PPPL--2427

DE87 010635

VECTORIZING AND MACROTASKING MONTE CARLO NEUTRAL PARTICLE ALGORITHMS

D.B. Heifetz
Princeton Plasma Physics Laboratory
Princeton, N.J. 08544

Abstract

Monte Carlo algorithms for computing neutral particle transport in plasmas have been vectorized and macrotasked. The techniques used are directly applicable to Monte Carlo calculations of neutron and photon transport, and Monte Carlo integration schemes in general. A highly vectorized code was achieved by calculating test flight trajectories in loops over arrays of flight data, isolating the conditional branches to as few a number of loops as possible. A number of solutions are discussed to the problem of gaps appearing in the arrays due to completed flights, which impede vectorization. A simple and effective implementation of macrotasking is achieved by dividing the calculation of the test flight profile among several processors. A tree of random numbers is used to ensure reproducible results. The additional memory required for each task may preclude using a large number of tasks. In future machines, the limit of macrotasking may be possible, with each test flight, and split test flight, being a separate task.

1. INTRODUCTION

We describe here programming techniques used to vectorize and macrotask Monte Carlo algorithms for simulating the transport of mass, momentum, and energy by atoms and neutral molecules in plasmas and high vacuums. The problems which arise in vectorizing and macrotasking these algorithms occur also in Monte Carlo simulation of the transport of neutral particles such as neutrons and photons, as well as Monte Carlo integration computations in general. The techniques discussed here should apply to many of these other calculations.

The specific application treated here is computing neutral particle transport in plasmas magnetically contained in controlled fusion experiments. The goal of such experiments is to create a hot enough (100 million degrees C) deuterium, D, and tritium, T, plasma, to produce the reaction

$$D + T \rightarrow {}^4He\ (3.5\ MeV) + n\ (14.3\ MeV),$$

which creates helium, ^{4}He, and energetic neutrons, n. The plasma must also be dense enough (2×10^{14} ions/cm^3), so that enough heat is produced to sustain the reaction in a burn state. Producing substantial power at this density and temperature requires further that the heat be retained in the plasma for an average of at least one second.

The most successful approach so far to achieving these parameters is the tokamak, where a toroidally shaped plasma is confined in a field formed by a combination of a field induced by a current running axially through the plasma, and a field generated externally by coils encircling the plasma. The Tokamak Fusion Test Reactor at the Princeton Plasma Physics Laboratory, for example, has recently achieved temperatures over 150 million degrees C.

2. THE PHYSICS OF NEUTRAL PARTICLE TRANSPORT IN PLASMAS

Plasma confinement in controlled fusion experiments is not perfect, and a steady current of ions and electrons leaves the core plasma. Various means are used to control the escaping particles and their energy. In tokamaks, for example, graphite limiters are commonly used to protect the device structure from damage due to escaping ions and electrons.

Ions striking the limiter recombine with electrons to form atoms and molecules, which then re-enter the plasma. Atoms and molecules are also introduced through wall desorption processes, gas fueling, and by plasma heating with neutral particle beams.

Neutral particles play an important role in controlled fusion experiments [1]:

- They are unaffected by the magnetic field which confines the plasma, and transport mass, momentum, and energy across magnetic flux surfaces, significantly influencing the confinement of the plasma.
- Neutral particles carry a significant portion of the plasma's energy to the device structure. They also sputter material off the walls, contaminating the plasma and reducing power production efficiency.

- A number of important experimental measurements of the plasma temperature and overall particle recycling are made by observing atoms as they emit radiation or leave the core plasma.

A schematic of the primary physical processes determining neutral particle kinetics, including both neutral/plasma and neutral/wall interactions, is given in Fig. 1. Once formed, hydrogenic atoms interact with the plasma in a number of ways. They can be ionized by plasma electrons or ions. This is analogous to the process of absorption in neutron and photon kinetics. An atom can also transfer its one electron to an ion by passing near enough to the ion to form momentarily a quasi-molecule, which immediately breaks apart due to the high energy of the two nuclei, with the electron now binding to the original ion. The effect is to change the velocity of the atom, analogous to the process of scattering in neutron kinetics. Hydrogen molecules are also formed, and they dissociate under electron impact, to produce both ionic and atomic hydrogen.

3. THE BOLTZMANN EQUATION

A. The Differential Form

Neutral particle kinetics is described by the Boltzmann equation

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = C(f),$$

where $f = f(\mathbf{x}, \mathbf{v}, t)$ is the distribution of neutral particle population as a function of position, $\mathbf{x}$, velocity, $\mathbf{v}$, and time t, and the collision operator, C, describes the interaction of the neutral particles with the plasma and the tokamak structure.

For the low neutral populations typically found in a tokamak, there are no nonlinear effects, such as neutrals interacting with other neutrals, so that C, and the entire Boltzmann equation, is linear in f. It can also be assumed that $\partial f/\partial t = 0$, since the time scale for change in the plasma conditions, and hence C, is much longer than the lifetime of any neutral particle.

B. The Integral Form

The Boltzmann equation can also be put into an integral form, with a kernel, $C(\mathbf{x}, \mathbf{y})$, which is the probability that a neutral from point $\mathbf{y}$ will charge transfer at point $\mathbf{x}$. In a discretized solution of the Boltzmann equation, for a circularly cylindrical geometry, integration using C becomes multiplication by a matrix $\mathbf{C} = (c_{ij})$, where c_{ij} now is the probability that an atom starting in zone j charge-transfers in zone i [3].

Given a source $\mathbf{S}$ of atoms, the distribution of the first generation of charge-transfers is given by $\mathbf{C} \cdot \mathbf{S}$, the next by $\mathbf{C} \cdot \mathbf{C} \cdot \mathbf{S} = \mathbf{C}^2 \cdot \mathbf{S}$, and so on. The total charge-transfer rate is the sum over all the generations

$$(\mathbf{I} + \mathbf{C} + \mathbf{C}^2 + \ldots) \cdot \mathbf{S} = (\mathbf{I} - \mathbf{C})^{-1} \cdot \mathbf{S}. \qquad (1)$$

4. MONTE CARLO SOLUTIONS TO THE BOLTZMANN EQUATION

Many partial analytical and numerical solutions exist to the Boltzmann equation for neutral particle transport in plasmas [3]. These solutions, in one and two spatial dimensions, contain realistic descriptions of physical processes, they show explicit parametric dependence, so that extrapolations and interpolations can be made directly from them, and they are computationally fast. However many questions require three-dimensional geometric calculations, of detailed, experimentally measured distributions describing physical phenomena. Two strengths of Monte Carlo methods are that using them (1) fully three-dimensional geometries can be modeled with very general algorithms, and (2) detailed physical models can be easily introduced into the simulation. Thus the Monte Carlo approach is a good choice for solving the three-dimensional Boltzmann equation.

5. VECTORIZING MONTE CARLO ALGORITHMS

A. Two Basic Difficulties

Monte Carlo neutral particle calculations consist of a series of mutually independent logically complex calculations of test flight trajectories (see Appendix). The basic approach to exploiting vectorizing hardware is to compute these trajectories in loops over groups of flights, keeping in arrays information about the flights such as their positions and velocities. This is possible because of the independence of the trajectory calculations.

There are two obstacles in vectorizing the loops of test flight calculations:

1. Monte Carlo algorithms contain many conditional branchings, which inhibit easy vectorization.

2. As test flights finish, holes are left in the flight information arrays which must be "filled" to achieve vectorizable loops.

The first difficulty is addressed by isolating conditional statements to as few a number of loops as possible. Numerically expensive calculations, in particular, should be collected into loops free of "IF" statements, which will vectorize. This approach has been implemented in a calculation of the Green's function described in the next section.

Accounting for the gaps left by completed test flights can be done in a number of ways: (1) The arrays can be repacked, (2) array indices of active flights can be used together with hardware or software gather/scatter algorithms, (3) intrinsic functions such as `CVMGM` can be used, or (4) gaps can simply be filled in with new test flights. These methods are discussed in Section 5.B.3.

B. Example: The Green's Function Solution

1. Implementing Vectorization

The Green's function solution to the Boltzmann equation described above has been implemented in a highly vectorized program. The main part of the calculation is computing the matrix **C**. LU decomposition can then be used to determine the solution $(\mathbf{I}-\mathbf{C})^{-1}\cdot\mathbf{S}$ to Eq. 1, for a given source **S**. To compute **C**, test flights are launched in batches in each zone j, and are followed only until their first collisions with the plasma. If a collision is a charge-transfer, then the test flight's weight is added to the total, c_{ij}, in the zone i where

the flight collided. The Monte Carlo tracking section is thus relatively short; the infinite sum in Eq. (1) replaces tracking through subsequent generations of charge transferring atoms.

A flow chart describing the tracking algorithm is shown in Fig. 2. The computationally expensive operations, involving trigonometric, exponential, and logarithm functions, are contained in vectorized loops. Two loops are not vectorized, the repacking loop discussed below, and a loop which calculates neutral particle reflection. This last loop involves sampling from multi-dimensional distributions.

2. Performance

The speedup, σ, from vectorization can be estimated using a version of Amdahl's relation:

$$\sigma = \frac{1}{(1-\alpha) + \alpha/\nu}.$$

Here α is the fraction of the calculation done in vectorized loops, and ν is the average speedup of those loops over their scalar versions.

The individual vectorized loops in the Green's function calculation showed speedups of from 4 to 11. The speedup averaged over all the vectorized loops for a typical calculation on a CRAY-XMP/22 using 20 cylindrical zones was about $\nu = 5.8$. Since approximately 78% of the computation of C took place in these loops, the overall speedup during this calculation was $\sigma \approx 2.85$. The fraction $\alpha = 0.78$ could be increased, for example, by using a simpler reflection model. Amdahl's relation shows that α is already large enough so that a small increase will give a large increase in the speedup.

3. Accounting for Gaps

As the calculation progresses, test flights finish, and must be effectively deleted from the arrays containing test flight information. In the present implementation of the Green's function calculation, the flight information arrays are repacked "back to front" as gaps appear. A search for gaps is made beginning at the head of the array. When one is found, it is filled with data on the last active flight in the loop. This minimizes the amount of shuffling required to repack; however, it is not effectively vectorizable.

Another approach is to pack the active test flights into a full array using loops such as:

```
      DO 50 I=1,K
        X(I)=X(JPA(I))
        ...
50    CONTINUE
```

Here `JPA` is an integer array containing the indices of the `K` flights which are still active. This array is monotonically increasing, and this loop can be vectorized on a machine with hardware gather/scatter such as the CRAY-XMP/4 or CRAY-2.

The `CVMGM` function can also be used, together with a flight status array, to keep track of active flights. If `JPA(I) = 1` when the `I`-th flight is active, and `JPA(I) = 0` when not, then the entire computation can be done with loops like

```
      DO 100 I=1,NOFLTS
        EO(I)=CVMGM(0.5*RMASS(I)*VEL(I)**2,0.0,JPA(I))
        ...
100   CONTINUE
```

A fourth approach is to fill in spent flights with new ones. This is actually not very useful since, as we next discuss, it is best to track all the test flights entirely in one loop, assuming that storage allows.

4. Optimal Loop Size

One subtlety is that the number of active test flights eventually falls so low that the vectorized loops lose their speed advantage over their scalar versions. To minimize the number of such occurrences, the test flight loops should be as large as possible. Loops of $8-10 \times 64 = 512-640$ test flights are typically flown in each zone to calculate the Green's matrix **C**, as determined by the requirement that the standard errors in the coefficients of **C** be below a given minimum.

6. MACROTASKING MONTE CARLO ALGORITHMS

A. A "Natural" Approach

A growing number of Monte Carlo algorithms, most more complex than this Green's function calculation, have been vectorized. Many of these programs were originally scalar. A common experience has been that vectorizing these existing codes required extensive rewriting [4]. Vectorization as implemented in general purpose computers does not seem to be a "natural" hardware for Monte Carlo algorithms. It would be better to make parallel calculations on the scale size of subroutines, that is, macrotasking with different tasks tracking independent test flights.

B. Example: The Green's Function Solution

Asynchronous macrotasking, such as implemented on the CRAY-XMP series and the CRAY-2 [5], can be applied to Monte Carlo algorithms in a variety of ways. For example, the columns of the Green's matrix **C** are independently computed. Their calculation could thus be distributed over a number of processors. Each independent task would consist of a copy of the vectorized algorithm in Fig. 2. A high degree of processor overlap should be possible in theory. This approach may not be practical, however, because the granularity of the tasks is too small. They each would each finish in only a few milliseconds of CPU time. The usefulness of macrotasking with such small tasks depends intimately on the system's scheduling algorithm.

C. Example: The DEGAS Code

1. Implementing Macrotasking

Tasks on a larger scale have been implemented in the DEGAS neutral particle transport code [6]. The DEGAS code is a three-dimensional calculation of transport in plasmas and high vacuums, where the neutral population is so small that the problem is linear. The emphasis in writing the program has been on including the most realistic descriptions

of the relevant physical phenomena, with less attention given to programming efficiency. It is mainly a scalar program, which would require a major effort to substantially vectorize. With the exceptions described below, however, macrotasking was implemented in DEGAS with the addition or modification of only about 20 FORTRAN lines.

A flow chart of the overall macrotasking scheme is shown in Fig. 3. To compute a profile consisting of NOFLTS test flights, NOTASKS tasks are used, each consisting of a partial profile of NOFLTS/NOTASKS flights:

```
      SUBROUTINE PROFILE(KSEED,LOCKVAR)
      TASK COMMON/COMTRACK/
            ...
      DO 10 J=1,NOFLTS/NOTASKS
10      CALL TRACK
```

The arguments to PROFILE are the random number seed KSEED (described below) and a lock identifier LOCKVAR (also described below). The TASK COMMON/COMTRACK/ contains the arrays of test flight tracking information, and their Monte Carlo scorings.

The tasks are started using TSKSTART [7]:

```
      PARAMETER (NPTASKS=4)
      DIMENSION ITCA(3,NPTASKS),NSEED(NPTASKS)
      EXTERNAL PROFILE
            ...
C              Assign lock identifier
      CALL LOCKASGN(LOCKVAR,ISTAT)
C                  Start tasks
      DO 10 I=1,NOTASKS
        ITCA(1,I)=3
10      CALL TSKSTART(ITCA(1,I),PROFILE,NSEED(I),LOCKVAR)
```

The array ITCA is the task control array. Note that the random number seed NSEED, is passed to the task through an array, so that each task will use a different address to retrieve that task's random number seed.

The routine LOCKASGN is used to assign a lock identification for the one lock used in the algorithm. It is used in the tasks on the routine DEPOSIT, where the results of the tasks, stored separately in TASK COMMON/COMTRACK/, are added to the COMMON totals over all tasks:

```
      SUBROUTINE PROFILE(KSEED,LOCKVAR)
            ...
C           Add task scorings to total
      CALL LOCKON(LOCKVAR)
      CALL DEPOSIT
      CALL LOCKOFF(LOCKVAR)
```

This lock is necessary to avoid two tasks simultaneously modifying the same COMMON variable.

The macrotasking finishes with a loop which waits for all the tasks to finish, before ending the calculation:

```
      DO 30 I=1,NOTASKS
30      CALL TSKWAIT(ITCA(1,I))
```

2. Performance

The macrotasking DEGAS program has been extensively run on the four processor CRAY-2 of the National Magnetic Fusion Energy Computer Center, in Livermore, California. One measure of macrotasking is processor overlap, defined as the sum of the individual processor's CPU times divided by the time during which at least one processor was in use. The maximum theoretical processor overlap during the flight tracking profile of the DEGAS calculation, where over 95% of its computing time is spent, approaches the number of processors used. This overlap is actually less since tasks vary in length, due to the statistical nature of the calculation, and because of intrinsic limitations in the system, such as memory bank delays.

The actual processor overlap has only been measured in a multiuser environment, in the Cray Time-sharing System (CTSS). The CTSS scheduling algorithm favors macrotasking programs, sending related tasks to the top of the processor queues when any one task gains a processor. Using CTSS and the CRAY-2, a range of overlaps from 1.40 to 3.75 has been observed during the flight tracking section of the calculation. The tasks were from 30–120 seconds long, and the executable code was 9–12 Mwords long. The maximum overlap of 3.75 was achieved during a test on the CRAY-2 with no other programs running, which compares favorably with other macrotasking Monte Carlo programs [8].

The processor overlap in DEGAS varies with the number of simultaneously running CTSS jobs, and the sizes of those jobs. The lowest overlaps (less than 2) have been observed when DEGAS was competing with a number of other large programs, which were probably not themselves macrotasking. This behavior depends critically on system-defined scheduling parameters.

Another somewhat weaker, but more useful, measure of macrotasking is the ratio of the wall-clock time for the calculation while unitasking over the macrotasking wall-clock time. While this ratio has not been systematically measured, it is generally bigger than the processor overlap. For example, the same problem was run twice in similar system conditions, using one and four tasks. The processor overlap for the macrotasking job was 2.0, but the unitasking job took over 3.5 times as long to finish, by wall-clock time, as the macrotasking job.

The enhanced wall-clock speedup occurs because, even though the tasks may not receive processors simultaneously, they may receive processors before the lead task exits its processor (Fig. 4), and this is what determines the wall-clock speedup. The wall-clock performance may also not degrade as quickly as processor overlap with the competition for processors. Again, all this depends heavily on the system's program scheduling algorithm.

Note that just as it is best to make the size of a vectored loop to be a multiple of the length of the vector registers, it is best, when using a small number of large tasks, to make

the number of tasks a multiple of the number of processors. It is inefficient, for example, to run a calculation of five tasks of equal length using four processors. Four will run in unison, with the fifth waiting for a free processor. After the four finish, three processors will be unused while the fifth runs.

3. Some Bookkeeping Chores

While the modification of the structure of the DEGAS program required to introduce macrotasking was minimal, a few time-consuming bookkeeping chores were necessary to complete the implementation. The first was a careful inspection of the FORTRAN coding, to verify that the task routines were truly re-entrant. Then the `TASK COMMON/COMTRACK/` was checked to be sure that it contained all the critical flight tracking variables. Finally, it was necessary to add to `TASK COMMON/COMTRACK/` array variables to store the test flight scorings of a task. It is these arrays that are added in `SUBROUTINE DEPOSIT` to `COMMON` arrays containing scorings totalled over all tasks.

While this work required extensive program editing, no essential changes in the algorithm were needed. This is in contrast to the modifications typically needed to vectorize an existing scalar Monte Carlo algorithm, where the extensive changes necessary in the logical structure can introduce programming errors.

4. A Storage Problem

Each `TASK COMMON` generated by a task adds 10–15% to the size of the executable code. The reason is that DEGAS collects scorings in the `TASK COMMON` for all the elements in its geometric grid, because it is necessary to know the entire neutral particle population $f(\mathbf{x},\mathbf{v})$. This is unlike a typical calculation of neutron transport, where only a small number of quantities are calculated in a single computation.

Creating four tasks may increase the code size by as much as 60%. While tolerable now, this growth may become prohibitively expensive if larger numbers of tasks were run concurrently (see the next section). An alternative, less expensive storage approach, using `TASK COMMON` "buffers," is currently being developed.

5. Future Approaches

The computational time for a single test flight can be as short as a few milliseconds, which may be too short to justify a single task (cf. the section on macrotasking the Green's function solution). In some applications, however, it can take as long as a few seconds to track one flight. In such cases it is reasonable to make a task consisting of a single flight. New tasks could also be spawned at the splitting of a test flight, and ended if Russian Roulette is fatal. Figure 5 shows how this macrotasking approach might be implemented in DEGAS. Such a scheme should be very attractive on future machines with large numbers of processors.

D. The Problem of Reproducibility

The CFT random number generator produces a single sequence of pseudo-random numbers. Separate tasks sample asynchronously from this sequence, and different calculations from the same input will agree only statistically. A systematic approach to achieving exact reproducibility between calculations is to use trees of pseudo-random numbers, called

Lehmer trees [10]. Two linear congruential pseudo-random sequences are used to generate two successors to an element X in the tree:

$$L(X) = a_L X + c_L \text{ (Mod m) and } R(X) = a_R X + c_R \text{ (Mod m)},$$

where the constants a_L, a_R, c_L, c_R, and m are chosen to ensure that the tree's branches are reasonably independent.

Figure 6 illustrates how Lehmer trees can be used in DEGAS' macrotasking schemes. In the present scheme, with a fixed number of tasks, the individual task random number seeds, NSEED, are just an original seed, and three consecutive left successors (Fig. 6a). In the scheme using separate tasks for individual flights and split flights, left successors are used as initial seeds for the branches of the Lehmer tree defined by right successors. The Lehmer trees thus correspond to trees of test flights and their split descendents (Fig. 6b).

7. CONCLUSION

Monte Carlo algorithms for computing neutral particle transport in plasmas can be effectively vectorized and macrotasked. The techniques to do so described here are directly applicable to many Monte Carlo calculations of the transport of neutrons and photons, and Monte Carlo integrations in general.

A highly vectorized code was achieved by calculating test flight trajectories in loops over arrays of flight data, with the conditional branches isolated to as few a number of loops as possible. A number of solutions are possible to the problem of gaps appearing in the arrays due to completed flights, which otherwise impedes vectorization.

Experience has shown that it is easier to introduce macrotasking into an existing Monte Carlo program than extensive vectorization. A simple implementation of macrotasking in neutral transport calculations is achieved by dividing the calculation of the test flight profile among several processors. In future machines, the limit of macrotasking may be possible, with each test flight, and split test flight, being a separate task. A Lehmer tree of random numbers can be used to ensure reproducible results.

Acknowledgments

The author would like to thank A. Mirin, E. Horowitz, D. Anderson, K. Fong, and H. Bruijnes of the NMFECC for many educational discussions. This work was supported by the U.S. Department of Energy Contract No. DE-AC02-76-CHO-3073.

APPENDIX: THE MONTE CARLO METHOD

Monte Carlo methods approximate continuous distributions by sampling them using sequences of random numbers [11]. Integrating multidimensional distributions then reduces to tallying the samplings. The level of confidence in the results of Monte Carlo calculations is roughly proportional to the square root of the number of samplings taken. Thus, to decrease the uncertainty by factor of α requires an increase in computational effort of the order of α^2. This is the incentive for developing new approaches to Monte Carlo calculation utilizing vectorization and microtasking.

A. Monte Carlo Calculation of Neutral Particle Transport

The heart of a Monte Carlo neutral particle transport calculation is a profile consisting of a relatively small number of test flights, each one representing a group of actual particles. For example a profile of 10^4 test flights might be used to simulate the trajectories of 10^{22} actual neutral particles. The tracking of test flights is done using random numbers to sample from distributions describing the processes affecting the trajectories of actual particles. Quantities, such as the neutral particle distribution, $f(\mathbf{x}, \mathbf{v})$, are then determined from the information collected from all of the test trajectories, and levels of confidence in these results are computed.

B. Tracking Test Flights

The track of a test flight is determined by its collisions with the plasma and walls. If ξ is the probability that a particle travels from the point $\mathbf{x}$ to the point $\mathbf{x} + \Delta\mathbf{x}$ without a collision with the plasma, that is, without ionizing, charge transferring, or dissociating, then

$$\xi(\mathbf{x} + \Delta\mathbf{x}) = \xi(\mathbf{x}) - \xi(\mathbf{x})\Delta\mathbf{x}/\lambda(\mathbf{x}),$$

where $\lambda(\mathbf{x})$ is the mean-free path length to the next collision. The solution for ξ in terms of λ is

$$\xi(\mathbf{x}) = \exp\left(-\frac{\int_{\mathbf{x}_o}^{\mathbf{x}} ds}{\lambda(s)}\right).$$

This distribution is sampled to determine the points of collisions between the test flight and the plasma.

A standard sampling method is illustrated in Fig. 7. Assuming that the test flight is at the point $\mathbf{x}_o$ and has a velocity $\mathbf{v}$, the problem is to solve $-\ln\xi = \int_{\mathbf{x}_o}^{\mathbf{x}} ds/\lambda(s)$, given a choice of a random number ξ, for the collision point $\mathbf{x}$. The integrals to the points of intersection $\mathbf{x}_i$ of the flight's trajectory and the computational grid, $\int_{\mathbf{x}_o}^{\mathbf{x}_i} ds/\lambda(s)$, are easily computed because λ is constant within each grid region. At some point, $\mathbf{x}_n$, $\int_{\mathbf{x}_o}^{\mathbf{x}_n} ds/\lambda(s) > -\ln\xi$. The flight is then linearly backtracked within a single region to the point $\mathbf{x}$ of collision.

C. Test Flight Weighting and Scoring

A test flight begins representing $\omega = S/N$ actual particles, where S is the neutral particle source rate, and N is the number of test flights. The value ω is called the flight's weight. At each collision with the plasma the probability ρ that a charge-transfer occurred is computed, and the flight is given the new weight of $\rho\omega$. The reduction in weight, $(1-\rho)\omega$, represents the loss due to ionization. Eventually the flight weight becomes insignificantly low. Then a test is made at each collision whether a charge-transfer occurs, with probability ρ, in which case the flight continues with unchanged weight, or an ionization occurs, ending the flight.

The rate at charge-transfer occurs in a region is given by $\sum \rho\omega$, where the sum is over all collisions in that region. The ionization rate, momentum and energy transfer rates, wall fluxes and other quantities, are computed in a similar way.

D. Splitting and Russian Roulette

Some regions of interest may be remote from the neutral source, or small in volume, so that few scorings occur there. Test flights may be focused on such regions by ranking regions by their importance. When a test flight changes regions from one of lesser interest to a more important one, it is split into a number, m, of flights, each with weight $1/m$ times that of the original. Multiplying the number of flights will, in general, multiply the number of collisions in that more important region. Russian Roulette can be combined with splitting to keep the additional flights from adding unnecessarily to the number of scorings in the less important region. In Russian Roulette those flights traveling into less important regions are killed off with a probability of $1 - (1/m)$, and the weights of the survivors multiplied by m.

References

[1] D. Heifetz, "Neutral Particle Transport," in *Physics of Plasma-Wall Interactions in Controlled Fusion*, D. Post and R. Behrisch, Eds., Plenum, New York (1986).

[2] M. Tendler and D. Heifetz, Fusion Technol. **11** No. 2 (1987) 289-310.

[3] K. Audenaerde et al., J. Comput. Phys. **34** (1980) 268.

[4] Y. Tokunaga et al., Comput. Phys. Commun. **38** (1985) 15.

[5] L. Nelson and P. Rigsbee, "Multitasking at CRAY," Cray Channels, Summer, 1985.

[6] D. Heifetz, et al., J. Comput. Phys. **46** (1982) 309.

[7] *Multitasking User Guide*, Cray Research Publication SN-0222.

[8] Y. Chauvet, Comput. Phys. Commun. **37** (1985) 281.

[9] NMFECC Computational Physics Group, "Parallel Computing on the NMFECC System II," Bull. Am. Phys. Soc., **31**, No. 9 (Oct., 1986) 1606.

[10] P. Fredrickson et al., Parallel Computing **1** (1984) 175.

[11] J. M. Hammersley and D.C. Handscomb, *Monte Carlo Methods*, Methuen, London, 1964, and L. L. Carter and E. D. Cashwell, *Particle Transport Simulation with the Monte Carlo Method, TID-26607*, U. S. Energy Research and Development [Agency Report (1975).

Figure Captions

Fig. 1. The physical processes affecting neutral particle kinetics in a tokamak [2].

Fig. 2. A flow chart [1] of the loops in the vectorized calculation of the Green's function matrix **C** used in the solution Eq. 1 to the Boltzmann equation. `NOFLTS` is the number of test flights, `NCOLL` and `NESCAP` are the number of test flights which have ionized and escaped respectively, and `IPART` is the number of active flights. The sizes of the loops are indicated in parentheses, and are underlined if the loop is vectorized.

Fig. 3. Flow chart of the macrotasking algorithm used in DEGAS. `NOTASKS` is the number of tasks, `ITCA` is the task identifier array, `NSEED` is the arrays of random number seeds, one for each task, `NOFLTS` is the total number of test flights, and `LOCKVAR` is the lock identifier. Each task begins at `SUBROUTINE PROFILE`, which contains a loop tracking `NOFLTS/NOTASKS` test flights. The scores from these flights, kept in `TASK COMMON/COMTRACK/`, are added to overall totals in `SUBROUTINE DEPOSIT`. A lock is used before calling `DEPOSIT` to avoid more than one task simultaneously adding to the totals. The program finishes when a loop with calls to `TSKWAIT` completes.

Fig. 4. Two possible sequencings of four tasks from a single program, both showing good performance in terms of reducing the wall-clock computation time, but with (a) good and (b) bad processor overlaps [9]. Even though the tasks in example (b) do not receive processors simultaneously, all of them may get a processor before the lead task exits its processor. Thus both cases will finish in roughly the same amount of wall-clock time.

Fig. 5. Flow chart of a macrotasking algorithm where each test flight is a task. A loop spawns `NOFLTS` tasks, each starting with the routine `TRACK`. If a test flight splits, the descendent is also tracked using a task beginning at `TRACK`. The random number seeds, `NSEED(J)` and `LSEED`, are picked from a Lehmer tree.

Fig. 6. (a) A Lehmer tree of pseudo-random numbers for a simple macrotasking scheme of four tasks with fixed, predetermined lengths. Four left successors serve as initial seeds for random number trees for each task, defined by right successors. (b) A Lehmer tree for an open-ended scheme using separate tasks for individual flights and split flights. The tree of flights and splittings corresponds to the Lehmer tree as shown.

Fig. 7 Sampling $\xi(\mathbf{x})$ for the points $\mathbf{x}$ of collision between the test particle flights and the plasma. Assuming that the test flight is at the point $\mathbf{x}_o$ and has a velocity $\mathbf{v}$, the problem is to solve $-\ln\xi = \int_{\mathbf{x}_o}^{\mathbf{x}} ds/\lambda(s)$, given a choice of a random number ξ, for the collision point $\mathbf{x}$. The integrals to the points of intersection $\mathbf{x}_i$ of the flight's trajectory and the computational grid, $\int_{\mathbf{x}_o}^{\mathbf{x}_i} ds/\lambda(s)$, are easily computed because λ is constant within each grid region. At some point, $\mathbf{x}_n$, $\int_{\mathbf{x}_o}^{\mathbf{x}_n} ds/\lambda(s) > -\ln\xi$. The flight is then linearly backtracked within a single region to the point $\mathbf{x}$ of collision.

#86P0357

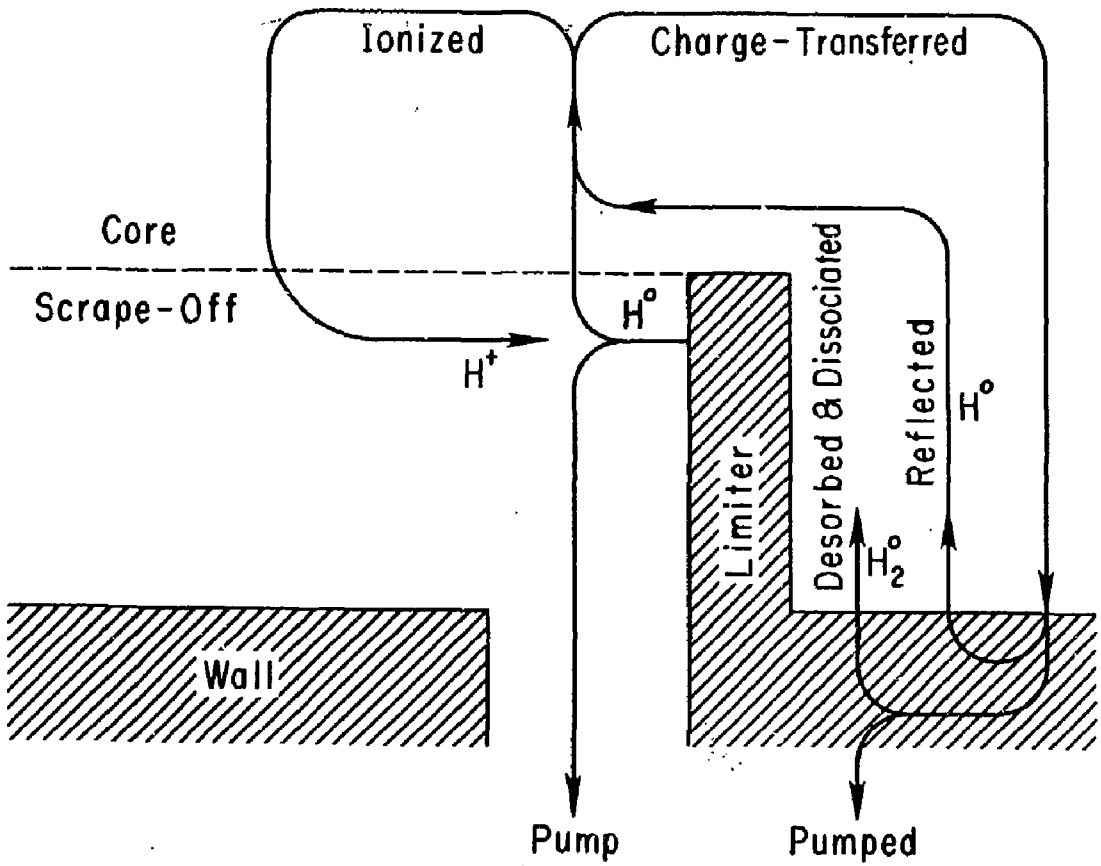

Fig. 1

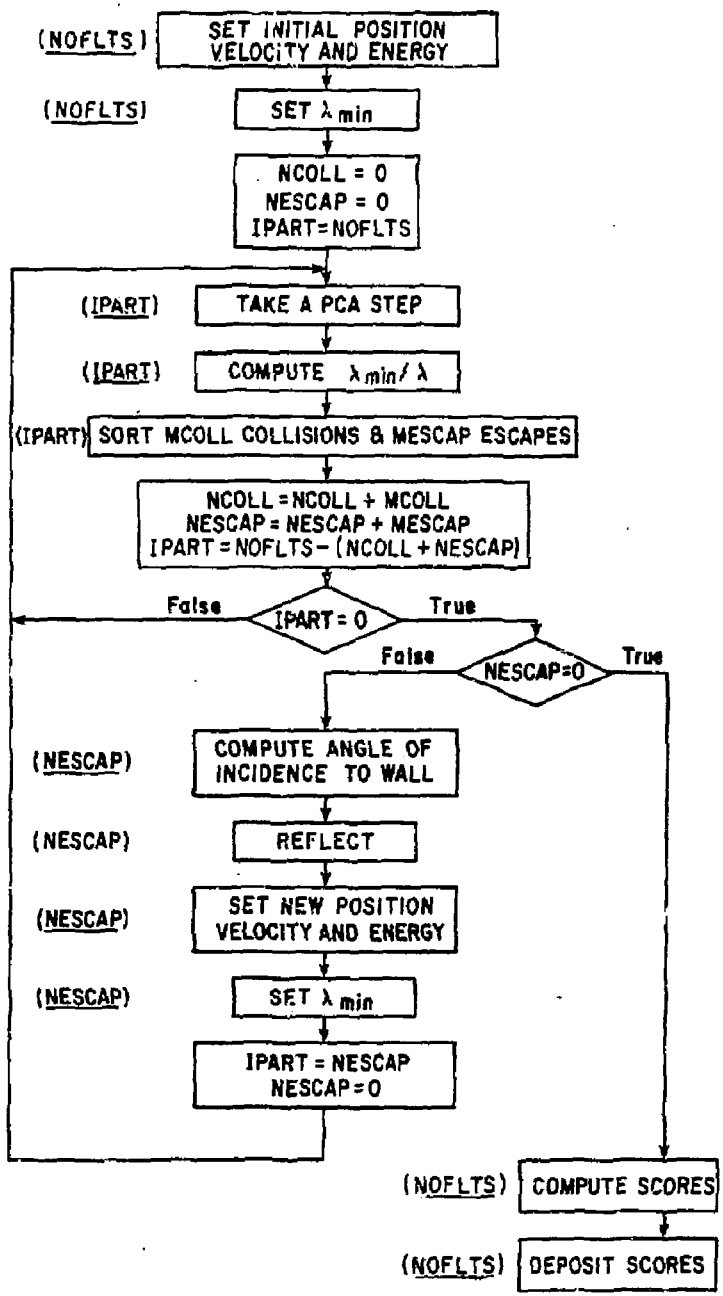
(NOFLTS)
SET INITIAL POSITION VELOCITY AND ENERGY
(NOFLTS)
SET λ min
NCOLL = 0
NESCAP = 0
IPART=NOFLTS
(IPART)
TAKE A PCA STEP
(IPART)
COMPUTE λ min/ λ
(IPART)
SORT MCOLL COLLISIONS & MESCAP ESCAPES
NCOLL = NCOLL + MCOLL
NESCAP = NESCAP + MESCAP
IPART = NOFLTS − (NCOLL + NESCAP)
False
IPART = 0
True
False
NESCAP=0
True
(NESCAP)
COMPUTE ANGLE OF INCIDENCE TO WALL
(NESCAP)
REFLECT
(NESCAP)
SET NEW POSITION VELOCITY AND ENERGY
(NESCAP)
SET λ min
IPART = NESCAP
NESCAP=0
(NOFLTS)
COMPUTE SCORES
(NOFLTS)
DEPOSIT SCORES

Fig. 2

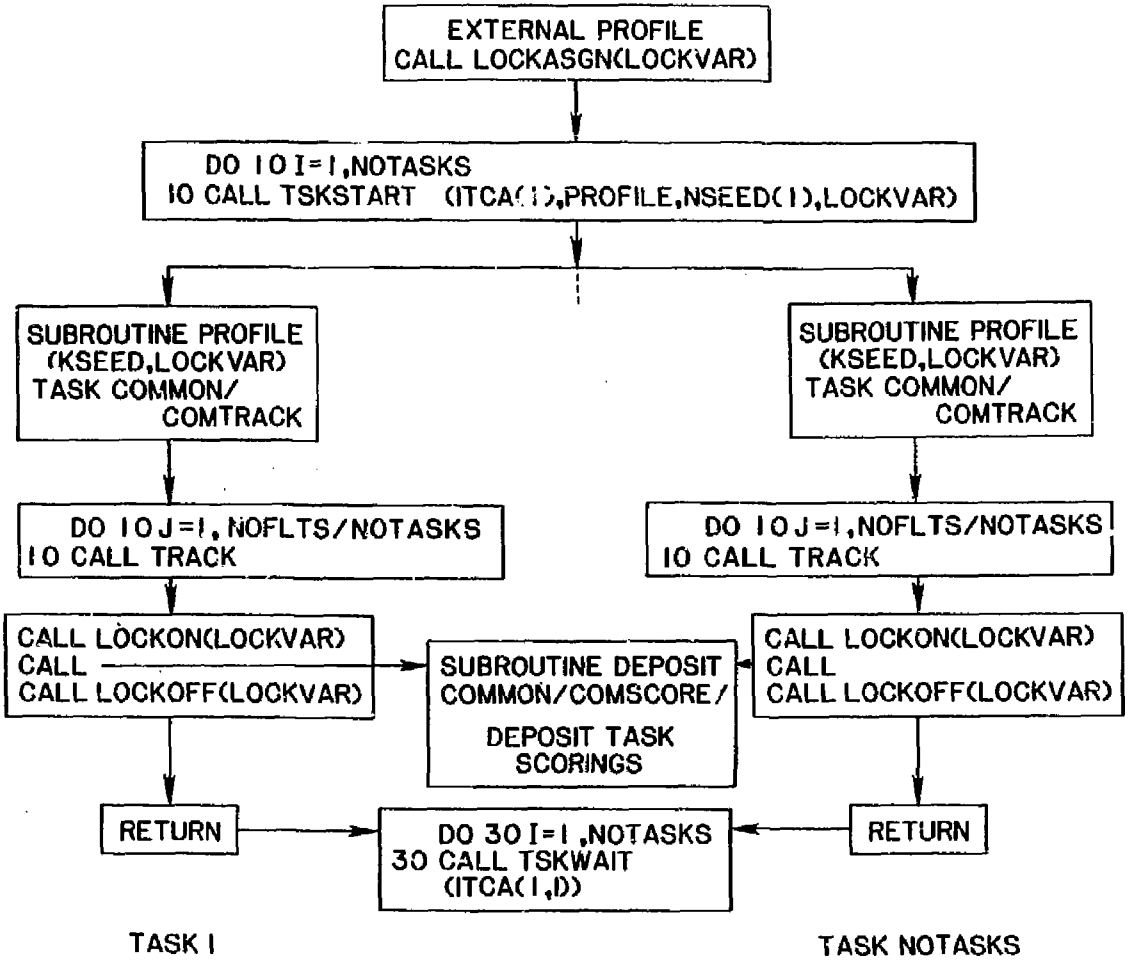
EXTERNAL PROFILE
CALL LOCKASGN(LOCKVAR)
DO 10 I=1,NOTASKS
10 CALL TSKSTART (ITCA(I),PROFILE,NSEED(I),LOCKVAR)
SUBROUTINE PROFILE
(KSEED,LOCKVAR)
TASK COMMON/
COMTRACK
SUBROUTINE PROFILE
(KSEED,LOCKVAR)
TASK COMMON/
COMTRACK
DO 10 J=1,NOFLTS/NOTASKS
10 CALL TRACK
DO 10 J=1,NOFLTS/NOTASKS
10 CALL TRACK
CALL LOCKON(LOCKVAR)
CALL
CALL LOCKOFF(LOCKVAR)
SUBROUTINE DEPOSIT
COMMON/COMSCORE/
DEPOSIT TASK
SCORINGS
CALL LOCKON(LOCKVAR)
CALL
CALL LOCKOFF(LOCKVAR)
RETURN
DO 30 I=1,NOTASKS
30 CALL TSKWAIT
(ITCA(I,D))
RETURN
TASK 1
TASK NOTASKS

Fig. 3

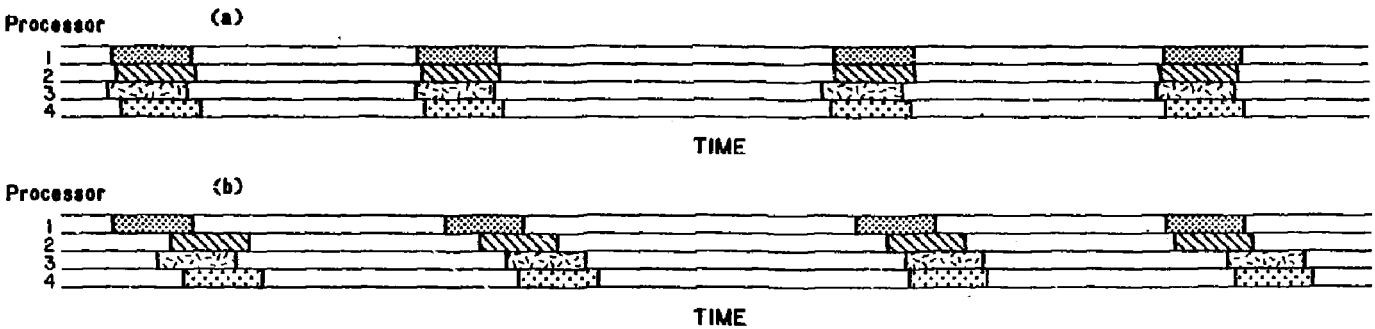

Fig. 4

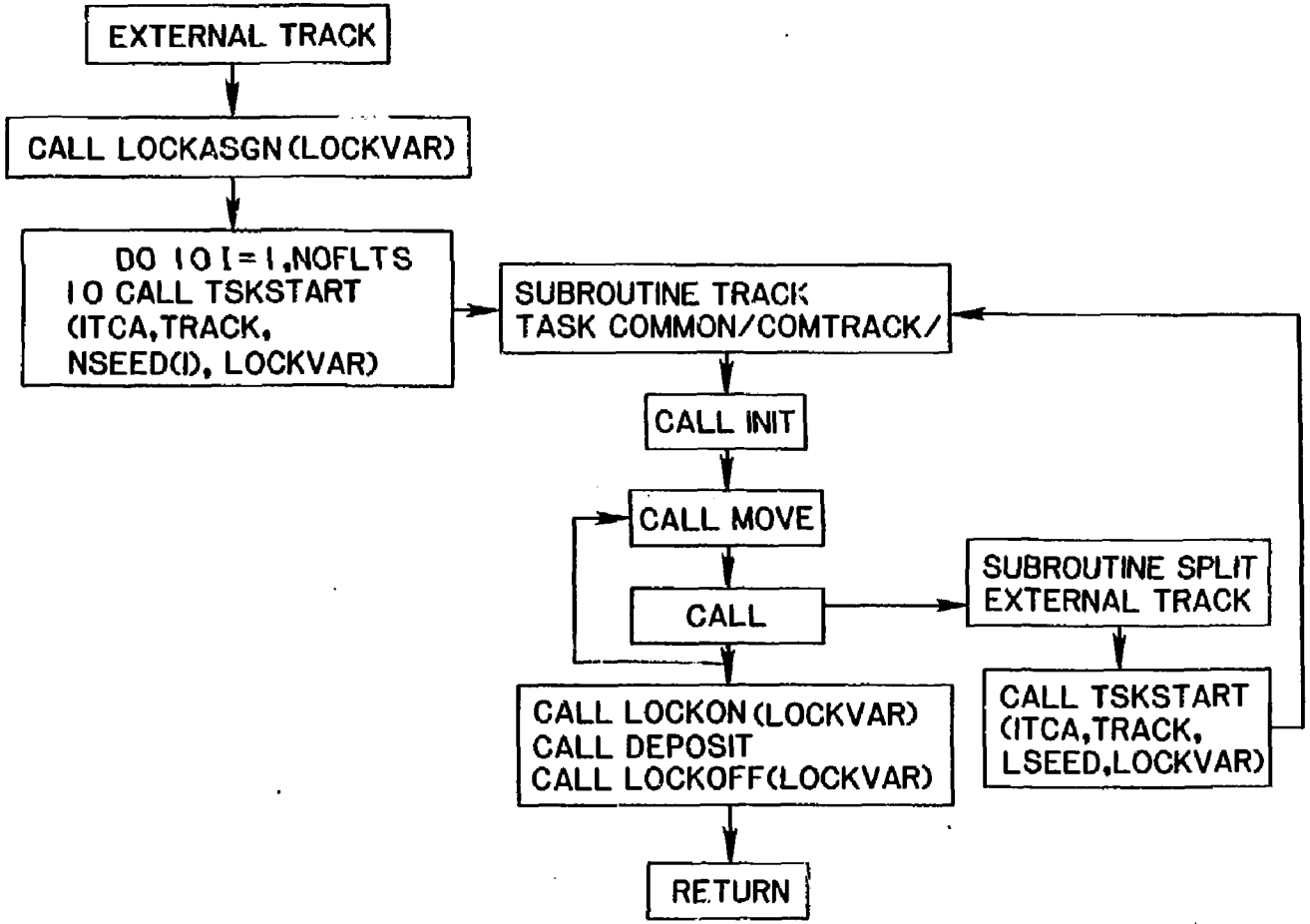
EXTERNAL TRACK
CALL LOCKASGN(LOCKVAR)
DO 10 I=1,NOFLTS
10 CALL TSKSTART
(ITCA,TRACK,
NSEED(I), LOCKVAR)
SUBROUTINE TRACK
TASK COMMON/COMTRACK/
CALL INIT
CALL MOVE
CALL
SUBROUTINE SPLIT
EXTERNAL TRACK
CALL TSKSTART
(ITCA,TRACK,
LSEED,LOCKVAR)
CALL LOCKON(LOCKVAR)
CALL DEPOSIT
CALL LOCKOFF(LOCKVAR)
RETURN

Fig. 5

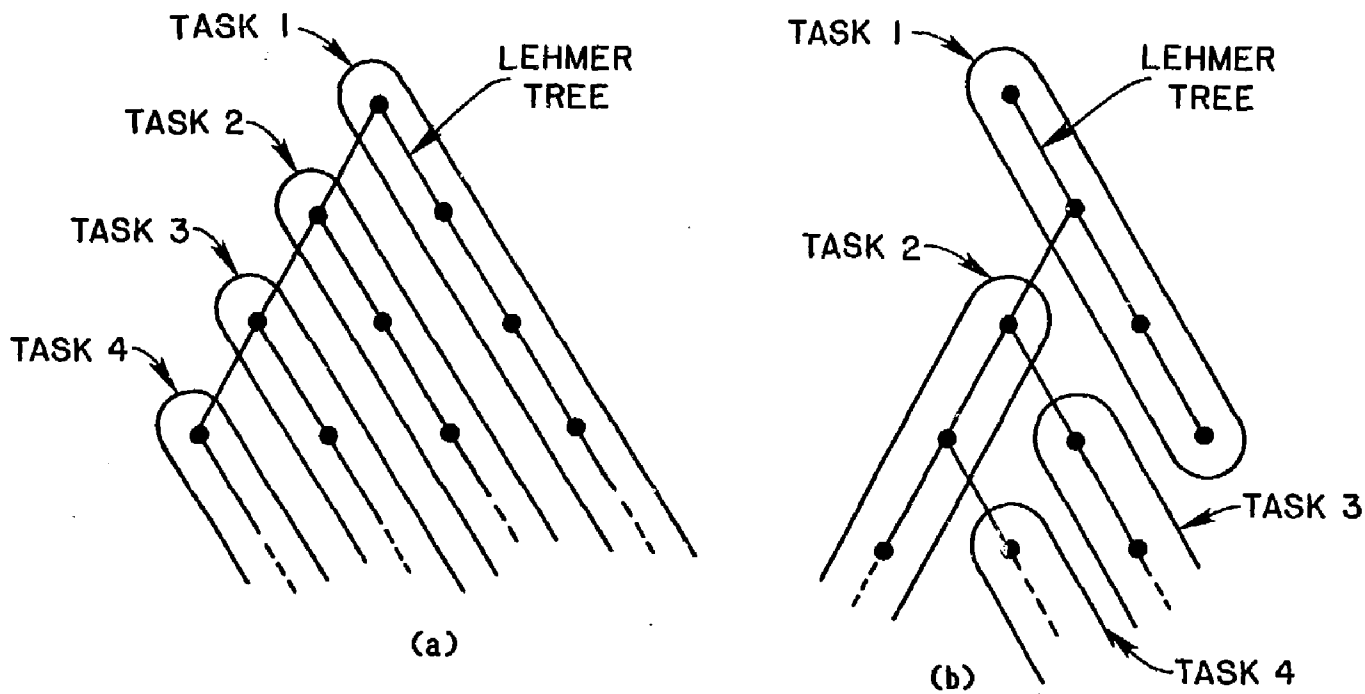
TASK 1
LEHMER TREE
TASK 2
TASK 3
TASK 4
(a)
TASK 1
LEHMER TREE
TASK 2
TASK 3
TASK 4
(b)

Fig. 6

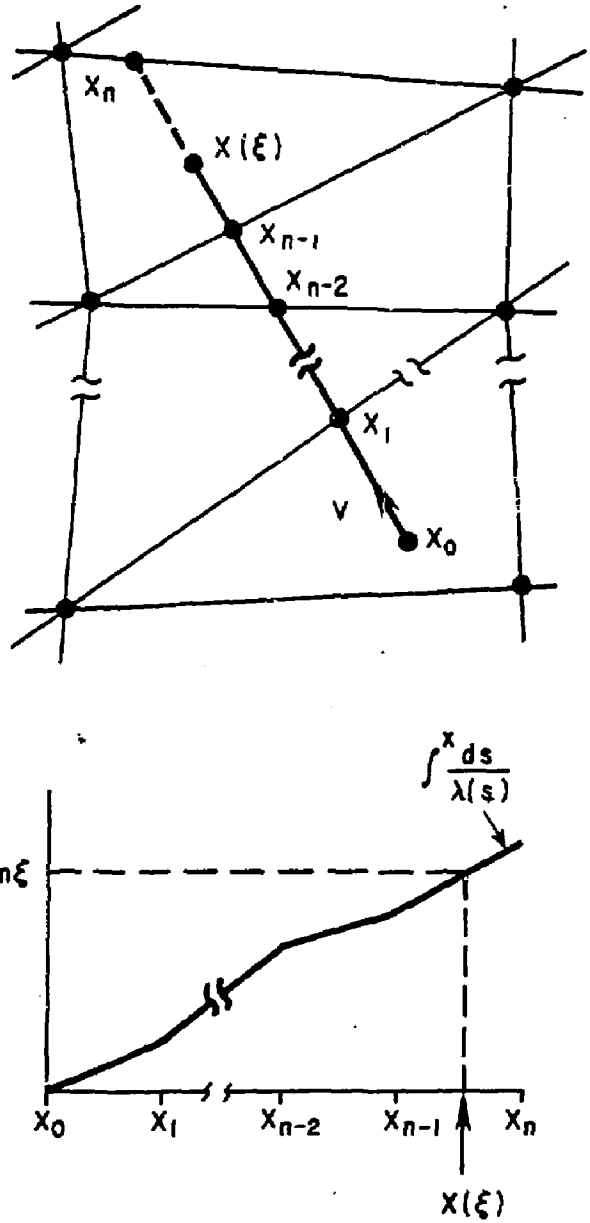
X_n
$X(\xi)$
X_{n-1}
X_{n-2}
X_1
V
X_0
$\int_{X_0}^{X} \frac{ds}{\lambda(s)}$
$-\ln\xi$
X_0
X_1
X_{n-2}
X_{n-1}
X_n
$X(\xi)$

Fig. 7

EXTERNAL DISTRIBUTION IN ADDITION TO UC-20

Dr. Frank J. Paoloni, Univ of Wollongong, AUSTRALIA
Prof. M.H. Brennan, Univ Sydney, AUSTRALIA
Plasma Research Lab., Australian Nat. Univ., AUSTRALIA
Prof. I.R. Jones, Flinders Univ., AUSTRALIA
Prof. F. Cap, Inst Theo Phys, AUSTRIA
Prof. M. Heindler, Institut fur Theoretische Physik, AUSTRIA
M. Goossens, Astronomisch Instituut, BELGIUM
Ecole Royale Militaire, Lab de Phys Plasmas, BELGIUM
Com. of European, Dg XII Fusion Prog, BELGIUM
Prof. R. Boucique, Laboratorium voor Natuurkunde, BELGIUM
Dr. P.H. Sakanaka, Univ Estadual, BRAZIL
Instituto De Pesquisas Espaciais-INPE, BRAZIL
Library, Atomic Energy of Canada Limited, CANADA
Dr. M.P. Bachynski, MPB Technologies, Inc., CANADA
Dr. H.M. Skarsgard, Univ of Saskatchewan, CANADA
Dr. H. Barnard, University of British Columbia, CANADA
Prof. J. Teichmann, Univ. of Montreal, CANADA
Prof. S.R. Sreenivasan, University of Calgary, CANADA
Prof. Tudor W. Johnston, INRS-Energie, CANADA
Dr. G.R. James, Univ. of Alberta, CANADA
Dr. Peter Lukac, Komenskeho Univ, CZECHOSLOVAKIA
The Librarian, Culham Laboratory, ENGLAND
Mrs. S.A. Hutchinson, JET Library, ENGLAND
C. Mouttet, Lab. de Physique des Milieux Ionises, FRANCE
J. Radet, CEN/CADARACHE - Bat 506, FRANCE
Dr. Tom Mual, Academy Bibliographic, HONG KONG
Preprint Library, Cent Res Inst Phys, HUNGARY
Dr. B. Dasgupta, Saha Inst, INDIA
Dr. R.K. Chhajlani, Vikram Univ, INDIA
Dr. P. Kaw, Institute for Plasma Research, INDIA
Dr. Phillip Rosenau, Israel Inst Tech, ISRAEL
Prof. S. Cuperman, Tel Aviv University, ISRAEL
Librarian, Int'l Ctr Theo Phys, ITALY
Prof. G. Rostagni, Univ Di Padova, ITALY
Miss Clelia De Palo, Assoc EURATOM-ENEA, ITALY
Biblioteca, del CNR EURATOM, ITALY
Dr. H. Yamato, Toshiba Res & Dev, JAPAN
Prof. I. Kawakami, Atomic Energy Res. Institute, JAPAN
Prof. Kyoji Nishikawa, Univ of Hiroshima, JAPAN
Direc. Dept. Lg. Tokamak Res. JAERI, JAPAN
Prof. Satoshi Itoh, Kyushu University, JAPAN
Research Info Center, Nagoya University, JAPAN
Prof. S. Tanaka, Kyoto University, JAPAN
Library, Kyoto University, JAPAN
Prof. Nobuyuki Inoue, University of Tokyo, JAPAN
S. Mori, JAERI, JAPAN
M.H. Kim, Korea Advanced Energy Research Institute, KOREA
Prof. D.I. Choi, Adv. Inst Sci & Tech, KOREA
Prof. B.S. Liley, University of Waikato, NEW ZEALAND
Institute of Plasma Physics, PEOPLE'S REPUBLIC OF CHINA
Librarian, Institute of Phys., PEOPLE'S REPUBLIC OF CHINA
Library, Tsing Hua University, PEOPLE'S REPUBLIC OF CHINA
Z. Li, Southwest Inst. Physics, PEOPLE'S REPUBLIC OF CHINA
Prof. J.A.C. Cabral, Inst Superior Tecn, PORTUGAL
Dr. Octavian Petrus, AL I CUZA University, ROMANIA
Dr. Johan de Villiers, Plasma Physics, AEC, SO AFRICA
Prof. M.A. Hellberg, University of Natal, SO AFRICA
Fusion Div. Library, JEN, SPAIN
Dr. Lennart Stenflo, University of UMEA, SWEDEN
Library, Royal Inst Tech, SWEDEN
Prof. Hans Wilhelmson, Chalmers Univ Tech, SWEDEN
Centre Phys des Plasmas, Ecole Polytech Fed, SWITZERLAND
Bibliotheek, Fom-Inst Voor Plasma-Fysica, THE NETHERLANDS
Dr. D.D. Ryutov, Siberian Acad Sci, USSR
Dr. G.A. Eliseev, Kurchatov Institute, USSR
Dr. V.A. Glukhikh, Inst Electro-Physical, USSR
Dr. V.T. Tolok, Inst. Phys. Tech, USSR
Dr. L.M. Kovrizhnykh, Institute Gen. Physics, USSR
Prof. T.J.M. Boyd, Univ College N Wales, WALES
Nuclear Res. Establishment, Julich Ltd., W. GERMANY
Bibliothek, Inst. fur Plasmaforschung, W. GERMANY
Dr. K. Schindler, Ruhr Universitat, W. GERMANY
ASDEX Reading Rm, IPP/Max-Planck-Institut fur Plasmaphysik, W. GERMANY
Librarian, Max-Planck Institut, W. GERMANY
Prof. R.K. Janev, Inst Phys, YUGOSLAVIA