)[NMAX],double (*DD)[NMAX],double PXX,double QXX,
    double PYY,double QYY,double PXY,double QXY,double (*RX0)[NMAX],
    double (*RY0)[NMAX],double R,double (*D)[NMAX],double (*RX1)[NMAX],
    double (*RY1)[NMAX],double (*RX2)[NMAX],double (*RY2)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM);

#endif

//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define XX() XX
#define YX() YX
#define XY() XY
#define YY() YY
#define A() A
#define B() B
#define C() C
#define AA(t,i,j) AA[i * N + j - N - 1]
#define DD(t,i,j) DD[i * N + j - N - 1]
#define PXX() PXX
#define QXX() QXX
#define PYY() PYY
#define QYY() QYY
#define PXY() PXY
#define QXY() QXY
#define RX0(t,i,j) RX0[i * N + j - N - 1]
#define RY0(t,i,j) RY0[i * N + j - N - 1]
#define R() R
#define D(t,i,j) D[i * N + j - N - 1]
#define RX1(t,i,j) RX1[i * N + j - N - 1]
#define RY1(t,i,j) RY1[i * N + j - N - 1]
#define RX2(t,i,j) RX2[i * N + j - N - 1]
#define RY2(t,i,j) RY2[i * N + j - N - 1]
#define X(t,i,j) X[i * N + j - N - 1]
#define Y(t,i,j) Y[i * N + j - N - 1]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]

#elif ALLOCATION == STATIC_ARY_ARY
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define XX() XX
#define YX() YX
#define XY() XY
```

```
#define YY() YY
#define A() A
#define B() B
#define C() C
#define AA(t,i,j) AA[i][j]
#define DD(t,i,j) DD[i][j]
#define PXX() PXX
#define QXX() QXX
#define PYY() PYY
#define QYY() QYY
#define PXY() PXY
#define QXY() QXY
#define RX0(t,i,j) RX0[i][j]
#define RY0(t,i,j) RY0[i][j]
#define R() R
#define D(t,i,j) D[i][j]
#define RX1(t,i,j) RX1[i][j]
#define RY1(t,i,j) RY1[i][j]
#define RX2(t,i,j) RX2[i][j]
#define RY2(t,i,j) RY2[i][j]
#define X(t,i,j) X[i][j]
#define Y(t,i,j) Y[i][j]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]

#endif

#if ALLOCATION == MALLOC_ARRAY
    static double XX, YX, XY, YY;
    static double C, R, A, B;
    static double PXX, QXX, PYY, QYY, PXY, QXY;
    double *AA;
    double *DD;
    double *RX0;
    double *RY0;
    double *D;
    double *RX1;
    double *RY1;
    double *RX2;
    double *RY2;
    double *X;
    double *Y;

#elif ALLOCATION == STATIC_ARY_ARY
    static double XX, YX, XY, YY;
    static double A, B, C, AA[NMAX][NMAX], DD[NMAX][NMAX];
    static double PXX, QXX, PYY, QYY, PXY, QXY;
    static double RX0[NMAX][NMAX], RY0[NMAX][NMAX], R, D[NMAX][NMAX];
    static double RX1[NMAX][NMAX], RY1[NMAX][NMAX], RX2[NMAX][NMAX],
RY2[NMAX][NMAX];
    static double X[NMAX][NMAX], Y[NMAX][NMAX];
#endif

//Function bodies
void Tomcatv(long ITACT,long N,double** X0,double** Y0,double*** RXM,
    double*** RYM){
    // Parameter checking
    if (!((ITACT-4>= 0 && N-4>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }
```

```
#if ALLOCATION == MALLOC_ARRAY
    AA  = (double *) malloc(N*N * sizeof(double));
    DD  = (double *) malloc(N*N * sizeof(double));
    RX0 = (double *) malloc(N*N * sizeof(double));
    RY0 = (double *) malloc(N*N * sizeof(double));
    D   = (double *) malloc(N*N * sizeof(double));
    RX1 = (double *) malloc(N*N * sizeof(double));
    RY1 = (double *) malloc(N*N * sizeof(double));
    RX2 = (double *) malloc(N*N * sizeof(double));
    RY2 = (double *) malloc(N*N * sizeof(double));
    X   = (double *) malloc(N*N * sizeof(double));
    Y   = (double *) malloc(N*N * sizeof(double));
#endif

    #define S0(t,i,j,i3,i4,i5,i6) XX() = (X(i-1,i3+1,i5))-(X(i-1,i3-1,i5));
    #define S1(t,i,j,i3,i4,i5,i6) YX() = (Y(i-1,i3+1,i5))-(Y(i-1,i3-1,i5));
    #define S2(t,i,j,i3,i4,i5,i6) XY() = (X(i-1,i3,i5+1))-(X(i-1,i3,i5-1));
    #define S3(t,i,j,i3,i4,i5,i6) YY() = (Y(i-1,i3,i5+1))-(Y(i-1,i3,i5-1));
    #define S4(t,i,j,i3,i4,i5,i6) A() = (0.25)*(((XY())*(XY()))+\
        ((YY())*(YY())));
    #define S5(t,i,j,i3,i4,i5,i6) B() = (0.25)*(((XX())*(XX()))+\
        ((YX())*(YX())));
    #define S6(t,i,j,i3,i4,i5,i6) C() = (0.125)*(((XX())*(XY()))+\
        ((YX())*(YY())));
    #define S7(t,i,j,i3,i4,i5,i6) AA(i,i3,i5) = (-(1))*(B());
    #define S8(t,i,j,i3,i4,i5,i6) DD(i,i3,i5) = ((B())+(B()))+\
        ((A())*((2.0)/(0.98)));
    #define S9(t,i,j,i3,i4,i5,i6) PXX() = ((X(i-1,i3+1,i5))-\
        ((2.0)*(X(i-1,i3,i5))))+(X(i-1,i3-1,i5));
    #define S10(t,i,j,i3,i4,i5,i6) QXX() = ((Y(i-1,i3+1,i5))-\
        ((2.0)*(Y(i-1,i3,i5))))+(Y(i-1,i3-1,i5));
    #define S11(t,i,j,i3,i4,i5,i6) PYY() = ((X(i-1,i3,i5+1))-\
        ((2.0)*(X(i-1,i3,i5))))+(X(i-1,i3,i5-1));
    #define S12(t,i,j,i3,i4,i5,i6) QYY() = ((Y(i-1,i3,i5+1))-\
        ((2.0)*(Y(i-1,i3,i5))))+(Y(i-1,i3,i5-1));
    #define S13(t,i,j,i3,i4,i5,i6) PXY() = (((X(i-1,i3+1,i5+1))-\
        (X(i-1,i3+1,i5-1)))-(X(i-1,i3-1,i5+1)))+(X(i-1,i3-1,i5-1));
    #define S14(t,i,j,i3,i4,i5,i6) QXY() = (((Y(i-1,i3+1,i5+1))-\
        (Y(i-1,i3+1,i5-1)))-(Y(i-1,i3-1,i5+1)))+(Y(i-1,i3-1,i5-1));
    #define S15(t,i,j,i3,i4,i5,i6) RX0(i,i3,i5) = (((A())*(PXX()))+\
        ((B())*(PYY())))-((C())*(PXY()));
    #define S16(t,i,j,i3,i4,i5,i6) RY0(i,i3,i5) = (((A())*(QXX()))+
        (B())*(QYY())))-((C())*(QXY()));
    #define S17(t,i,j,i3,i4,i5,i6) R() = (AA(i,i3,i5))*(D(i,i3,i5-1));
    #define S18(t,i,j,i3,i4,i5,i6) D(i,i3,i5) = (1)/(DD(i,i3,i5));
    #define S19(t,i,j,i3,i4,i5,i6) D(i,i3,i5) = (1)/((DD(i,i3,i5))-\
        ((AA(i,i3,i5-1))*(R())));
    #define S20(t,i,j,i3,i4,i5,i6) RX1(i,i3,i5) = RX0(i,i3,i5);
    #define S21(t,i,j,i3,i4,i5,i6) RX1(i,i3,i5) = (RX0(i,i3,i5))-\
        ((RX0(i,i3,i5-1))*(R()));
    #define S22(t,i,j,i3,i4,i5,i6) RY1(i,i3,i5) = RY0(i,i3,i5);
    #define S23(t,i,j,i3,i4,i5,i6) RY1(i,i3,i5) = (RY0(i,i3,i5))-\
        ((RY0(i,i3,i5-1))*(R()));
    #define S24(t,i,j,i3,i4,i5,i6) RX2(i,i3,-i5+N) = (RX1(i,i3,-i5+N))*\
        (D(i,i3,-i5+N));
    #define S25(t,i,j,i3,i4,i5,i6) RX2(i,i3,-i5+N) = (RX1(i,i3,-i5+N))-\
        (((AA(i,i3,-i5+N))*(RX1(i,i3,-i5+N+1)))*(D(i,i3,-i5+N)));
    #define S26(t,i,j,i3,i4,i5,i6) RY2(i,i3,-i5+N) = (RY1(i,i3,-i5+N))*\
        (D(i,i3,-i5+N));
    #define S27(t,i,j,i3,i4,i5,i6) RY2(i,i3,-i5+N) = (RY1(i,i3,-i5+N))-\
        (((AA(i,i3,-i5+N))*(RY1(i,i3,-i5+N+1)))*(D(i,i3,-i5+N)));
```

```
#define S28(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = X0(i3,i5);
#define S29(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = X(i-1,i3,i5);
#define S30(t,i,j,i3,i4,i5,i6) X(i,i3,i5) = (X(i-1,i3,i5))+\
    (RX2(i,i3,i5));
#define S31(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = Y0(i3,i5);
#define S32(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = Y(i-1,i3,i5);
#define S33(t,i,j,i3,i4,i5,i6) Y(i,i3,i5) = (Y(i-1,i3,i5))+\
    (RY2(i,i3,i5));
#define S34(t,i1,i2,i3,i4,i5,i6) RXM(i1,0,0) = Tomcatv_RXM_reduce_1(\
    ITACT,N,i1,X0,Y0,XX,YX,XY,YY,A,B,C,AA,DD,PXX,QXX,PYY,QYY,PXY,QXY,\
    RX0,RY0,R,D,RX1,RY1,RX2,RY2,X,Y,RXM,RYM);
#define S35(t,i1,i2,i3,i4,i5,i6) RYM(i1,0,0) = Tomcatv_RYM_reduce_1(
    ITACT,N,i1,X0,Y0,XX,YX,XY,YY,A,B,C,AA,DD,PXX,QXX,PYY,QYY,PXY,QXY,\
    RX0,RY0,R,D,RX1,RY1,RX2,RY2,X,Y,RXM,RYM);
{
    int c2,c4,c6;
    if(N >= 5){
        for(c4=2;c4 <= N + -1;c4+=1){
            for(c6=2;c6 <= N + -1;c6+=1){
                S0((0),(1),(0),(c4),(0),(c6),(0));
                S1((0),(1),(0),(c4),(0),(c6),(1));
                S2((0),(1),(0),(c4),(0),(c6),(2));
                S3((0),(1),(0),(c4),(0),(c6),(3));
                S4((0),(1),(0),(c4),(0),(c6),(4));
                S5((0),(1),(0),(c4),(0),(c6),(5));
                S6((0),(1),(0),(c4),(0),(c6),(6));
                S7((0),(1),(0),(c4),(0),(c6),(7));
                S8((0),(1),(0),(c4),(0),(c6),(8));
                S9((0),(1),(0),(c4),(0),(c6),(9));
                S10((0),(1),(0),(c4),(0),(c6),(10));
                S11((0),(1),(0),(c4),(0),(c6),(11));
                S12((0),(1),(0),(c4),(0),(c6),(12));
                S13((0),(1),(0),(c4),(0),(c6),(13));
                S14((0),(1),(0),(c4),(0),(c6),(14));
                S15((0),(1),(0),(c4),(0),(c6),(15));
                S16((0),(1),(0),(c4),(0),(c6),(16));
            }
        }
        S34((0),(1),(1),(N),(0),(N),(0));
        S35((0),(1),(1),(N),(0),(N),(1));
        for(c4=2;c4 <= N + -1;c4+=1){
            S18((0),(1),(2),(c4),(0),(2),(1));
            S20((0),(1),(2),(c4),(0),(2),(2));
            S22((0),(1),(2),(c4),(0),(2),(3));
            for(c6=3;c6 <= N + -1;c6+=1){
                S17((0),(1),(2),(c4),(0),(c6),(0));
                S19((0),(1),(2),(c4),(0),(c6),(1));
                S21((0),(1),(2),(c4),(0),(c6),(2));
                S23((0),(1),(2),(c4),(0),(c6),(3));
            }
        }
        for(c4=2;c4 <= N + -1;c4+=1){
            S24((0),(1),(3),(c4),(0),(1),(0));
            S26((0),(1),(3),(c4),(0),(1),(1));
            for(c6=3;c6 <= N + -2;c6+=1){
                S25((0),(1),(3),(c4),(0),(c6),(0));
                S27((0),(1),(3),(c4),(0),(c6),(1));
            }
        }
        for(c6=1;c6 <= N + -1;c6+=1){
            S28((0),(1),(4),(1),(0),(c6),(0));
```

```
            S29((0),(1),(4),(1),(0),(c6),(0));
            S31((0),(1),(4),(1),(0),(c6),(1));
            S32((0),(1),(4),(1),(0),(c6),(1));
        }
        S28((0),(1),(4),(1),(0),(N),(0));
        S29((0),(1),(4),(1),(0),(N),(0));
        S31((0),(1),(4),(1),(0),(N),(1));
        S32((0),(1),(4),(1),(0),(N),(1));
        for(c4=2;c4 <= N + -1;c4+=1){
            S28((0),(1),(4),(c4),(0),(1),(0));
            S29((0),(1),(4),(c4),(0),(1),(0));
            S31((0),(1),(4),(c4),(0),(1),(1));
            S32((0),(1),(4),(c4),(0),(1),(1));
            for(c6=2;c6 <= N + -1;c6+=1){
                S28((0),(1),(4),(c4),(0),(c6),(0));
                S31((0),(1),(4),(c4),(0),(c6),(1));
            }
            S28((0),(1),(4),(c4),(0),(N),(0));
            S29((0),(1),(4),(c4),(0),(N),(0));
            S31((0),(1),(4),(c4),(0),(N),(1));
            S32((0),(1),(4),(c4),(0),(N),(1));
        }
        for(c6=1;c6 <= N;c6+=1){
            S28((0),(1),(4),(N),(0),(c6),(0));
            S29((0),(1),(4),(N),(0),(c6),(0));
            S31((0),(1),(4),(N),(0),(c6),(1));
            S32((0),(1),(4),(N),(0),(c6),(1));
        }
    }
    if(N >= 5){
        for(c2=2;c2 <= ITACT;c2+=1){
            for(c4=2;c4 <= N + -1;c4+=1){
                for(c6=2;c6 <= N + -1;c6+=1){
                    S0((0),(c2),(0),(c4),(0),(c6),(0));
                    S1((0),(c2),(0),(c4),(0),(c6),(1));
                    S2((0),(c2),(0),(c4),(0),(c6),(2));
                    S3((0),(c2),(0),(c4),(0),(c6),(3));
                    S4((0),(c2),(0),(c4),(0),(c6),(4));
                    S5((0),(c2),(0),(c4),(0),(c6),(5));
                    S6((0),(c2),(0),(c4),(0),(c6),(6));
                    S7((0),(c2),(0),(c4),(0),(c6),(7));
                    S8((0),(c2),(0),(c4),(0),(c6),(8));
                    S9((0),(c2),(0),(c4),(0),(c6),(9));
                    S10((0),(c2),(0),(c4),(0),(c6),(10));
                    S11((0),(c2),(0),(c4),(0),(c6),(11));
                    S12((0),(c2),(0),(c4),(0),(c6),(12));
                    S13((0),(c2),(0),(c4),(0),(c6),(13));
                    S14((0),(c2),(0),(c4),(0),(c6),(14));
                    S15((0),(c2),(0),(c4),(0),(c6),(15));
                    S16((0),(c2),(0),(c4),(0),(c6),(16));
                }
            }
            S34((0),(c2),(1),(N),(0),(N),(0));
            S35((0),(c2),(1),(N),(0),(N),(1));
            for(c4=2;c4 <= N + -1;c4+=1){
                S18((0),(c2),(2),(c4),(0),(2),(1));
                S20((0),(c2),(2),(c4),(0),(2),(2));
                S22((0),(c2),(2),(c4),(0),(2),(3));
                for(c6=3;c6 <= N + -1;c6+=1){
                    S17((0),(c2),(2),(c4),(0),(c6),(0));
                    S19((0),(c2),(2),(c4),(0),(c6),(1));
```

```
                    S21((0),(c2),(2),(c4),(0),(c6),(2));
                    S23((0),(c2),(2),(c4),(0),(c6),(3));
                }
            }
            for(c4=2;c4 <= N + -1;c4+=1){
                S24((0),(c2),(3),(c4),(0),(1),(0));
                S26((0),(c2),(3),(c4),(0),(1),(1));
                for(c6=3;c6 <= N + -2;c6+=1){
                    S25((0),(c2),(3),(c4),(0),(c6),(0));
                    S27((0),(c2),(3),(c4),(0),(c6),(1));
                }
            }
            for(c6=1;c6 <= N;c6+=1){
                S29((0),(c2),(4),(1),(0),(c6),(0));
                S32((0),(c2),(4),(1),(0),(c6),(1));
            }
            for(c4=2;c4 <= N + -1;c4+=1){
                S29((0),(c2),(4),(c4),(0),(1),(0));
                S32((0),(c2),(4),(c4),(0),(1),(1));
                for(c6=2;c6 <= N + -1;c6+=1){
                    S30((0),(c2),(4),(c4),(0),(c6),(0));
                    S33((0),(c2),(4),(c4),(0),(c6),(1));
                }
                S29((0),(c2),(4),(c4),(0),(N),(0));
                S32((0),(c2),(4),(c4),(0),(N),(1));
            }
            for(c6=1;c6 <= N;c6+=1){
                S29((0),(c2),(4),(N),(0),(c6),(0));
                S32((0),(c2),(4),(N),(0),(c6),(1));
            }
        }
    }
    if(N == 4){
        for(c4=2;c4 <= 3;c4+=1){
            for(c6=2;c6 <= 3;c6+=1){
                S0((0),(1),(0),(c4),(0),(c6),(0));
                S1((0),(1),(0),(c4),(0),(c6),(1));
                S2((0),(1),(0),(c4),(0),(c6),(2));
                S3((0),(1),(0),(c4),(0),(c6),(3));
                S4((0),(1),(0),(c4),(0),(c6),(4));
                S5((0),(1),(0),(c4),(0),(c6),(5));
                S6((0),(1),(0),(c4),(0),(c6),(6));
                S7((0),(1),(0),(c4),(0),(c6),(7));
                S8((0),(1),(0),(c4),(0),(c6),(8));
                S9((0),(1),(0),(c4),(0),(c6),(9));
                S10((0),(1),(0),(c4),(0),(c6),(10));
                S11((0),(1),(0),(c4),(0),(c6),(11));
                S12((0),(1),(0),(c4),(0),(c6),(12));
                S13((0),(1),(0),(c4),(0),(c6),(13));
                S14((0),(1),(0),(c4),(0),(c6),(14));
                S15((0),(1),(0),(c4),(0),(c6),(15));
                S16((0),(1),(0),(c4),(0),(c6),(16));
            }
        }
        S34((0),(1),(1),(4),(0),(4),(0));
        S35((0),(1),(1),(4),(0),(4),(1));
        for(c4=2;c4 <= 3;c4+=1){
            S18((0),(1),(2),(c4),(0),(2),(1));
            S20((0),(1),(2),(c4),(0),(2),(2));
            S22((0),(1),(2),(c4),(0),(2),(3));
            S17((0),(1),(2),(c4),(0),(3),(0));
```

```
            S19((0),(1),(2),(c4),(0),(3),(1));
            S21((0),(1),(2),(c4),(0),(3),(2));
            S23((0),(1),(2),(c4),(0),(3),(3));
        }
        for(c4=2;c4 <= 3;c4+=1){
            S24((0),(1),(3),(c4),(0),(1),(0));
            S26((0),(1),(3),(c4),(0),(1),(1));
        }
        for(c6=1;c6 <= 4;c6+=1){
            S28((0),(1),(4),(1),(0),(c6),(0));
            S29((0),(1),(4),(1),(0),(c6),(0));
            S31((0),(1),(4),(1),(0),(c6),(1));
            S32((0),(1),(4),(1),(0),(c6),(1));
        }
        for(c4=2;c4 <= 3;c4+=1){
            S28((0),(1),(4),(c4),(0),(1),(0));
            S29((0),(1),(4),(c4),(0),(1),(0));
            S31((0),(1),(4),(c4),(0),(1),(1));
            S32((0),(1),(4),(c4),(0),(1),(1));
            for(c6=2;c6 <= 3;c6+=1){
                S28((0),(1),(4),(c4),(0),(c6),(0));
                S31((0),(1),(4),(c4),(0),(c6),(1));
            }
            S28((0),(1),(4),(c4),(0),(4),(0));
            S29((0),(1),(4),(c4),(0),(4),(0));
            S31((0),(1),(4),(c4),(0),(4),(1));
            S32((0),(1),(4),(c4),(0),(4),(1));
        }
        S28((0),(1),(4),(4),(0),(1),(0));
        S29((0),(1),(4),(4),(0),(1),(0));
        S31((0),(1),(4),(4),(0),(1),(1));
        S32((0),(1),(4),(4),(0),(1),(1));
        for(c6=2;c6 <= 4;c6+=1){
            S28((0),(1),(4),(4),(0),(c6),(0));
            S29((0),(1),(4),(4),(0),(c6),(0));
            S31((0),(1),(4),(4),(0),(c6),(1));
            S32((0),(1),(4),(4),(0),(c6),(1));
        }
    }
    if(N == 4){
        for(c2=2;c2 <= ITACT;c2+=1){
            for(c4=2;c4 <= 3;c4+=1){
                for(c6=2;c6 <= 3;c6+=1){
                    S0((0),(c2),(0),(c4),(0),(c6),(0));
                    S1((0),(c2),(0),(c4),(0),(c6),(1));
                    S2((0),(c2),(0),(c4),(0),(c6),(2));
                    S3((0),(c2),(0),(c4),(0),(c6),(3));
                    S4((0),(c2),(0),(c4),(0),(c6),(4));
                    S5((0),(c2),(0),(c4),(0),(c6),(5));
                    S6((0),(c2),(0),(c4),(0),(c6),(6));
                    S7((0),(c2),(0),(c4),(0),(c6),(7));
                    S8((0),(c2),(0),(c4),(0),(c6),(8));
                    S9((0),(c2),(0),(c4),(0),(c6),(9));
                    S10((0),(c2),(0),(c4),(0),(c6),(10));
                    S11((0),(c2),(0),(c4),(0),(c6),(11));
                    S12((0),(c2),(0),(c4),(0),(c6),(12));
                    S13((0),(c2),(0),(c4),(0),(c6),(13));
                    S14((0),(c2),(0),(c4),(0),(c6),(14));
                    S15((0),(c2),(0),(c4),(0),(c6),(15));
                    S16((0),(c2),(0),(c4),(0),(c6),(16));
                }
```

```
                }
                S34((0),(c2),(1),(4),(0),(4),(0));
                S35((0),(c2),(1),(4),(0),(4),(1));
                for(c4=2;c4 <= 3;c4+=1){
                    S18((0),(c2),(2),(c4),(0),(2),(1));
                    S20((0),(c2),(2),(c4),(0),(2),(2));
                    S22((0),(c2),(2),(c4),(0),(2),(3));
                    S17((0),(c2),(2),(c4),(0),(3),(0));
                    S19((0),(c2),(2),(c4),(0),(3),(1));
                    S21((0),(c2),(2),(c4),(0),(3),(2));
                    S23((0),(c2),(2),(c4),(0),(3),(3));
                }
                for(c4=2;c4 <= 3;c4+=1){
                    S24((0),(c2),(3),(c4),(0),(1),(0));
                    S26((0),(c2),(3),(c4),(0),(1),(1));
                }
                for(c6=1;c6 <= 4;c6+=1){
                    S29((0),(c2),(4),(1),(0),(c6),(0));
                    S32((0),(c2),(4),(1),(0),(c6),(1));
                }
                for(c4=2;c4 <= 3;c4+=1){
                    S29((0),(c2),(4),(c4),(0),(1),(0));
                    S32((0),(c2),(4),(c4),(0),(1),(1));
                    for(c6=2;c6 <= 3;c6+=1){
                        S30((0),(c2),(4),(c4),(0),(c6),(0));
                        S33((0),(c2),(4),(c4),(0),(c6),(1));
                    }
                    S29((0),(c2),(4),(c4),(0),(4),(0));
                    S32((0),(c2),(4),(c4),(0),(4),(1));
                }
                for(c6=1;c6 <= 4;c6+=1){
                    S29((0),(c2),(4),(4),(0),(c6),(0));
                    S32((0),(c2),(4),(4),(0),(c6),(1));
                }
            }
        }
    }
}
#undef S0
#undef S1
#undef S2
#undef S3
#undef S4
#undef S5
#undef S6
#undef S7
#undef S8
#undef S9
#undef S10
#undef S11
#undef S12
#undef S13
#undef S14
#undef S15
#undef S16
#undef S17
#undef S18
#undef S19
#undef S20
#undef S21
#undef S22
#undef S23
```

```
    #undef S24
    #undef S25
    #undef S26
    #undef S27
    #undef S28
    #undef S29
    #undef S30
    #undef S31
    #undef S32
    #undef S33
    #undef S34
    #undef S35
}

#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A,double B,
    double C,double* AA,double* DD,double PXX,double QXX,double PYY,
    double QYY,double PXY,double QXY,double* RX0,double* RY0,double R,
    double* D,double* RX1,double* RY1,double* RX2,double* RY2,double* X,
    double* Y,double*** RXM,double*** RYM)
#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A,double B,
    double C,double (*AA)[NMAX],double (*DD)[NMAX],double PXX,double QXX,
    double PYY,double QYY,double PXY,double QXY,double (*RX0)[NMAX],
    double (*RY0)[NMAX],double R,double (*D)[NMAX],double (*RX1)[NMAX],
    double (*RY1)[NMAX],double (*RX2)[NMAX],double (*RY2)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM)
#endif
{
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RX0(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        int c2,c3;
        for(c2=2;c2 <= N + -1;c2+=1){
            for(c3=2;c3 <= N + -1;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}

#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A, double B,
    double C,double* AA,double* DD,double PXX,double QXX,double PYY,
    double QYY,double PXY,double QXY,double* RX0,double* RY0,double R,
    double* D,double* RX1,double* RY1,double* RX2,double* RY2,double* X,
    double* Y,double*** RXM,double*** RYM)
#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double XX,double YX,double XY,double YY,double A,double B,
    double C,double (*AA)[NMAX],double (*DD)[NMAX],double PXX,double QXX,
    double PYY,double QYY,double PXY,double QXY,double (*RX0)[NMAX],
    double (*RY0)[NMAX],double R,double (*D)[NMAX],double (*RX1)[NMAX],
    double (*RY1)[NMAX],double (*RX2)[NMAX],double (*RY2)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM)
```

```
#endif
{
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RY0(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        int c2,c3;
        for(c2=2;c2 <= N + -1;c2+=1){
            for(c3=2;c3 <= N + -1;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}
#undef X0
#undef Y0
#undef XX
#undef YX
#undef XY
#undef YY
#undef A
#undef B
#undef C
#undef AA
#undef DD
#undef PXX
#undef QXX
#undef PYY
#undef QYY
#undef PXY
#undef QXY
#undef RX0
#undef RY0
#undef R
#undef D
#undef RX1
#undef RY1
#undef RX2
#undef RY2
#undef X
#undef Y
#undef RXM
#undef RYM
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
```

```
#undef RMIN
```

Modified AlphaZ for Tomcatv: fewer arrays

```
//Preamble
// To compile this code, use -lm option for math library.
// Includes
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include "external_functions.h"
#include<omp.h>
// Common Macros
#define MAX(x,y)  ((x)>(y) ? (x) : (y))
#define max(x,y)  ((x)>(y) ? (x) : (y))
#define MIN(x,y)  ((x)>(y) ? (y) : (x))
#define min(x,y)  ((x)>(y) ? (y) : (x))
#define CEILD(n,d) (int)ceil(((double)(n))/((double)(d)))
#define ceild(n,d) (int)ceil(((double)(n))/((double)(d)))
#define FLOORD(n,d) (int)floor(((double)(n))/((double)(d)))
#define floord(n,d) (int)floor(((double)(n))/((double)(d)))
#define CDIV(x,y)  CEILD((x),(y))
#define div(x,y)  CDIV((x),(y))
#define FDIV(x,y)  FLOORD((x),(y))
#define LB_SHIFT(b,s) ((int)ceild(b,s) * s)
#define MOD(i,j)  ((i)%(j))
// Reduction Operators
#define RADD(x,y)  ((x)+=(y))
#define RMUL(x,y)  ((x)*=(y))
#define RMAX(x,y)  ((x)=MAX((x),(y)))
#define RMIN(x,y)  ((x)=MIN((x),(y)))

#define NMAX (2048 + 17)
#define MALLOC_ARRAY       1
#define STATIC_ARY_ARY     2

//Here we choose our allocation.
#define ALLOCATION MALLOC_ARRAY

//Local Function Declarations
#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RXM_reduce_1(long,long,int,double**,double**,double*,double*,
    double*,double*,double*,double***,double***);
double Tomcatv_RYM_reduce_1(long,long,int,double**,double**,double*,double*,
    double*,double*,double*,double***,double***);

#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double (*D)[NMAX],double (*RX)[NMAX],double (*RY)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM);
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double (*D)[NMAX],double (*RX)[NMAX],double (*RY)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM);

#endif
```

```
//Memory Macros
#if ALLOCATION == MALLOC_ARRAY
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define D(t,i,j) D[j * N + i - N - 1]
#define RX(t,i,j) RX[j * N + i - N - 1]
#define RY(t,i,j) RY[j * N + i - N - 1]
#define X(t,i,j) X[j * N + i - N - 1]
#define Y(t,i,j) Y[j * N + i - N - 1]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]

#elif ALLOCATION == STATIC_ARY_ARY
#define X0(i,j) X0[i][j]
#define Y0(i,j) Y0[i][j]
#define D(t,i,j) D[i][j]
#define RX(t,i,j) RX[i][j]
#define RY(t,i,j) RY[i][j]
#define X(t,i,j) X[i][j]
#define Y(t,i,j) Y[i][j]
#define RXM(t,i1,i2) RXM[t][i1][i2]
#define RYM(t,i1,i2) RYM[t][i1][i2]

#endif

#if ALLOCATION == MALLOC_ARRAY
    double *D;
    double *RX;
    double *RY;
    double *X;
    double *Y;
#elif ALLOCATION == STATIC_ARY_ARY
    static double RX[NMAX][NMAX], RY[NMAX][NMAX], D[NMAX][NMAX];
    static double X[NMAX][NMAX], Y[NMAX][NMAX];
#endif

//Function bodies
void Tomcatv(long ITACT,long N,double** X0,double** Y0,double*** RXM,
    double*** RYM){
    // Parameter checking
    if (!((ITACT-4>= 0 && N-4>= 0))) {
        printf("The value of parameters are not vaild.\n");
        exit(-1);
    }

#if ALLOCATION == MALLOC_ARRAY
    D = (double *) malloc(N*N * sizeof(double));
    RX = (double *) malloc(N*N * sizeof(double));
    RY = (double *) malloc(N*N * sizeof(double));
    X = (double *) malloc(N*N * sizeof(double));
    Y = (double *) malloc(N*N * sizeof(double));
#endif

    #define S0(t,i,j) D(t,i,j) = (((j-1== 0))?(1)/((((0.5)*(pow((X(t-1,i+1,\
        j))-(X(t-1,i-1,j))),2)))+((2)*(pow((Y(t-1,i+1,j))-(Y(t-1,i-1,j))),\
        2))))+(((0.25)*(pow((X(t-1,i,j+1))-(X(t-1,i,j-1)),2)))+(((2.0)/\
        (0.98))*(pow((Y(t-1,i,j+1))-(Y(t-1,i,j-1)),2))))):((1)/((((2)*\
        (((0.25)*(pow((X(t-1,i+1,j))-(X(t-1,i-1,j))),2)))+(pow((Y(t-1,i+1,\
        j))-(Y(t-1,i-1,j))),2))))+((((0.25)*(pow((X(t-1,i,j+1))-(X(t-1,i,\
        j-1)),2)))+(pow((Y(t-1,i,j+1))-(Y(t-1,i,j-1)),2)))*((2.0)/(0.98))))-\
        (((((0.25)*(pow((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1)),2)))+(pow((Y(t-1,\
```

```
    i+1,j-1))-(Y(t-1,i-1,j-1)),2)))*(((0.25)*(pow((X(t-1,i+1,j))-(X(t-1,\
    i-1,j)),2)))+(pow((Y(t-1,i+1,j))-(Y(t-1,i-1,j)),2))))*(D(t,i,\
    j-1))))));
#define S1(t,i,j) RX(t,i,j) = (((j-1== 0))?(((((((0.25)*((X(t-1,i,j+1))-\
    (X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
    (Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-\
    ((2.0)*(X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-\
    ((2.0)*(X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,\
    j+1))-(X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1))))+\
    ((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))))*(((((((0.25)*((X(t-1,i,j+2))-(X(t-1,i,j))))*((X(t-1,i,\
    j+2))-(X(t-1,i,j))))+(((Y(t-1,i,j+2))-(Y(t-1,i,j)))*((Y(t-1,i,j+2))-\
    (Y(t-1,i,j)))))*(((X(t-1,i+1,j+1))-((2.0)*(X(t-1,i,j+1))))+(X(t-1,\
    i-1,j+1))))+(((((0.25)*((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,\
    i+1,j+1))-(X(t-1,i-1,j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*\
    ((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))))*(((X(t-1,i,j+2))-((2.0)*\
    (X(t-1,i,j+1))))+(X(t-1,i,j)))))-(((((0.125)*((X(t-1,i+1,j+1))-\
    (X(t-1,i-1,j+1))))*((X(t-1,i,j+2))-(X(t-1,i,j))))+(((Y(t-1,i+1,\
    j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i,j+2))-(Y(t-1,i,j)))))*((((X(t-1,\
    i+1,j+2))-(X(t-1,i+1,j)))-(X(t-1,i-1,j+2)))+(X(t-1,i-1,j))))))*\
    (D(t,i,j))):(((-N+j+2== 0))?(((((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1))))+\
    ((((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
    (((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))))*\
    (D(t,i,j-1))))*(D(t,i,N-2)):(((((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1))))+\
    ((((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
```

```
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
    (((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j))))))*\
    (D(t,i,j-1))))+((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,\
    i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,\
    i+1,j))-(Y(t-1,i-1,j)))))*(((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1))))*(((X(t-1,i+1,j))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((X(t-1,i,j+1))-((2.0)*\
    (X(t-1,i,j))))+(X(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((X(t-1,i+1,j+1))-\
    (X(t-1,i+1,j-1)))-(X(t-1,i-1,j+1)))+(X(t-1,i-1,j-1)))))+\
    (((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
    i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
    j-2)))))*(((X(t-1,i+1,j-1))-((2.0)*(X(t-1,i,j-1))))+(X(t-1,i-1,\
    j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
    j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
    ((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((X(t-1,i,j))-((2.0)*(X(t-1,\
    i,j-1))))+(X(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
    i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
    (Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((X(t-1,i+1,j))-\
    (X(t-1,i+1,j-2)))-(X(t-1,i-1,j)))+(X(t-1,i-1,j-2)))))*((((0.25)*\
    ((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,i+1,j+1))-(X(t-1,i-1,\
    j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i+1,j+1))-\
    (Y(t-1,i-1,j+1))))))*(D(t,i,j)))))*(D(t,i,j))))));
#define S2(t,i,j) RY(t,i,j) = (((j-1== 0))?(((((((0.25)*((X(t-1,i,j+1))-\
    (X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
    (Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-\
    ((2.0)*(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-\
    ((2.0)*(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
    (X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
    (Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,\
    j+1))-(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
    ((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))))*(((((((0.25)*((X(t-1,i,j+2))-(X(t-1,i,j))))*((X(t-1,i,\
    j+2))-(X(t-1,i,j))))+(((Y(t-1,i,j+2))-(Y(t-1,i,j)))*((Y(t-1,i,j+2))-\
    (Y(t-1,i,j)))))*(((Y(t-1,i+1,j+1))-((2.0)*(Y(t-1,i,j+1))))+(Y(t-1,\
    i-1,j+1))))+(((((0.25)*((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,\
    i+1,j+1))-(X(t-1,i-1,j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*\
    ((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))))*(((Y(t-1,i,j+2))-((2.0)*\
    (Y(t-1,i,j+1))))+(Y(t-1,i,j)))))-(((((0.125)*((X(t-1,i+1,j+1))-\
    (X(t-1,i-1,j+1))))*((X(t-1,i,j+2))-(X(t-1,i,j))))+(((Y(t-1,i+1,\
    j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i,j+2))-(Y(t-1,i,j)))))*((((Y(t-1,\
    i+1,j+2))-(Y(t-1,i+1,j)))-(Y(t-1,i-1,j+2)))+(Y(t-1,i-1,j))))))*\
    (D(t,i,j))):(((-N+j+2== 0))?(((((((0.25)*((X(t-1,i,j+1))-(X(t-1,i,\
    j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-(Y(t-1,i,\
    j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-((2.0)*\
    (Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-(X(t-1,\
    i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,\
    i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-((2.0)*\
```

```
(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-(X(t-1,\
i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-(Y(t-1,\
i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,j+1))-\
(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
(((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
j-2)))))*(((Y(t-1,i+1,j-1))-((2.0)*(Y(t-1,i,j-1))))+(Y(t-1,i-1,\
j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((Y(t-1,i,j))-((2.0)*(Y(t-1,\
i,j-1))))+(Y(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
(Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((Y(t-1,i+1,j))-\
(Y(t-1,i+1,j-2)))-(Y(t-1,i-1,j)))+(Y(t-1,i-1,j-2)))))*((((0.25)*\
((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,\
j))))))*(D(t,i,j-1))))*(D(t,i,N-2)):((((((((((0.25)*((X(t-1,i,j+1))-\
(X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
(Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,\
j+1))-(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
(((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
j-2)))))*(((Y(t-1,i+1,j-1))-((2.0)*(Y(t-1,i,j-1))))+(Y(t-1,i-1,\
j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((Y(t-1,i,j))-((2.0)*(Y(t-1,\
i,j-1))))+(Y(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
(Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((Y(t-1,i+1,j))-\
(Y(t-1,i+1,j-2)))-(Y(t-1,i-1,j)))+(Y(t-1,i-1,j-2)))))*((((0.25)*\
((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+\
(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,\
j))))))*(D(t,i,j-1))))+((((((0.25)*((X(t-1,i+1,j))-(X(t-1,i-1,j))))*((X(t-1,\
i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))*((Y(t-1,\
i+1,j))-(Y(t-1,i-1,j)))))*(D(t,i,j)))*(((((((((0.25)*((X(t-1,i,j+1))-\
(X(t-1,i,j-1))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i,j+1))-\
(Y(t-1,i,j-1)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*(((Y(t-1,i+1,j))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i-1,j))))+(((((0.25)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i+1,j))-(X(t-1,i-1,j))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i+1,j))-(Y(t-1,i-1,j)))))*(((Y(t-1,i,j+1))-\
((2.0)*(Y(t-1,i,j))))+(Y(t-1,i,j-1)))))-(((((0.125)*((X(t-1,i+1,j))-\
(X(t-1,i-1,j))))*((X(t-1,i,j+1))-(X(t-1,i,j-1))))+(((Y(t-1,i+1,j))-\
(Y(t-1,i-1,j)))*((Y(t-1,i,j+1))-(Y(t-1,i,j-1)))))*((((Y(t-1,i+1,\
j+1))-(Y(t-1,i+1,j-1)))-(Y(t-1,i-1,j+1)))+(Y(t-1,i-1,j-1)))))+\
(((((((((0.25)*((X(t-1,i,j))-(X(t-1,i,j-2))))*((X(t-1,i,j))-(X(t-1,\
i,j-2))))+(((Y(t-1,i,j))-(Y(t-1,i,j-2)))*((Y(t-1,i,j))-(Y(t-1,i,\
j-2)))))*(((Y(t-1,i+1,j-1))-((2.0)*(Y(t-1,i,j-1))))+(Y(t-1,i-1,\
j-1))))+(((((0.25)*((X(t-1,i+1,j-1))-(X(t-1,i-1,j-1))))*((X(t-1,i+1,\
j-1))-(X(t-1,i-1,j-1))))+(((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))*\
((Y(t-1,i+1,j-1))-(Y(t-1,i-1,j-1)))))*(((Y(t-1,i,j))-((2.0)*(Y(t-1,\
i,j-1))))+(Y(t-1,i,j-2)))))-(((((0.125)*((X(t-1,i+1,j-1))-(X(t-1,\
i-1,j-1))))*((X(t-1,i,j))-(X(t-1,i,j-2))))+(((Y(t-1,i+1,j-1))-\
(Y(t-1,i-1,j-1)))*((Y(t-1,i,j))-(Y(t-1,i,j-2)))))*((((Y(t-1,i+1,j))-\
(Y(t-1,i+1,j-2)))-(Y(t-1,i-1,j)))+(Y(t-1,i-1,j-2)))))*((((0.25)*\
((X(t-1,i+1,j+1))-(X(t-1,i-1,j+1))))*((X(t-1,i+1,j+1))-(X(t-1,i-1,\
```

```
        j+1))))+(((Y(t-1,i+1,j+1))-(Y(t-1,i-1,j+1)))*((Y(t-1,i+1,j+1))-\
        (Y(t-1,i-1,j+1))))))*(D(t,i,j)))))))));
    #define S3(t,i,j) X(t,i,j) = (((t== 0) || (-N+j+1== 0) || (i== 0) ||\
        (-N+i+1== 0) || (j== 0))?X0(i,j):((X(t-1,i,j))+(RX(t,i,j))));
    #define S4(t,i,j) Y(t,i,j) = (((t== 0) || (-N+j+1== 0) || (i== 0) ||\
        (-N+i+1== 0) || (j== 0))?Y0(i,j):((Y(t-1,i,j))+(RY(t,i,j))));
    #define S5(t,i1,i2) RXM(t,0,0) = Tomcatv_RXM_reduce_1(ITACT,N,t,X0,Y0,D,\
        RX,RY,X,Y,RXM,RYM);
    #define S6(t,i1,i2) RYM(t,0,0) = Tomcatv_RYM_reduce_1(ITACT,N,t,X0,Y0,D,\
        RX,RY,X,Y,RXM,RYM);
    {
        int c1,c2,c3;
        for(c2=0;c2 <= N + -1;c2+=1){
            for(c3=0;c3 <= N + -1;c3+=1){
                S3((0),(c2),(c3));
                S4((0),(c2),(c3));
            }
        }
        for(c1=1;c1 <= ITACT;c1+=1){
            for(c3=0;c3 <= N + -1;c3+=1){
                S3((c1),(0),(c3));
                S4((c1),(0),(c3));
            }
            for(c2=1;c2 <= N + -2;c2+=1){
                S3((c1),(c2),(0));
                S4((c1),(c2),(0));
                for(c3=1;c3 <= N + -2;c3+=1){
                    S0((c1),(c2),(c3));
                    S1((c1),(c2),(c3));
                    S2((c1),(c2),(c3));
                    S3((c1),(c2),(c3));
                    S4((c1),(c2),(c3));
                }
                S3((c1),(c2),(N + -1));
                S4((c1),(c2),(N + -1));
            }
            for(c3=0;c3 <= N + -1;c3+=1){
                S3((c1),(N + -1),(c3));
                S4((c1),(N + -1),(c3));
            }
            S5((c1),(N),(N));
            S6((c1),(N),(N));
        }
    }
    #undef S0
    #undef S1
    #undef S2
    #undef S3
    #undef S4
    #undef S5
    #undef S6
}

#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double* D,double* RX,double* RY,double* X,double* Y,
    double*** RXM,double*** RYM)
#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RXM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double (*D)[NMAX],double (*RX)[NMAX],double (*RY)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM)
```

```
#endif
{
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RX(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        int c2,c3;
        for(c2=1;c2 <= N + -2;c2+=1){
            for(c3=1;c3 <= N + -2;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}

#if ALLOCATION == MALLOC_ARRAY
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double* D,double* RX,double* RY,double* X,double* Y,
    double*** RXM,double*** RYM)
#elif ALLOCATION == STATIC_ARY_ARY
double Tomcatv_RYM_reduce_1(long ITACT,long N,int t_p,double** X0,
    double** Y0,double (*D)[NMAX],double (*RX)[NMAX],double (*RY)[NMAX],
    double (*X)[NMAX],double (*Y)[NMAX],double*** RXM,double*** RYM)
#endif
{
    double reduceVar = LONG_MIN;
#define S0(t,k,l) {double __temp__ = abs(RY(t,k,l));\
    reduceVar = max(reduceVar,__temp__); }
    {
        int c2,c3;
        for(c2=1;c2 <= N + -2;c2+=1){
            for(c3=1;c3 <= N + -2;c3+=1){
                S0((t_p),(c2),(c3));
            }
        }
    }
    #undef S0
    return reduceVar;
}
#undef X0
#undef Y0
#undef D
#undef RX
#undef RY
#undef X
#undef Y
#undef RXM
#undef RYM
//Postamble
#undef MAX
#undef max
#undef MIN
#undef min
#undef CEILD
#undef ceild
#undef FLOORD
#undef floord
#undef CDIV
#undef FDIV
```

```
#undef LB_SHIFT
#undef MOD
#undef RADD
#undef RMUL
#undef RMAX
#undef RMIN
```

Script to modify AlphaZ-generated code

```
# Filename: fixAlphaZcode.py

import os
import sys
import time
import string
import socket
import subprocess

global OUTNAME
global OUTFILE
OUTNAME = 'null.out'
OUTFILE = open('null.out', 'w')

# ============================================================ #
#         Correct MOD function; add timing information         #
# ============================================================ #

def FixFile(path, dims, ops):
    def StrDimsOps(dims):
        mult_dim = '('
        for i in range(len(dims) - 1):
            mult_dim += str(dims[i]) + '*'
        mult_dim += str(dims[-1]) + ')'
        return ('    mflops = (((double) ops_per_iter*' + mult_dim +
            ') / run_time) / 1000000L;\n')

    old_code_file = open(path + '.c', 'r')
    old_code_list = old_code_file.readlines()
    old_code_file.close()
    charon = subprocess.call('rm ' + path + '.c', shell=True, stderr=OUTFILE)
    new_code = open(path + '.c', 'a')

    if 'Micah WALTER' not in old_code_list[2]:
        while len(old_code_list) > 1:
            if '// To compile this code,' in old_code_list[0]:
                new_code.write('// Micah WALTER\'s script edited ')
                new_code.write('this code too :-)\n')
            if '#define MOD(i,j)' in old_code_list[0]:
                new_code.write('#define MOD(i,j)  ((i)%(j))\n')
            elif '#include<omp.h>' in old_code_list[0]:
                new_code.write('#include <time.h>\n')
                new_code.write('#include <sys/time.h>\n')
                new_code.write(old_code_list[0])
            elif '//Memory Macros' in old_code_list[0]:
                new_code.write('#ifdef _CLUSTER_OPENMP\n')
                new_code.write('#define MALLOC kmp_sharable_malloc\n')
                new_code.write('#define FREE kmp_sharable_free\n')
                new_code.write('#else\n#define MALLOC malloc\n')
                new_code.write('#define FREE free\n#endif\n')
                new_code.write('\ndouble cur_time(void) {\n')
```

```
            new_code.write('\tstruct timeval tv;\n')
            new_code.write('\tstruct timezone tz;\n')
            new_code.write('\tgettimeofday(&tv, &tz);\n')
            new_code.write('\treturn tv.tv_sec + tv.tv_usec*1.0e-6;')
            new_code.write('\n}\n\n//Memory Macros\n')
        elif '// Parameter checking' in old_code_list[0]:
            new_code.write('\tint ops_per_iter = ' + str(ops))
            new_code.write('; // constant\n')
            new_code.write('\tdouble run_time, mflops, total = 0;\n')
            new_code.write(old_code_list[0])
        elif ('#define S' in old_code_list[0] and 'S0' not in
                old_code_list[0] and '{' in old_code_list[1]):
            new_code.write(old_code_list[0])
            new_code.write(old_code_list[1])
            new_code.write('\trun_time = cur_time();\n')
            del old_code_list[0]
        elif ('#undef S0' in old_code_list[1] and
                '#undef S1' in old_code_list[2]):
            new_code.write('\n\t/* Get and print MFLOPS */\n')
            new_code.write('\n\trun_time = cur_time() - run_time;\n')
            new_code.write(StrDimsOps(dims))
            new_code.write('\n\tprintf("%f MFLOPS, ", mflops);\n')
            new_code.write('\tprintf("%f s\\n", run_time);\n\n    }\n\n')
        else:
            if 'vaild' in old_code_list[0]:
                new_code.write(string.replace(old_code_list[0],
                    'vaild', 'valid'))
            elif 'malloc' in old_code_list[0]:
                new_code.write(string.replace(old_code_list[0],
                    'malloc', 'MALLOC'))
            elif 'free' in old_code_list[0]:
                new_code.write(string.replace(old_code_list[0],
                    'free', 'FREE'))
            else:
                new_code.write(old_code_list[0])
        del old_code_list[0]
    new_code.write(old_code_list[0] + '\n')
else:
    while len(old_code_list) > 0:
        new_code.write(old_code_list[0])
        del old_code_list[0]
new_code.close()

wrapper_path = path + '-wrapper.c'
if os.path.exists(wrapper_path):
    old_wrapper_file = open(wrapper_path, 'r')
    old_wrapper_list = old_wrapper_file.readlines()
    old_wrapper_file.close()
    charon = subprocess.call('rm ' + wrapper_path, shell=True,
        stderr=OUTFILE)
    new_wrapper = open(wrapper_path, 'a')

    if 'Micah WALTER' not in old_wrapper_list[2]:
        while len(old_wrapper_list) > 0:
            if '// To compile this code,' in old_wrapper_list[0]:
                new_wrapper.write('// Micah WALTER\'s script edited')
                new_wrapper.write('this code too :-)\n')
            elif ('Execution' in old_wrapper_list[0] and
                    '//' not in old_wrapper_list[0]):
                new_wrapper.write('// ' + old_wrapper_list[0])
            elif '#define MOD(i,j)' in old_wrapper_list[0]:
```

```
                    new_wrapper.write('#define MOD(i,j)  ((i)%(j))\n')
                elif 'malloc' in old_wrapper_list[0]:
                    new_wrapper.write(string.replace(old_wrapper_list[0],
                        'malloc', 'MALLOC'))
                elif '//Memory Macros' in old_wrapper_list[0]:
                    new_wrapper.write('#ifdef _CLUSTER_OPENMP\n')
                    new_wrapper.write('#define MALLOC kmp_sharable_malloc\n')
                    new_wrapper.write('#define FREE kmp_sharable_free\n')
                    new_wrapper.write('#else\n#define MALLOC malloc\n')
                    new_wrapper.write('#define FREE free\n#endif\n')
                    new_wrapper.write('//Memory Macros\n')
                else:
                    if 'vaild' in old_wrapper_list[0]:
                        new_wrapper.write(string.replace(old_wrapper_list[0],
                            'vaild', 'valid'))
                    elif 'free' in old_wrapper_list[0]:
                        new_wrapper.write(string.replace(old_wrapper_list[0],
                            'free', 'FREE'))
                    else:
                        new_wrapper.write(old_wrapper_list[0])
                del old_wrapper_list[0]
        else:
            while len(old_wrapper_list) > 0:
                new_wrapper.write(old_wrapper_list[0])
                del old_wrapper_list[0]
        new_wrapper.close()

# ============================================================ #
#                   *  EDITING THE MAKEFILE  *                 #
# ============================================================ #

# This part of the script recognizes whether we are on the system
# with gcc or the one with icc installed.
def GetCompiler(path):
    makepath = RemoveFilename(path) + 'Makefile'
    if os.path.exists(makepath):
        if ('terri' in socket.gethostname()
                or 'sherri' in socket.gethostname()
                or 'snake' in socket.gethostname()
                or 'ziff' in socket.gethostname()):
            print 'Using icc in new Makefile'
            return '/opt/intel/cce/10.1.022/bin/icc'
        elif ('apu' in socket.gethostname()
                or 'maggie' in socket.gethostname()
                or 'selma' in socket.gethostname()
                or 'krabappel' in socket.gethostname()
                or 'burns' in socket.gethostname()):
            print 'Using gcc in new Makefile'
            return 'gcc'
        else:
            print 'No compiler chosen for Makefile'
            return None
    else:
        print 'No Makefile to edit'
        return None

def Make(path, compiler_to_use):
    makepath = RemoveFilename(path) + 'Makefile'
    if os.path.exists(makepath):
        old_make_file = open(makepath, 'r')
        old_make_list = old_make_file.readlines()
```

```
        old_make_file.close()
        charon = subprocess.call('rm ' + makepath, shell=True,
            stderr=OUTFILE)

        new_make = open(makepath, 'a')
        while len(old_make_list) > 0:
            if 'CC=' in old_make_list[0]:
                new_make.write('CC=' + compiler_to_use + '\n')
            elif '-fopenmp' in old_make_list[0] and 'icc' in compiler_to_use:
                new_make.write(string.replace(old_make_list[0],
                    '-fopenmp', '-openmp'))
            elif '-openmp' in old_make_list[0] and 'gcc' in compiler_to_use:
                new_make.write(string.replace(old_make_list[0],
                    '-openmp', '-fopenmp'))
            else:
                new_make.write(old_make_list[0])
            del old_make_list[0]
        new_make.close()

# ============================================================ #
#                    *  EXTERNAL FUNCTIONS  *                  #
# ============================================================ #

# This subfunction gets the first character in a string,
# neglecting whitespace
def FirstBlackChar(charseq):
    if charseq == '':
        return ''
    elif charseq[0] == ' ' or charseq[0] == '    ':
        return FirstBlackChar(charseq[1:])
    else:
        return charseq[0]

def GetPath():
    userPath = raw_input('Path and filename of file to edit: ')
    while userPath[-1] == ' ':
        userPath = userPath[:-1]
    if userPath == '':
        sys.exit()
    elif len(userPath) > 0 and userPath[-2:] == '.c':
        userPath = userPath[:-2]
        if userPath[-8:] == '-wrapper':
            userPath = userPath[:-8]
    if os.path.exists(userPath + '.c'):
        return userPath
    else:
        return GetPath()

def GetDimensions(path):
    dimString = GetParameters(GetAlphabets(path, GetScript(path)))
    print 'Using parameters ' + dimString
    dimString = '[\'' + string.replace(dimString, ',', '\', \'') + '\']'
    return eval(dimString)

def GetOperations(path):
    cfile = open(path + '.c', 'r')
    clist = cfile.readlines()
    cfile.close()
    current_flop_max = 0
    for i in range(len(clist)):
        if '#define S' in clist[i]:
```

```
            this_flop_max = ExtractFlops(clist[i])
            if this_flop_max > current_flop_max:
                current_flop_max = this_flop_max
    # Fallback if we didn't get any FLOPS
    if current_flop_max == 0 or current_flop_max == 1:
        return input('Operations per iteration: ')
    elif 'tomcatv' in path.lower():
        print 'Using 68 ops per iter specially for Tomcatv (=\'.\'=)'
        return 68    # Magic number hack
    else:
        print 'Using ' + str(current_flop_max) + ' ops per iter'
        return current_flop_max

def RemoveFilename(path):
    while path[-1] != '/':
        path = path[:-1]
    return path

def RemoveTemp():
    OUTFILE.close()
    charon = subprocess.call('rm ' + OUTNAME, shell=True)
    sys.exit()

# This function returns the script file as a list, given the path.
# This will be useful for finding the Alphabets file, which gives us
# the parameters, so that they don't have to be input manually.
def GetScript(path):
    directory = RemoveFilename(path)
    file_list = os.listdir(directory)
    list_of_script_names = []
    for i in range(len(file_list)):
        if file_list[i][-3:] == '.cs':
            list_of_script_names += [file_list[i]]
    if len(list_of_script_names) == 0:
        main()
    elif len(list_of_script_names) == 1:
        script_name = list_of_script_names[0]
    else:
        for i in range(len(list_of_script_names)):
            print ' ' + str(i + 1) + '. ' + list_of_script_names[i]
        user_script_choice = input('Enter your choice: ')
        while (user_script_choice < 1 or
                user_script_choice > len(list_of_script_names)):
            user_script_choice = input('Enter your choice: ')
        script_name = list_of_script_names[user_script_choice - 1]

    script_file = open(directory + script_name, 'r')
    script_list = script_file.readlines()
    script_file.close()
    return script_list

# This function extracts the portion of 'line'
# in between 'lchar' and 'rchar'.
def Extract(line, lchar, rchar):
    lcharpos = string.find(line, lchar)
    rcharpos = string.find(line, rchar, lcharpos + 1)
    return line[lcharpos + 1:rcharpos]

# This function returns the Alphabets file as a list,
# given a path and a script file (as a list).
def GetAlphabets(path, script_list):
```

```
    i = 0
    while 'program =' not in script_list[i]:
        i += 1
    alpha_path = RemoveFilename(path) + Extract(script_list[i], '"', '"')
    alpha_file = open(alpha_path, 'r')
    alpha_list = alpha_file.readlines()
    alpha_file.close()
    return alpha_list

# This function gets the number of floating-point operations in a line
def ExtractFlops(line):
    # Here we find a right parenthesis and recursively count it
    def FlopsIn(linepart):
        flopscount = 0
        for i in range(len(linepart)):
            if linepart[i] == ')':
                if FirstBlackChar(linepart[i + 1:]) in '+-*/':
                    flopscount += 1
        return flopscount
    # Get the number of FLOPS in the line after the equals sign
    eqpos = string.find(line, '=')
    return FlopsIn(line[eqpos + 1:])

# This function returns the parameters needed for fixing AlphaZ-generated
# C code, given an Alphabets file (as a list).
def GetParameters(alpha):
    i = 0
    while 'affine' not in alpha[i]:
        i += 1
    dimString = Extract(alpha[i], '{', '|')
    # Magic string: see if we are doing Tomcatv
    if 'ITACT' in dimString:
        dimString = string.replace(dimString, 'N', 'N,N')
    return dimString

def AlreadyFixed(path):
    old_code_file = open(path + '.c', 'r')
    old_code_list = old_code_file.readlines()
    old_code_file.close()
    for i in range(len(old_code_list)):
        if 'WALTER' in old_code_list[2]:
            print 'The file mentioned has already been edited by this',
            print 'script;\nplease recompile the file before running',
            print 'the script again.'
            return True
    return False

# ============================================================ #
#                          *  MAIN  *                          #
# ============================================================ #

def main():
    PATH = GetPath()

    if not AlreadyFixed(PATH):
        DIMS = GetDimensions(PATH)
        OPS = GetOperations(PATH)
        FixFile(PATH, DIMS, OPS)

    MAKE = GetCompiler(PATH)
    if MAKE != None:
```

```
        Make(PATH, MAKE)

    RemoveTemp()

if __name__ == '__main__':
    main()
```

Script to merge AlphaZ function and wrapper files

```
# Filename: generateMerge.py

import os
import sys
import time
import string
import subprocess
from fixAlphaZcode import *

def Merge(path, mergefile):
    # Open function and wrapper files and convert them into lists
    Function = open(path + '.c', 'r')
    Wrapper = open(path + '-wrapper.c', 'r')
    funclist = Function.readlines()
    wraplist = Wrapper.readlines()
    Function.close()
    Wrapper.close()

    def LoopMerge(head, tail, prog):
        # About head and tail: if the lines at the top of the two
        # files are equally permissible (i.e. each of them is unique),
        # we continue with the one that has been in use.
        # About prog: this boolean value keeps track of whether a
        # function is in progress; if so, that function shouldn't stop
        # until the first lines are equal again.
        if len(head) > 0 and len(tail) > 0:
            if 'main' in head[0]:
                (head, tail) = (tail, head)

            if prog == False:
                if head[0][0] != '    ' and head[0][-2:] == '{\n':
                    mergefile.write(head[0])
                    del head[0]
                    LoopMerge(head, tail, True)
                elif head[0] == tail[0]:
                    mergefile.write(head[0])
                    del funclist[0]
                    del wraplist[0]
                    LoopMerge(head, tail, prog)
                elif OccursIn(head[0], tail) and not OccursIn(tail[0], head):
                    mergefile.write(tail[0])
                    del tail[0]
                    LoopMerge(tail, head, prog)
                else:
                    mergefile.write(head[0])
                    del head[0]
                    LoopMerge(head, tail, prog)
            else:
                if head[0][0] == '}':
                    prog = False
                mergefile.write(head[0])
```

```
                del head[0]
                LoopMerge(head, tail, prog)

    LoopMerge(funclist, wraplist, False)
    if len(funclist) > 0:
        for i in range(len(funclist)):
            mergefile.write(funclist[i])
    if len(wraplist) > 0:
        for i in range(len(wraplist)):
            mergefile.write(wraplist[i])

def OccursIn(Line, List):
    for i in range(len(List)):
        if Line == List[i]:
            return True
    return False

def main():
    PATH = GetPath()
    FixFile(PATH, GetDimensions(PATH), GetOperations(PATH))
    if os.path.exists(PATH + '-merged.c'):
        charon = subprocess.call('rm ' + PATH + '-merged.c', shell=True)
    Merged = open(PATH + '-merged.c', 'a')
    Merge(PATH, Merged)
    Merged.close()
    RemoveTemp()

if __name__ == '__main__':
    main()
```

Script to compile original Fortran Tomcatv

```
# Filename: compile_and_run.py

import os
import sys
import time
import string
import socket
import subprocess

global ERRORS
global REMTEMP

FILENAME = 'tomcatv-ours'
ERRORS = open('null.err', 'w')
REMTEMP = True

# ============================================================ #
#      *  Convert tomcatv-ours.f into C code using f2c  *      #
# ============================================================ #

def Convert(filename):
    charon = subprocess.call('f2c ' + filename + '.f', shell=True,
stderr=ERRORS)

# ============================================================ #
#     *  Fix conversion error with MAIN__ function name  *     #
# ============================================================ #
```

```
def FixMain(filename, ITACT, N):
    old_code = open(filename + '.c', 'r')
    old_list = old_code.readlines()
    old_code.close()
    if os.path.exists(filename + '-fixed.c'):
        charon = subprocess.call('rm ' + filename + '-fixed.c', shell=True)
    new_code = open(filename + '-fixed.c', 'a')
    while len(old_list) > 0:
        if '+ + +' in old_list[0]:
            new_code.write('n = ' + str(N) + ';\n')
            new_code.write('itact = ' + str(ITACT) + ';\n')
        else:
            if 'MAIN__' in old_list[0]:
                old_list[0] = string.replace(old_list[0], 'MAIN__', 'main')
            new_code.write(old_list[0])
        old_list = old_list[1:]
    new_code.close()

# ============================================================ #
#              *  Compile new C code using gcc  *              #
# ============================================================ #

def Compile(filename):
    # This part of the script recognizes whether we are on the system
    # with gcc or the one with icc installed.
    def GetCompiler():
        if ('terri' in socket.gethostname()
                or 'sherri' in socket.gethostname()
                or 'snake' in socket.gethostname()
                or 'ziff' in socket.gethostname()):
            print 'Using icc'
            return '/opt/intel/cce/10.1.022/bin/icc '
        else:
            print 'Using gcc'
            return 'gcc '

    charon = subprocess.call(GetCompiler() + filename + '-fixed.c -O3 -o '
        + filename, shell=True, stderr=ERRORS, stdout=ERRORS)

# ============================================================ #
#       *  Run compiled code and print execution time  *       #
# ============================================================ #

def RunTime(filename, ITACT, N):
    ops_per_iter = 68
        # We are assigning 68 to ops_per_iter since that is what
        # the equivalent AlphaZ-generated program uses.
    start_time = time.time()
    charon = subprocess.call('./' + filename, shell=True, stderr=ERRORS,
stdout=ERRORS)
    end_time = time.time()
    elapsed_time = '%.6f' % (end_time - start_time)
    mflops = '%.6f' % (ITACT * N * N * ops_per_iter / (1000000 *
        (end_time - start_time)))
    print mflops + ' MFLOPS, ' + elapsed_time + ' s'
    if end_time - start_time < 1 and N > 512:
        print 'The execution time seems too fast. Please check null.err'
        print 'to see if there\'s anything written there.'
        ERRORS.close()
        sys.exit()
```

```
# ============================================================ #
#                          *  MAIN  *                          #
# ============================================================ #

def main():
    N = 513     # Default parameters, not magic numbers
    ITACT = 30  # See above
    parameters = raw_input('Parameters (ITACT, N): ')
    if len(parameters) > 0:
        if ',' in parameters:
            parameters = string.replace(parameters, ',', ' ')
        if '  ' in parameters:
            parameters = string.replace(parameters, '  ', ' ')
        param_list = parameters.split(' ')
        ITACT = eval(param_list[0])
        N = eval(param_list[1])
    # The following line only needs to be run once.
    Convert(FILENAME)
    FixMain(FILENAME, ITACT, N)
    Compile(FILENAME)
    RunTime(FILENAME, ITACT, N)

    ERRORS.close()

    if REMTEMP == True:
        charon = subprocess.call('rm null.err', shell=True)
    else:
        REMOVE = raw_input('Remove temporary files? ')
        if len(REMOVE) > 0 and REMOVE[0].lower() == 'y':
            charon = subprocess.call('rm null.err', shell=True)

if __name__ == '__main__':
    main()
```