**TITLE: SOME COMPUTATIONAL CHALLENGES OF DEVELOPING
EFFICIENT PARALLEL ALGORITHMS FOR DATA-
DEPENDENT COMPUTATIONS IN THERMAL-HYDRAULICS
SUPERCOMPUTER APPLICATIONS**

LA-UR--92-859

DE92 011368

AUTHOR(S): S. B. Woodruff

**SUBMITTED TO: 5th International Topical Meeting on Nuclear Reactor Thermal
Hydraulics
September 21-24, 1992
Salt Lake City, UT**

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article, the publisher recognizes that the U. S. Government retains a nonexclusive, royalty free license to publish or reproduce the published form of this contribution, to allow others to do so, for U. S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U. S. Department of Energy.

Los Alamos

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

MAILED

2

SOME COMPUTATIONAL CHALLENGES OF DEVELOPING EFFICIENT PARALLEL ALGORITHMS FOR DATA-DEPENDENT COMPUTATIONS IN THERMAL-HYDRAULICS SUPERCOMPUTER APPLICATIONS

**S. B. Woodruff
Los Alamos National Laboratory**

ABSTRACT

The Transient Reactor Analysis Code (TRAC), which features a two-fluid treatment of thermal-hydraulics, is designed to model transients in water reactors and related facilities. One of the major computational costs associated with TRAC and similar codes is calculating constitutive coefficients. Although the formulations for these coefficients are local, the costs are flow-regime- or data-dependent; i.e., the computations needed for a given spatial node often vary widely as a function of time. Consequently, poor load balancing will degrade efficiency on either vector or data parallel architectures when the data are organized according to spatial location.

A standard method for treating data-dependent models on vector architectures has been to use gather operations (or indirect addressing) to sort the nodes into subsets that (temporarily) share a common computational model. However, this method is not effective on distributed memory data parallel architectures, where indirect addressing involves expensive communication overhead. Another serious problem with this method involves software engineering challenges in the areas of maintainability and extensibility. For example, a hand-tuned implementation may have to be completely rewritten when the decision tree governing the sorting is modified even slightly. Using an example based on the calculation of the wall-to-liquid and wall-to-vapor heat-transfer coefficients for three nonboiling flow regimes, we describe how the use of the Fortran 90 WHERE construct and automatic inlining of functions can be used to ameliorate this problem while improving both efficiency and software engineering.

Unfortunately, a general automatic solution to the load-balancing problem associated with data-dependent computations is not yet available for massively parallel architectures. We discuss why developers should either wait for such solutions or consider alternative numerical algorithms, such as a neural net representation, that do not exhibit load-balancing problems.

1.0. INTRODUCTION

The Transient Reactor Analysis Code (TRAC), which features a two-fluid treatment of thermal-hydraulics, is designed to model transients in water reactors and related facilities. TRAC simulations of postulated off-normal scenarios are used for safety studies of nuclear reactors. Because one such simulation can take on the order of 10^{12} floating point operations, TRAC is currently a supercomputer application for this purpose; that is, a calculation of this type would require several hours at a sustained performance of 100 million floating point operations per second (MFlops). The advantages of better performance are obvious and have been discussed previously.¹ These include the possibilities for accident mitigation through faster-than-real-time simulation and operator training with realistic simulators.

Thermal-hydraulics encompasses three major processes: fluid flow or convection, thermal response or conduction within fixed heat structures surrounding or embedded in the fluid, and heat transfer between the fluid and the heat structures. TRAC models these processes by providing a number of logically independent finite-difference grids that communicate information with each other. There is one grid to simulate the fluid flow and an arbitrary number of additional small grids of different shapes to simulate the heat structures. (It is assumed that there are no significant conduction paths between the individual heat structures.) The fluid grid consists of irregularly coupled regular meshes* corresponding to a general or irregular sparse matrix; the additional grids consist of regular meshes corresponding to banded or regular sparse matrices.

Improving the computational cost of TRAC is quite difficult because the computational effort is spread out over thousands of lines of source code and, like other complex system codes, contains numerous algorithms with differing computational requirements.^{2,3} These algorithms include linear algebra on small-dense, small-banded, and large irregular matrices; spatially local data-independent algorithms; and spatially local data-dependent algorithms. Unlike many other grid-based codes that also use implicit schemes, linear algebra is generally only a small portion of the total computational cost.** This is partly because of the small number of nodes in the grids, typically no more than several thousand, and partly because of the complexity of the rest of the computation. In fact, the computational costs are dominated by the spatially local computations, especially the data-dependent ones that require inter-grid communication. This report focuses on parallel implementation of the last class, i.e., where the computational domain is *inhomogeneous* in that different nodes require widely varying amounts of computation.⁴ In addition, the character of the inhomogeneity is both problem- and time-dependent for a given

*This nomenclature was suggested to me by Lorie Liebrock of Rice University.

**The exact percentage is problem-dependent and would increase if other parts of the computation were carried out more efficiently; a value of 10% is currently typical for large systems.

problem. Thermal-hydraulics is not the only application area to exhibit this well-known problem.^{5,6}

This paper is organized as follows. In Sec. 2, we discuss high-performance computing issues, including typical parallel programming paradigms and the recent proposal for a set of standard extensions to Fortran. Section 3 details a typical data-dependent algorithm in TRAC and uses this example to explore implementation issues. In Sec. 4, we propose a new implementation involving neural nets that might be useful on data parallel architectures. A summary and conclusions are presented in the final section.

2.0. HIGH-PERFORMANCE COMPUTING ISSUES

A recent trend in high-performance computing has been to develop implementations that are portable across different architectures. This allows problems to be run in the most cost-effective manner.⁷ There is an increasing realization that continual tuning to the latest compiler and/or hardware is not an effective use of programmer effort. (This is not meant to imply that the matching of architectures to algorithms is no longer necessary but merely that the amount of architecture-specific coding should be reduced significantly.) A recent effort by a coalition of industrial and academic groups, the High Performance Fortran Forum (HPFF), to develop a set of standard extensions to Fortran is a major step in this direction.⁸ Although these standards have not yet been agreed upon, it is clear that they will rely heavily on Fortran 90 array syntax.

We describe here three common parallel programming paradigms: Single-Instruction Multiple-Data (SIMD), Single-Program Multiple-Data (SPMD), and Multiple-Instruction Multiple-Data (MIMD). In the discussion below, we compare these programming paradigms and alternative implementations of the same paradigm on different machine architectures, in particular, the CRAY Y-MP, which is a shared-memory multiprocessor with vector-mode instructions available on each processor, and the CM-2 and CM-5, which are distributed-memory massively parallel machines.*

With respect to the computations described in this report, the significant distinction between shared and distributed memory is for algorithms, such as gather-scatter, that would require interprocessor data movement on the distributed-memory machines. The major consideration in designing computing strategies for a machine that has a few powerful, expensive central processing units vs one that has thousands of less powerful, inexpensive processing units is that on the former machine, multitasking does not generally improve the price/performance ratio, and

*CRAY and CRAY Y-MP are registered trademarks of Cray Research Inc.; Connection Machine is a registered trademark, and CM-2 and CM-5 are trademarks of Thinking Machines Corporation.

on the latter machine, the use of the large number of processors is essential to achieving supercomputer computational times.

The SIMD paradigm carries out the same instruction sequence for all of the elements of a data set. A SIMD computation may be done in a pipelined manner on a single processor on a vector machine such as a CRAY Y-MP or simultaneously on the multiple processors of a data parallel machine such as a CM-2. We use the SIMD designation especially for the case of instruction sequences that do not contain conditional logic, although vectorized loops can contain conditional logic and purely SIMD data parallel architectures provide for the exclusion of certain data elements from a calculational sequence based on their values. (In this latter case, the processor containing those elements "drops out" while the calculation is occurring.) A similar mechanism occurs on a vector machine using conditional vector merge hardware. Here, both branches of the computation are done for all elements of the data set, and then the conditional logic is used to select the proper result to be associated with each data element. Thus, with either mechanism, the wall-clock time is proportional to the sum of the costs of both branches.

A SPMD paradigm is a generalization of the SIMD paradigm in that the same instruction sequence can proceed asynchronously depending on the values of the data elements. A simple example of this occurs with use of a WHERE construct* on a data parallel multiprocessor such as the CM-5, where the two branches of the computation take place simultaneously with synchronization occurring automatically at the END WHERE. (It should be noted that nested WHERE statements are not part of Fortran 90, but they are being considered by the HPFF and would make longer asynchronous sequences possible.) This implementation of the SPMD paradigm does not necessarily lead to load balancing among the processors as the wall-clock time will correspond to whichever branch takes the longest. The SMPD paradigm is implemented in a different manner on a single processor of a vector machine such as the CRAY Y-MP using gather-scatter/compressed index hardware. Here the computer dynamically creates lists of the indices of those data elements requiring a certain branch of the computation, gathers the data, carries out the computation, and then uses the lists to scatter the results to the proper locations. Naturally, there is some overhead in creating the index lists and accessing the data indirectly. (This implementation also can be used on a MIMD machine.)

Both of the above-described SMPD implementations share the advantage that they are transparent to the programmer so that the same type of implementation can be used on either machine.** However, unless the programmer has a clear under-

*WHERE is simply an "if-then-else" construct applied to arrays; see also Sec. 3.1

**With CRAY's CFT77 compiler, the choice between conditional vector merge and gather-scatter/compressed index is not under the control of the programmer; the compiler makes the choice based on machine restrictions and the perceived efficiency of the alternatives.

standing of what the machine is doing, opportunities for improving computational efficiency may be overlooked.

The final paradigm, MIMD, is a generalization of SPMD where an entirely different sequence of instructions can take place on each processor. However, once again, the implementation of the paradigm depends on whether one is considering machines with a few (2 to 32) powerful processors or with many less powerful ones. In the former case, MIMD machines often use a *fork-join* programming model with load balancing handled by the operating system, whereas in the latter case, such machines often use a *work sharing* model.⁹ Obviously, the MIMD paradigm is the most flexible and also the most difficult to use and debug because the synchronization of tasks must be handled on a case-by-case basis by the programmer or a preprocessor. Although preprocessors are getting increasingly powerful at inserting the necessary directives, they still have some disadvantages with respect to software engineering issues. For example, preprocessors often change the source code. This means that the programmer would have to be familiar with the new source as well as the old if debugging is necessary. In addition, preprocessors are sometimes unable to extract the full parallelism when interprocedural analysis is required. Nonetheless, there has been considerable progress in this area. Possibly the ideal situation is one where the programmer has the option of improving the code aesthetics and readability by inserting multitasking directives and the preprocessor checks their validity by applying a data-flow analysis.

In summary, the single-processor implementation of the SPMD paradigm and the fork-join implementation of the MIMD paradigm appear best suited for the problem of efficiently implementing local data-dependent calculations. The SIMD paradigm is not well-suited to the presence of numerous conditionals, and the data-parallel implementation of the SPMD paradigm has problems with load balancing. The proposed work-sharing implementations of the MIMD paradigm are too new to evaluate. Moreover, the use of heroic load-balancing measures (such as one or more of the many suggested in Ref. 5) to fit an implementation onto an inappropriate architecture seem ultimately fruitless if measured by such software engineering attributes as maintainability and extensibility. Consequently, as detailed in the following section, we recommend clear implementations that could allow the compiler and run-time system to do much of the optimization.

3.0. DATA-DEPENDENT COMPUTATIONS IN TRAC.

TRAC requires the computation of a large number of local, data-dependent values at each node of the fluid and heat-transfer grids. These include the thermodynamic and transport properties and the constitutive coefficients arising from the closure relationships. The constitutive coefficients, including those for wall shear, interfacial shear, interfacial heat transfer, and wall-to-fluid heat transfer, exhibit the largest effect of data dependence on computational effort with the heat-transfer coefficients showing the most variation. This is a result of the numerous flow regimes that can occur in two-phase flow as well as to the highly nonlinear relationship between the

fluid properties, the wall temperatures, and the constitutive coefficients. TRAC provides for eight heat-transfer regimes; the three nonboiling regimes are used here as an illustration.

As stated earlier, heat structures that contact the fluid at their walls may be associated arbitrarily with nodes in the fluid grid. Because of the irregular nature of this connectivity, each node on a heat structure wall or surface must have a pointer to the fluid grid node with which it is associated. This alone presents some difficulties for distributed memory architectures, although some compilers can optimize even irregular time-independent interprocessor communication.* (To simplify the discussion, we ignore any additional complications arising from a lack of registration between the heat structure grids and the fluid grid, which necessitates an interpolation among several fluid grid nodes.) Figure 1 shows the computational flow in procedural form for the calculation of HV and HL, i.e., the wall-to-vapor and wall-to-liquid heat-transfer coefficients for one node of the heat structure grids. In this figure, MASKn represents a logical variable, Fn (a local function of both wall node and associated fluid node properties), and Tn (a temporary variable). The logical expressions that define the flow regimes result from a complex sorting process (not shown) that depends on a number of factors, including the wall and fluid temperatures and their relationships to the local saturation temperature. The functions are complicated ones that typically involve tens of floating point operations.

An examination of Fig. 1 indicates the great variation in the amount of work that depends on the truth-value of the logical expressions. For example, the least amount of work occurs if MASK1 is false because only F6 needs to be calculated, whereas the most work occurs when all of the logical expressions are true (assuming that F3 is more expensive than F4). Figure 1, which shows only a small fragment of the decision logic in the constitutive models in TRAC, also suggests the complexity of this logic.

Figure 2 shows the same computation as Fig. 1 but rewritten in a Fortran 90-like pseudocode using the WHERE statement as it is used in the Connection Machine subset of Fortran 90.¹⁰ Here the variables, including the logical variables and the function references, should be taken to represent conforming arrays. The syntax of a WHERE statement is

WHERE (mask-expression) array-assignment

and its meaning is that for all elements of the array "mask-expression" that are true, any computation embedded in "array-assignment" should be carried out and the

*The connectivity also can be time-dependent when it is necessary to rezone the heat structures to resolve large temperature gradients. This commonly occurs during reflood. Time-dependent connectivity also would occur in a code that considered severe transients involving relocation of the heat structures.

results assigned to the corresponding elements of "array." Here we have rearranged the coding by hand to exhibit the maximum parallelism. Note that the two instances of the MAX function were merged as were the two instances of F6; this required treating the scaling involving T4 as a separate operation.

Applying a data-flow analysis to Fig. 1 allows one to determine which computations can be done in parallel. (Because the calculations are known to be local, the data-flow analysis for a single heat-transfer coefficient is the same as for all of them.) In this case, TASK1 and TASK2 of Fig. 2 can be carried out in parallel, but TASK3 can begin only after both TASK1 and TASK2 are complete. In addition, each of the separate WHERE statements in these two tasks could be carried out in parallel except that the final WHERE statement of TASK1 cannot be done until all of the others in that task are complete. The last two assignment statements of TASK3 also could be carried out in parallel. Such a detailed description of the parallelism, which is only needed for a MIMD machine of course, sometimes can be expressed using vendor-specific compiler directives, but this can be a tedious and error-prone process. (The computer also could detect this parallelism as long as it was able to see that none of the function calls changed the input data and was able to properly interpret the masks.)

It is a matter of personal preference whether the code fragment in Fig. 1 or Fig. 2 is more readable or even whether the slight rearrangement resulting from the function merging described above should have been made. However, it is clear the latter implementation exhibits the maximum parallelism in a form that could be used by an automated load balancer or other software support tools.

Figure 3 shows the conventional implementation of a WHERE construct in FORTRAN 77 using a compressed index. This example is for illustration only because the most efficient method for compressing indices varies from machine to machine. This should clarify the information in Sec. 2 and show why a vector supercomputer with gather-scatter hardware and/or a multiprocessor with a self-scheduled fork-join MIMD implementation can handle this problem quite easily.

Figure 3 also involves an important issue in computational efficiency and software engineering. If the function subroutine that is called inside the loop is an elemental function, that is, all of its arguments and its return value are scalars, then the function easily can be inlined by some compilers. This has the computational advantage of preventing the overhead of the subroutine call as well as the software engineering advantage that there are no data structure details embedded in the implementation of the function. Such a function is almost identical to a Fortran 90 function that has been designed for use in an array-assignment with an interface block that specifies all of its arguments and its return value to be arrays of the same shape.

In summary, our approach to implementing data-dependent calculations has the following features: use of Fortran 90 array syntax and the WHERE construct, separa-

tion of decision logic from computation (even when that means recomputing a few intermediate variables), and highly modular source code.

4.0. ALTERNATIVE ALGORITHMS

As discussed above, the inhomogeneity of the computational domain and other characteristics of the computation make load balancing the data-dependent calculations in TRAC difficult on a data parallel computer architecture. However, the popularity and apparent cost-effectiveness of such architectures implies that they should not be ruled out without careful examination. This leads one to consider alternative algorithms for the data-dependent calculations in TRAC. Two other possible implementations are table lookup and analytic approximation. A table lookup does not seem feasible in this case because of the complex, nonlinear nature of the correlations. In addition, table lookups are also difficult on data parallel architectures. Again, because of the nature of the correlations, including the numerous regimes, a standard analytic approximation such as a piecewise polynomial would not be useful. However, this is just the type of problem that neural networks can handle.

It recently was reported that neural networks can correlate heat-transfer data.¹¹ The examples given in this reference are not as complex as the full set of heat-transfer correlations and flow regimes included in TRAC. However, good results were reported with the use of extremely small networks consisting of 12 to 16 neurons to reproduce a highly nonlinear relationship between three independent variables (input vector) and the dependent variables (output vector) for natural convection along slender vertical cylinders. Based on these results and the well-known successes of neural networks in other applications, it is not unreasonable to expect that a sufficiently large neural network could be trained to reproduce accurate heat-transfer coefficients corresponding to the TRAC correlations. When a network is trained, the presentation of the various input vectors and the determination of the output vectors are inherently data parallel as the operation of the trained network is not data-dependent.

We are planning to test this idea using the Connectionist Normalized Linear Spline network.¹² The primary difficulty anticipated with this method is actually in the area of software engineering and code developer acceptance rather than accuracy or computational efficiency. The current implementation (discussed above) is reasonably intuitive in that it parallels the way a heat-transfer expert breaks down the problem. In attempting to improve the predictive capability of a code like TRAC, the expert compares the predictions with data and uses heuristic judgments to make changes to the parameters in the correlations. This process is repeated until the results are satisfactory. Moreover, although this fact is not widely appreciated by persons not actively engaged in the maintenance of large scientific codes, such a process generally needs to be carried out repeatedly over a period of time. (This results from a number of factors, including code bugs, new experiments leading to better data, changes in the response function because of nonlinear interactions with other

code changes, and the need for extensions as an adjunct to other enhancements.) Consequently, any new implementation of data-dependent computations must not reduce the effectiveness of the expert who is making modeling improvements. This probably means that the procedure for training the network needs to be a part of TRAC, rather than an auxiliary or stand-alone code. In addition, TRAC should retain the option of using the coefficients obtained directly from the correlations rather than from the trained network to simplify the work of the expert as well as to provide for on-going checking and improvement of the neural network methodology itself.

5.0. SUMMARY AND CONCLUSIONS

The world of supercomputing is currently in a state of flux that creates problems for the scientific applications programmer requiring high performance. Affordable desk-top workstations are promising 10 or more Mflops in scalar mode. Distributed computing involving networks of workstations is becoming a feasible option. Massively parallel supercomputers using a variety of architectures are commercially available. The speed and user-friendliness of conventional vector multiprocessor supercomputers is increasing. Many applications need to be ported from architecture to architecture for a number of reasons, including cost-effectiveness, a widespread user community, and corporate decisions. Coding in standard FORTRAN 77 does not solve this portability problem; standard Fortran 90 compilers are just becoming available. The Fortran 90 subsets vary from vendor to vendor, and the Fortran 90 standard is not sufficient for high-performance computing. A fairly clear vision of scalable or portable programming practices for Fortran exists but is not yet universally agreed upon. However, it is clear that the FORTRAN programmer who is not moving toward Fortran 90 will soon be out-of-date.

We have detailed the load-balancing problem resulting from computing heat-transfer coefficients in TRAC, including the additional complexity arising from the irregular communication patterns between the heat structure and fluid nodes and have shown how the WHERE construct provides a clean software presentation of the computational requirements. We have implied that the applications programmer who wants to optimize an application containing local data-dependent computations can best use her or his time by writing code that separates decision logic from computation, uses the WHERE construct, and contains numerous highly modular scalar functions. By the time one is finished, the compiler writers will have at least caught up. Finally, we have recommended that programmers wait for the advent of automated load balancing or use alternative algorithms rather than attempting to port data-dependent computations on irregular grids to architectures using a data-parallel or work-sharing parallel programming model.⁵

REFERENCES

1. J. H. Mahaffy, D. R. Liles, and S. B. Woodruff, "Current Algorithms Used in Reactor Safety Codes and the Impact of Future Computer Development on These

Algorithms," in *Proc. Specialist's Meeting on Small Break LOCA Analysis in LWR's, Pisa, Italy, II* (1985).

2. M. I. Shiguro, H. Harada, N. Shinozawa, and K. Naraoka, "Vectorization of LWR Transient Analysis Code RELAP5/MOD1 and Its Effect," Japan Atomic Energy Research Institute report JAERI-M 85-040 (1985).
3. A. N. Choudhary and R. Ponnusamy, "Parallel Implementation and Evaluation of a Motion Estimation System Algorithm Using Several Data Decomposition Strategies," *Journal of Parallel and Distributed Computing* **14**, 50-65 (1992).
4. G. C. Fox et al., *Solving Problems on Concurrent Processors*, Vol 1 (Prentice Hall, New York, 1988).
5. R. V. Hanxleden and L. R. Scott, "Load Balancing on Message Passing Architectures," *Journal of Parallel and Distributed Computing* **13**, 312-324 (1991).
6. J. M. Rutledge, D. R. Jones, W. H. Chen, and E Y. Chung, "Application Issues for Large Scale Reservoir Simulation on Massively Parallel Computers," *Proceedings of Supercomputing '91* (1991), p. 541.
7. J. L. Larsen, "Collecting and Interpreting hpm Performance Data on the CRAY Y-MP system," *NCSA data link* 5(6), 14-24 (1991).
8. C. Koelbel, Announcement and Call for Participation, Inaugural Meeting of the High Performance Fortran Forum (HPFF), January 27-28, 1992, Houston.
9. D. M. Pase, "MPP Fortran Programming Model," Draft 1.2 (unpublished).
10. "CM Fortran Reference Manual," Thinking Machines Corporation, Cambridge, Massachusetts, 1991.
11. J. Thibault and B. P. A. Grandjean, "A neural network methodology for heat transfer data analysis," *Int. J. Heat Mass Transfer* **34**(8), 2063-2070 (1991).
12. R. D. Jones et al., "Nonlinear Adaptive Networks: A Little Theory, A Few Applications," Conference on Cognitive Modeling in System Control, Santa Fe. June 10-14, 1990.
13. CM-5 (Thinking Machine Corporation, Cambridge, 1991)

```

if (MASKNB) then
  if (MASK1) then

    T1 = F1 (...)
    T2 = F2 (...)

    if (MASK11) then
      T3 = F3 (...)
    else
      T3 = F4 (...)
    endif

    if (MASK12) then
      T4 = F5 (...)
      HV = T4*MAX (T1, T2, T3)
      HL = (1. - T4)*F6 (...)
    else
      HV = MAX (T1, T2, T3)
      HL = 0.
    endif

  else

    HV = 0.
    HL = F6 (...)

  endif
endif

```

Fig. 1. Computational flow for computing nonboiling heat-transfer coefficients.

TASK 1.

```
WHERE (MASKNB.and.MASK1)
  T1 = F1 (...)
  T2 = F2 (...)
END WHERE
WHERE (MASKNB.and.MASK1.and.MASK11) T3 = F3 (...)
WHERE (MASKNB.and.MASK1.and..not.MASK11) T3 = F4 (...)

WHERE (MASKNB.and.MASK1) HV = MAX (T1, T2, T3)
```

TASK 2.

```
WHERE (MASKNB.and..not.MASK1) HV = 0.
WHERE (MASKNB.and.((MASK1.and.MASK12).or..not.MASK1)) HL = F6(...)
WHERE (MASKNB.and.MASK1.and..not.MASK12) HL = 0.
```

TASK 3.

```
WHERE (MASKNB.and.MASK1.and.MASK12)
  T4 = F5 (...)
  HV = T4*HV
  HL = (1. - T4)*HL
END WHERE
```

Fig. 2. Fortran 90 representation of computational flow in Fig. 1.

```

integer n, list(1:n), i, j
logical mask(1:n)
real x(1:n), y(1:n)
external f

j = 0
do i = 1, n
  if (mask(i)) then
    j = j + 1
    list(j) = i
  endif
enddo

!The following coding is equivalent to
!  WHERE (MASK) Y = F (X)

do i = 1, j
  y(list(i)) = f (x(list(i)))
enddo

```

Fig. 3. Illustration of compressed index implementation to accomplish a WHERE construct.