Distributed Computing with Personal Computers

Christophe A. Poulain and Bruce A. Finlayson
Dept. of Chemical Engineering, University of Washington, Seattle, WA 98195

Computer programs for blood-tissue exchange (BTEX) modeling allow one to fit experimental data, find parameter estimates, and perform a sensitivity analysis. However, such extensive computations must be rapidly carried out to be practical.

To date, we have focused on the numerical technique used to solve the BTEX model equations since the most efficient algorithm is essential to carry out the calculation quickly (Finlayson, 1992; Poulain and Finlayson, 1993; Poulain et al., 1995). Nevertheless, hardware considerations are important. While supercomputers and mainframes are the fastest, they are very expensive; in recent years attention has focused on harnessing the computing power of idle workstations or personal computers distributed on networks (Dutto et al., 1994; Henriksen and Keunings, 1994; Sawitzki, 1992; Sharma et al., 1991; Fields, 1993). This approach is attractive, because it can provide large computational resources at a minimal cost. The Dept. of Chemical Engineering at the University of Washington has a cluster of networked Power PC Macintosh computers. This resource prompted us to implement a distributed version of a multipathway blood-tissue exchange model. In doing so, our overall objective is to demonstrate that parallel computation is an efficient approach to decrease computational times in blood-tissue exchange modeling, and that it can be done by scientists using standard tools. This approach is also applicable to other problems such as: stochastic (Laso, 1994) and Monte-Carlo simulations in which a model equation containing a random variable is solved repeatedly; the calculation of sensitivity coefficients of a model (Beck and Arnold, 1977); and parameter estimation.

Test Problem

The problem is shown in Figure 1. An organ is represented by a series of blood-tissue exchange units placed in parallel. Each exchange unit represents a differential operator which takes one input and provides one output. The input to the ith capillary is $C^i_{\text{in}}(t)$, the plasma concentration of the substrate of interest at the upstream end of the path, while the output is $C^i_{\text{out}}(t)$, the plasma concentration at the downstream end of the path. The capillary concentration input/output relationship is defined by a set of nonlinear partial differential equations:

$$\begin{cases} \dfrac{\partial C_p}{\partial t} + \dfrac{F^{(i)}L}{V_p}\dfrac{\partial C_p}{\partial x} = -\dfrac{PS_{p\to\text{isf}}}{V_p}C_p + \dfrac{PS_{\text{isf}\to p}}{V_p}C_{\text{isf}} \\ \dfrac{\partial C_{isf}}{\partial t} = +\dfrac{PS_{p\to\text{isf}}}{V_{\text{isf}}}Cp - \dfrac{PS_{\text{isf}\to p}}{V_{\text{isf}}}C_{\text{isf}} \end{cases}$$

with:

$$C_p(x,t_0) = C_{\text{isf}}(x,t_0) = C_0 \quad \text{for } 0 \le x \le L$$
$$C_p(0,t) = C^{(i)}_{\text{in}}(t) \quad \text{for } t_0 < t \le t_f$$

In addition:

$$PS_{p\to\text{isf}} = \frac{V_{\max}}{K_{s_p} + C_p} \text{ and } PS_{\text{isf}\to p} = \frac{V_{\max}}{K_{s_{\text{isf}}} + C_{\text{isf}}}$$

The result is $C^{(i)}_{\text{out}}(t) = C_p(L,t)$. Notation and derivation of the expressions for the permeability-surface area products PS

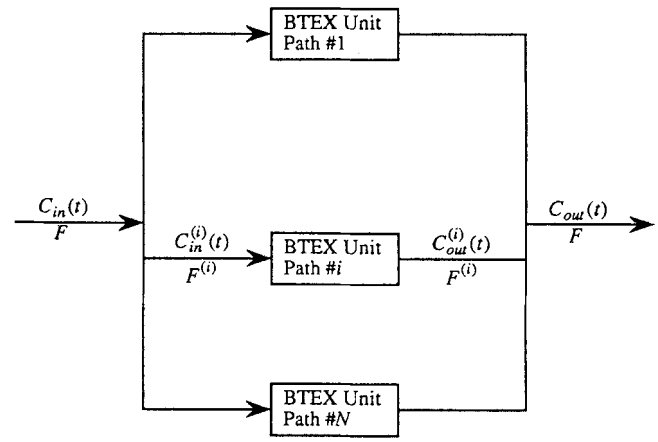

Figure 1. Multipathway blood-tissue exchange model.
An organ is approximated by a set of blood-tissue exchange (BTEX) units, which represent the capillaries, placed in parallel. Arrows indicate the direction of blood flow. F is the total flow to the organ in $\text{mL}\cdot\text{min}^{-1}\cdot\text{g}^{-1}$ and $F^{(i)}$ the flow to the ith pathway.

Correspondence concerning this article should be addressed to B. A. Finlayson.
Current address of C. A. Poulain: Aspen Technology, Inc., Cambridge, MA.

are detailed in Poulain et al. (1995). Each of the BTEX unit shown in Figure 1 is governed by the above problem, with possibly different parameters. The inlet flow is distributed among the pathways by the analyst.

Two flow distributions are used in this work. First, the total flow is equally distributed among all pathways, which are identical. The second calculation allows for differences in flow rates. In all test cases presented in this article, the random choice method is used to solve the nonlinear partial differential equations for a single path as described in Finlayson (1992) and Poulain and Finlayson (1993). In addition, the following parameter values (units are chosen to follow common usage and provide meaningful physiological values): $V_p = 0.05$ mL·g^{-1}, $V_{isf} = 0.15$ mL·g^{-1}, $L = 0.1$ cm, $K_{s_p} = 5$ mol·mL^{-1}, $K_{s_{isf}} = 10$ mol·mL^{-1}, $V_{max} = 3$ mol·min^{-1}·g^{-1}. The initial concentration C_0 is zero. The inlet concentration is a Gaussian curve:

$$C_{in}(t) = \frac{\Omega}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{t-\bar{t}}{\sigma}\right)^2\right]$$

with $\bar{t} = 5$ s, a relative dispersion RD $= \sigma/\bar{t}$ of 0.4, and $\Omega = 50$ mol·mL^{-1}. Each simulation is carried out from time $t_0 = 0$ until $t_f = 3$ min. The solution is recorded every 5 s, but the actual time-step used by the numerical method is variable and is determined according to the solution (Poulain et al., 1995).

Distributed Approach

The implementation uses the client-server model (Sinha, 1992; Sharma et al., 1993). Servers are computer processes which are available for use by the client. Therefore, when the client is asked to solve a problem, it breaks down that problem into individual tasks which are assigned to various servers. The servers work simultaneously and return their results to the client which processes them to obtain the solution to the original problem.

In this project, the computer process on a server can solve the set of partial differential equations for an organ. If there are two servers and N pathways (assuming N even), then each server solves the differential equations $N/2$ times, in succession. If there are N_s servers, then the N pathways are distributed as equitably as possible among the N_s servers.

Unlike sequential programs, distributed programs necessitate the sharing of information between processes running on separate machines and the synchronization of the work of all the processes. The client-server architecture we use is shown in Figure 2. Communication occurs only between client and servers. Information passed between these two processes is either input data (like V_p, V_{isf} or C_0) and output data [like $C_{out}^{(Si)}(t)$] or signals which synchronize the work (for instance, the client might invoke a server by sending it the message 'Do this task', and the server might let the client know that it has completed its task by returning the message 'Task completed'). In the present project, all information is transmitted through files written to or read from a hard disk. Notice that creating a file with a predetermined name and saving it in a specified location is equivalent to sending a message to a known recipient. On the other hand, recognizing the creation

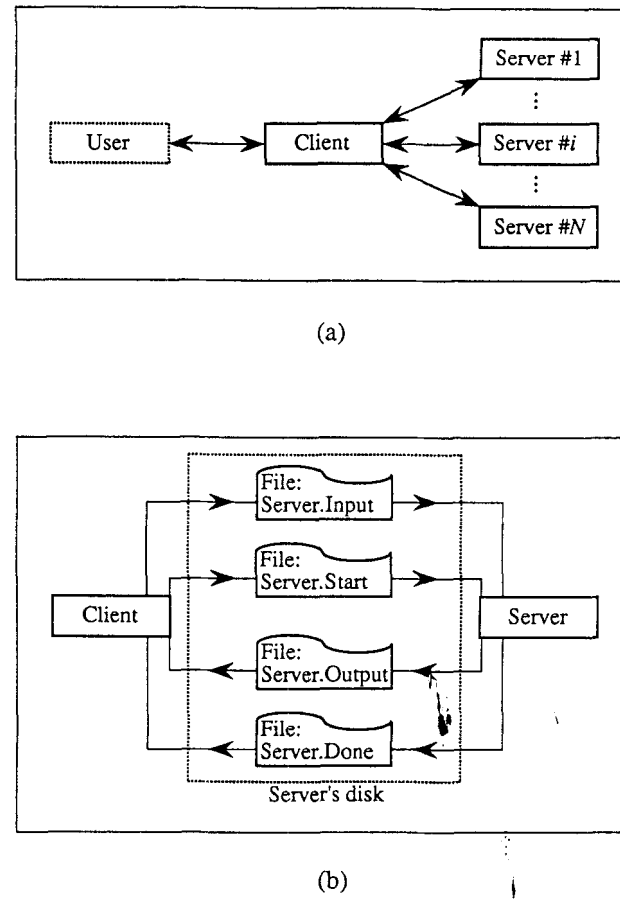

Figure 2. (a) Client-server model; (b) for client-server communication. In both figures, arrows indicate the direction in which information is passed.

of a particular file at a given location is similar to receiving a message. Therefore, the entire interprocess communication is implemented with the standard FORTRAN 77 statements: OPEN, CLOSE, READ, WRITE, and INQUIRE. In order to reduce the cost of communication, binary files are used. In addition, the Macintosh 'File Sharing' feature is activated to allow processes running on a local machine to create or access files on a remote machine which is connected to the local machine. This interprocess communication is simple, and the programming is very straightforward. It can be improved with additional programming work specific to each local computer system.

Figure 2a displays carrying out a distributed multipathway BTEX simulation. At one end stands the user who provides the client with the relevant input simulation data and directs it to start the computation. At this request, the client divides the workload as uniformly as possible among the servers and sends messages to the servers (Figure 2b). First, the client writes the file 'Server.Input' on the hard disk of the machine where the server is running to provide the input data. Then, the client creates the file 'Server.Start'. By detecting the creation of this last file, the server understands that it must start working. The server reads the input data from the file 'Server.Input'; performs the computation; writes its output to a file named 'Server.Output'; creates a file named

'Server.Done'; deletes obsolete files; and goes back to an idle state, waiting for another task to accomplish. The creation of the file 'Server.Done' informs the client that the server is finished with its work. Therefore, the client reads the server's output from the file 'Server.Output'; processes it; and deletes obsolete files. The FORTRAN code associated with these procedures is shown in the Appendix.

Results

The computers used in our experiments are Power PC 7100/66 Macintosh computers equipped with 16MB of RAM and running System 7.1.2. The machines are linked by an Ethernet using Apple's preinstalled Ethertalk card and software. On each machine, there is one server executing. In order to maximize efficiency, there is also a server executing on the machine where the client is running. Apple's 'File Sharing' is activated on each computer in order to permit the client to connect to the servers' machines and have read and write access privileges there. Client and server programs are written in FORTRAN. The compiler used is the Version 1.Beta.2 of FORTRAN for Power Macintosh from Language Systems.

The quality of a distributed algorithm is measured by the speedup and efficiency. The speedup is the ratio of the time required by one machine to solve a problem running the best sequential program to the time required by N_s machines to solve the same problem using the distributed program. Ideally, the speedup would be N_s. However, this limit is never attained because a distributed algorithm requires extra tasks in comparison to a sequential algorithm. The ratio of the actual to the ideal speedup is the efficiency. It measures the cost of distributing the workload among servers.

Figure 3 illustrates the benefits of distributed computing. As the number of servers is increased, the speedup increases

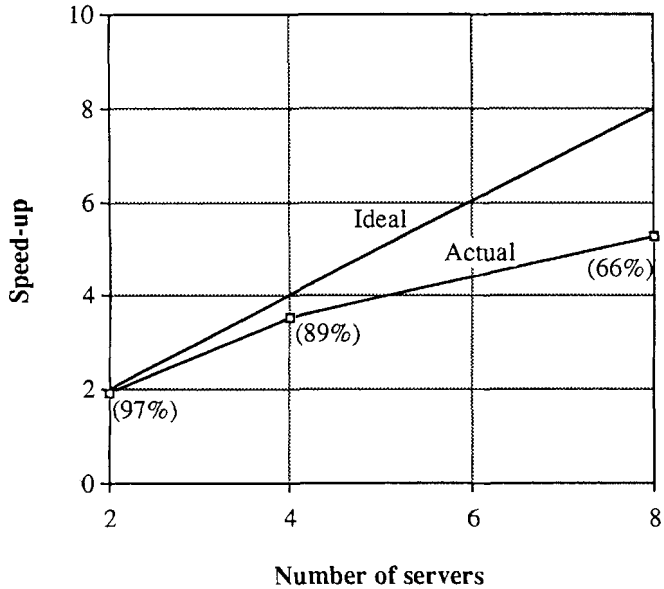

Figure 3. Speedup vs. number of servers.

The distributed multipathway BTEX model is employed to solve a problem including 16 parallel paths. The flow rate in each path is 3 $mL \cdot min^{-1} \cdot g^{-1}$. The numerical method uses 40 finite difference segments. Other parameter values are given at the end of the section entitled "Test problem." Efficiencies are shown in parenthesis.

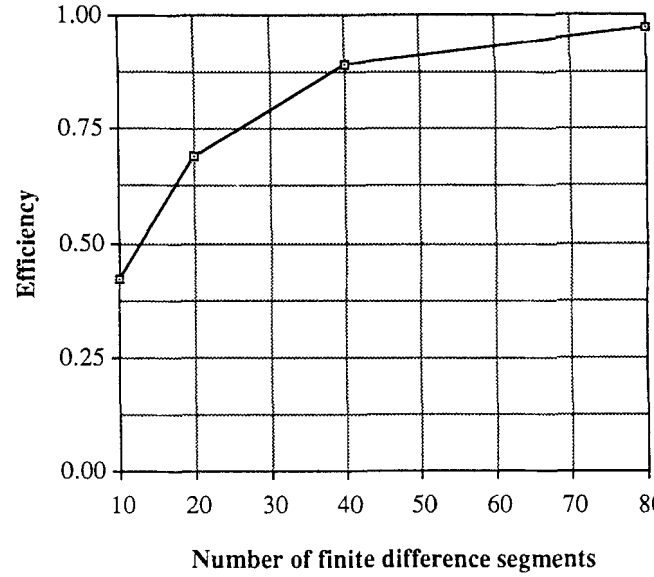

Figure 4. Effect of the computational load on the efficiency.

The model solved has 16 paths with equal flows (3 $mL \cdot min^{-1} \cdot g^{-1}$). The number of servers is four. The number of segments used by the numerical method to solve the BTEX equations for one path is increased from 10 to 80. The servers are assigned identical computational loads (each computes four paths). Changing the number of segments does not modify the time spent by the distributed algorithm on interprocess communication, but it affects the amount of computation performed by each server.

so the computational time decreases. However, the efficiency also diminishes as more servers are added. This decrease is more pronounced when going from four to eight than from two to four servers. This is expected since the client needs to communicate with every server, but it can only talk to one of them at a time. When using the centralized client-server architecture displayed in Figure 2, the point at which the efficiency decreases depends on the partial differential equations being solved and the efficiency is highest for computationally intensive problems generated by large numbers of finite difference grid points or pathways. The scheme employed here is simple. It might be improved by using intermediate layers of processes acting both as servers to the main client and as clients to subservers. In addition, the use of computer specific software, here *AppleEvents* on the Macintosh (Apple Computer, 1993), should improve interprocess communication. However, we have not attempted to compare the importance of these inefficiencies with the cost of communicating on an Ethernet. Furthermore, the above improvements would merely shift the point at which efficiency decreases, and this is problem dependent anyway. Figure 4 shows results for a 16-path model on 4 servers, using different numbers of finite difference segments N_{seg}. As N_{seg} increases, the problem becomes more computationally intensive and the efficiency increases. Figure 5 shows the effect of not being able to split the tasks evenly among processes. With four servers, the efficiency drops when the number of paths is not divisible by 4.

Multipathway BTEX models are needed to represent flow heterogeneities within an organ. Thus, in realistic simulations, each path has a distinct flow rate. The numerical

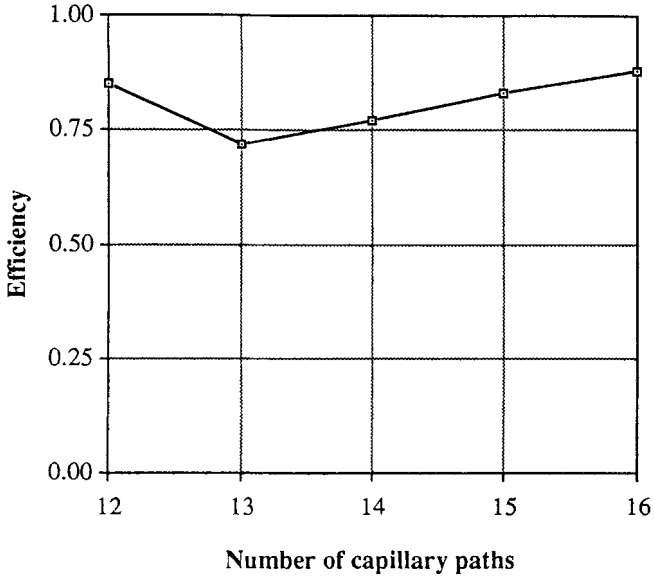

Figure 5. Effect of nonuniform load distribution on the efficiency.

The multipathway model is solved using $N_{seg} = 40$ segments, the same flow in each path (3 mL·min^{-1}·g^{-1}), and $N_s = 4$ servers. The number of paths is increased from 12 to 16.

method we employ requires that the Courant number be less than a threshold. Therefore, pathways with larger flow rates require smaller time-steps, which leads to longer computational times. Even though the model seems to be easily parallelizable, the situation is not so clear-cut because the allowable time-step is highly specific to each pathway. Table 1 shows two sets of results (corresponding to simulation A and B) obtained for a 24-path model with flow rates given in Table 2. The pathways' mean transit times are equally spaced between 0.48 and 12 s. The distributed computation is carried out with two servers which compute 12 paths each. In simulation A, the first server computes the paths with the 12 highest flow rates and the other server computes the paths with the 12 lowest flow rates. In contrast, simulation B attempts to balance the workload between servers by assigning approximately the same total flow rate to each. Table 1 shows that simulation B results in a higher efficiency than simulation A,

Table 1. Load Balancing for a Realistic Simulation

*Simulation	Computational Time (s)		Speedup	Efficiency
	Sequential	Distributed		
A	150	86	1.74	0.87
B	150	79	1.90	0.95

*Two simulations are carried out using $N = 24$ paths, $N_s = 2$ servers, and $N_{seg} = 40$ finite difference segments. In simulation A, the first server computes the paths with flow rate above the median flow rate and the second server computes the remainder. In simulation B, servers are assigned approximately the same total flow rate. Flow rates are detailed in Table 2.

although the disparity is nowhere as serious as the disparity in flow rates. This demonstrates that load-balancing is not crucial for this application.

To evaluate the computational power provided by a network of Power PC Macintosh computers, we use a sequential algorithm to solve a 16 paths BTEX model on various computers. The computational times obtained are: (1) 113 s on a Power PC 7100/66; (2) 88 s on a SunSPARCserver 470; (3) 86 s on an Apollo DN10000; (4) 26 s on a HP9000-720 workstation. This shows that a cluster of 5 personal computers executing a distributed program with an efficiency of 87% are as fast as a recent workstation.

Conclusion

An algorithm has been implemented for the parallel computation of a multipathway BTEX model on a network of Macintosh personal computers. The algorithm is based on the client-server model and proves to be an effective means to reduce computational times. Impressive speedups are obtained when the overall time spent on interprocess communication is kept small in comparison to the time each server devotes to actual model computation.

The distributed programming is very straightforward since it requires only knowledge of FORTRAN and a basic understanding of the Macintosh 'File Sharing' feature. Therefore, the approach seems appropriate for a scientist or engineer whose objective is to get a job done quickly. Nevertheless, there are tools that allow better and more flexible distributed programs.

Table 2. Flow Rates (in mL·min^{-1}·g^{-1}) for Simulation A and B of Table 1

Simulation A				Simulation B			
Server 1		Server 2		Server 1		Server 2	
Path No.	Flow	Path No.	Flow	Path No.	Flow	Path No.	Flow
1	6.25	13	0.46	1	6.25	13	3.06
2	3.06	14	0.43	2	1.21	14	2.02
3	2.02	15	0.40	3	0.75	15	1.51
4	1.51	16	0.38	4	0.55	16	1.01
5	1.21	17	0.35	5	0.50	17	0.86
6	1.01	18	0.33	6	0.46	18	0.67
7	0.86	19	0.32	7	0.38	19	0.60
8	0.75	20	0.30	8	0.35	20	0.43
9	0.67	21	0.29	9	0.29	21	0.40
10	0.60	22	0.27	10	0.27	22	0.33
11	0.55	23	0.26	11	0.26	23	0.32
12	0.50	24	0.25	12	0.25	24	0.30
Total Flow =	18.99	Total Flow =	4.04	Total Flow =	11.52	Total Flow =	11.51

Acknowledgments

Support has been provided by N.I.H. through the Simulation Resource in Mass Transport and Exchange, directed by James B. Bassingthwaighte. We thank the Dept. of Chemical Engineering for making available their undergraduate network of Power PC Macintosh computers.

Literature Cited

Apple Computer, Inc., *Inside Macintosh: Interapplication Communications*, Addison Wesley, Reading, MA (1993).

Bassingthwaighte, J. B., and C. A. Goresky, "Modeling in the Analysis of Solute and Water Exchange in the Microvasculature," *Handbook of Physiology: Section 2, the Cardiovascular System: IV, the Microcirculation*, E. M. Renkin and C. C. Michel, eds., Amer. Physiol. Soc., Bethesda, MD, p. 549 (1984).

Beck, J. V., and K. J. Arnold, *Parameter Estimation in Engineering and Science*, Wiley, New York (1977).

Dutto, L. C., W. G. Habashi, M. P. Robichaud, and M. Fortin, "A Method for Finite Element Parallel Viscous Compressible Flow Calculations," *Int. J. Numer. Fluids*, **19**, 275 (1994).

Fields, S., "Hunting for Wasted Computing Power," Research Sampler, Univ. of Wisconsin-Madison, Office of University-Industry Relations, p. 57 (1993).

Finlayson, B. A., *Numerical Methods for Problems with Moving Fronts*, Ravenna Park Publishing, Seattle, WA (1992).

Henriksen, P., and R. Keunings, "Parallel Computation of the Flow of Integral Viscoelastic Fluids on a Heterogeneous Network of Workstations," *Int. J. Numer. Fluids*, **18**, 1167 (1994).

Laso, M., "Stochastic Dynamic Approach to Transport Phenomena," *AIChE J.*, **40**(8), 1297 (1994).

Poulain, C., and B. A. Finlayson, "A Comparison of Numerical Methods Applied to Nonlinear Adsorption Columns," *Int. J. Numer. Fluids*, **17**, 839 (1993).

Poulain, C., B. A. Finlayson, and J. B. Bassingthwaighte, "A Two-Region, Axially Distributed, Blood-Tissue Exchange Model with Nonlinear Facilitated Transport and Reaction," in preparation (1995).

Sawitzki, G., "The NetWork Project: Asynchronous Distributed Computing on Personal Workstations," *Develop*, **11**, 82 (1992).

Sharma, S. K., J. W. Baugh, Jr., and H. S. Chadha, "A Client-Server Approach for Distributed Finite Element Analysis," *Adv. in Eng. Software*, 69 (1993).

Sinha, A., "Client-Server Computing," *Commun. of the ACM*, **35**(7), 77 (1992).

Appendix

Essential components of a FORTRAN distributed program are presented below. For clarity and compactness, the code shown is not in standard FORTRAN 77. However, many compilers support the extension used, and nonstandard statements or expressions can easily be modified to follow the FORTRAN 77 norm.

Each server is reading messages from and writing messages to the directory where it is executing. Thus, once a server is launched, it waits for a task with a loop like:

```
      DO WHILE (.TRUE.)
        INQUIRE (FILE = 'Server.Start', EXIST = ZStart)
        IF (ZStart) THEN
        LEAVE
          ENDIF
c       Add code here to provide a way to stop the server pro-
c       gram. Let operating system process other tasks on the
c       machine. With Language System FORTRAN/PPC on
c       the Macintosh this is:
        CALL F_DoBackGround
      ENDDO
```

When it starts a task, the server reads the input data (for example an array DatInp of LenInp single precision number), and deletes the files 'Server.Input' and 'Server.Start':

```
      OPEN (10, FILE = 'Server.Input', FORM =
      'Unformatted')
      READ (10) (DatInp (i), i = 1, LenInp)
      CLOSE (10, STATUS = 'Delete')
      OPEN (10, FILE = 'Server.Start', FORM =
      'Unformatted')
      CLOSE (10, STATUS = 'Delete')
```

Then, the server computes the solution (which, for example, is an array DatOut of LenOut single precision number) and returns it to the client:

```
      OPEN (10, FILE = 'Server.Output', FORM =
      'Unformatted')
      WRITE (10) (DatOut (i), i = 1, LenOut)
      CLOSE (10)
      OPEN (10, FILE = 'Server.Done', FORM =
      'Unformatted')
      CLOSE (10)
```

Finally, the server re-enters the loop where it waits for a task.

The client, on the other hand, knows the name of the directory where the kth server is executing. It stores that name in a substring SvrDir (k) (1:LenDir(k)). After dividing the workload, the client provides each server with their input data and gives them the order to work:

```
      DO k = 1, NumServer
        fname = SvrDir (k)(1:LenDir (k))//'/Server.Input'
        OPEN (10, FILE = fname, FORM = 'Unformatted')
        WRITE (10) (DatInp (i,k), i = 1, LenInp)
        CLOSE (10)
        fname = SvrDir (k)(1:LenDir (k))//'/Server.Start'
        OPEN (10, FILE = fname, FORM = 'Unformatted')
        CLOSE (10)
      ENDDO
```

Then, the client waits for each server's results and reads them (and, maybe, process them) as they arrive. ndone is the number of servers which have completed their task. The variable ZFinish (k) is of type logical and is true when the kth server is finished.

```
      ndone = NumServer
      DO k = 1, ndone
        ZFinish (k) = .FALSE.
      ENDDO
      DO WHILE (ndone.GT.0)
        DO k = 1, NumServer
          IF (.NOT.ZFinish (k)) THEN
            fname = SvrDir (k)(1:LenDir (k)) //
            '/Server.Done'
            INQUIRE (FILE = fname, EXIST = ZDone)
            IF (ZDone) THEN
              fname = SvrDir (k)(1:LenDir (k)) //
              '/Server.Output'
              OPEN (10, FILE = fname, FORM =
              'Unformatted')
              READ (10) (DatOut (i,k), i = 1, LenOut)
              CLOSE (10, STATUS = 'Delete')
              fname = SvrDir (k)(1:LenDir(k)) //
              '/Server.Done'
```

```
OPEN (10, FILE = fname, FORM =
'Unformatted')
CLOSE (10, STATUS = 'Delete')

Process results for this server now or do it
later.

ndone = ndone – 1
ZFinish (k) = .TRUE.
      ENDIF
    ENDIF
  ENDDO
  CALL F_DoBackGround
ENDDO
```

After that, the client can deliver the final output to the user.

Manuscript received Nov. 4, 1994, and revision received Mar. 6, 1995.