Parallel Sorting and Data Partitioning by Sampling

by

Jun S. Huang
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.

0046
April, 1983
ABSTRACT

A parallel sorting method which requires data partitioning is presented. The ability to partition the data into equal size ordered subsets is essential in the sorting process. We propose a data partitioning method by sampling. The complexity and the performance of the sorting and partitioning algorithm are analyzed. Storage bounds and the choice of parameters which determine the sampling size are also discussed. The analysis is developed for parallel sorting in local network environment with distributed data sets in secondary storage devices.


General Terms: Algorithms, Theory, Design.

Additional Key Words and Phrases : parallel sorting, data partitioning by sampling, local network, quick sort, negative hypergeometric distribution.
I. Introduction

Sorting is an essential operation in data processing as well as in many scientific researches. The recent advance in circuit technology and computer architecture has prompted several efforts in developing parallel sorting algorithms and parallel sorting architectures. In general, the parallel sorting algorithms depend heavily on the architecture of the sorting machines. Muller and Preparata [6] propose a network of $O(N^2)$ processing elements to sort $N$ numbers in $O(\log N)$ time. Hirschberg [3] uses $N$ processors to sort $N$ data and achieves the same $O(\log N)$ time complexity but with larger space requirement. Whereas Nassimi and Sahni [7] use cube and perfect shuffle array processor with $N^{1+1/k}$ processing elements, $1 \leq k \leq \log N$, which is capable of sorting $N$ data items in $O(k \log N)$ computing time. However, all of these approaches are too limited since in general the number of processing elements (or computers) is limited and should not depend on the size $N$ of the data set; especially when $N$ is large the above methods become unrealizable. Another drawback of these designs is the assumption that all data to be sorted are available simultaneously, i.e., the data accessing and input/output are completely ignored.

A more realistic approach is considered by Winslow and Chow [10] in which parallel sorting is performed by using Parallel Balanced Tree Sort in a conventional bus structured local computer network. The sorting consists of three stages: the distribution
(or partitioning) of the data set into ordered subsets, independent parallel sorting of each subset, and the concatenation of the sorted subsets. The performance of the sorting approach depends on how well the data set can be partitioned equally. It is the emphasis of this paper to develop the data partitioning strategy for parallel sorting and to analyze the complexities of the partitioning process.

Let a large data set of size $N$ be sorted on a multiple processor system with $n$ processors, $n < N$. Chow and Winslow [10] show that to gain a sorting speed up factor of $n$ when $n$ processors are utilized, it is necessary to partition the data set into $n$ equal size components such that all of the data in the $i$\textsuperscript{th} component are less than each data in the $(i + 1)$\textsuperscript{st} component, where $i = 1, 2, \ldots, n - 1$. These $n$ components are then sorted independently and simultaneously by the $n$ processors. Finally, the entire sorted data set is obtained by concatenating the sorted components which requires little computation time. The key point of this sorting method lies in developing an efficient procedure for partitioning the data set. However, in general, we do not know "the best way" to partition the $n$ data into $n$ equal size components good for later sorting. To overcome this difficulty we propose a partitioning procedure by taking random samples for the data set and using the order statistics of this sample to partition the $N$ data. The proper sample size to achieve the high probability of each component having size less than a prespecified limit, is analyzed and computed. The complexities of the sorting and the partitioning procedure are obtained. Another convenient method for the sample size problem is also developed.
2. Data Partitioning and Parallel Sorting

Let the data set to be sorted parallely on \( n \) processors be denoted by \( X \), and the size of \( X \) by \( N \), where \( N > n \). To partition \( X \) we first take a random sample of size \( n^\ell - 1 \) (the choice of \( \ell \) will be discussed later), and order this sample in ascending order to get order statistics:

\[
Y_1 < Y_2 < \ldots < Y_\ell < \ldots < Y_{2\ell} < \ldots < Y_{(n-1)\ell} < \ldots < Y_{n\ell - 1}
\]

Secondly, we use \( n - 1 \) points \( Y_\ell', Y_{2\ell}', \ldots, Y_{(n-1)\ell}' \) as pivot nodes and form a balanced binary tree having these \( n - 1 \) nodes. At the bottom of this tree are \( n \) buckets. Each data is steered to its correct bucket as it descends the tree (see Figure 1). Thus from

![Figure 1: Binary Tree with \( n = 5 \) Buckets.](image-url)
this binary tree we are able to partition $X$ into $n$ components such that all data in the $i^{th}$ component are less than each data in the $i + 1^{st}$ component, $i = 1, 2, \ldots, n - 1$. Let the $i^{th}$ component be denoted by $Q_i$, $i = 1, 2, \ldots, n$. Then

$$Q_1 = \{x : x < Y_1\},$$
$$Q_i = \{x : Y_{(i-1)\ell} < x < Y_{i\ell}\}, \text{ for } 2 \leq i \leq n-1,$$
$$Q_n = \{x : Y_{(n-1)\ell} < x\}.$$

Now we can use the sample sort method proposed by Frazer and McKellar [1] to sort these $n$ $Q_i$'s on $n$ processors simultaneously. To explain this more clearly, we note that there are $\ell - 1$ sample points between $Y_{(i-1)\ell}$ and $Y_{i\ell}$. These $\ell - 1$ sample points are again used as random sample taken from $Q_i$. Thus we can apply Frazer and McKellar's procedure to sort each $Q_i$ on the $i^{th}$ processor. Their procedure is a variation of Quick Sort (see Hoare [2]). After parallel sorting, we can easily insert these $n$ pivot nodes into $Q_i$ and then concatenate all together with very little effort to obtain the full sorted data set $X$. The entire sorting consists of sampling and insertion of pivot points, parallel sorting on each processor, and the final concatenation of the sorted components.
3. Analysis of the Sorting Method

Let $q_i(j)$ be the probability that $y_i = x_j$ where $y_i$ is the $i^{th}$ order statistic of the sample and $x_j$ is the $j^{th}$ elements of the sorted set of $X$. It is easy to see

$$q_i(j) = \frac{\binom{j-1}{i-1}(N-j)(n^k-i-1)}{N(n^k-1)}.$$  

Let $P_i(j)$ be the probability that the number of elements in $Q_i$ is $j$. Then we have

**Lemma 1.**

$$P_i(j) = \binom{N-j-1}{(n-1)k-1} \frac{j}{j-1} \binom{N}{n^k-1}, \text{ for } j \geq k-1.$$  

This probability is independent of $i = 1, 2, \ldots, n$.

**Proof.** For $i=1$, $P_1(j) = q_1(j+1) = \binom{j}{j}(n-j-1)\binom{N}{(n-1)k-1}$.  

For $i=n$, $P_n(j) = q_{n-1}^{(N-j)} = \binom{N-j-1}{(n-1)k-1} \frac{j}{j-1} \binom{N}{n^k-1}.$  

For $2 \leq i \leq n-1$,  

5
\[ P_i(j) = \sum_{t=(i-1)\ell}^{N-(n-1)\ell-j} q_{(i-1)\ell}(t)q_{i\ell}(t+j+1 \mid Y_{(i-1)\ell} = x_t), \]

\[
= \sum_{t=(i-1)\ell}^{N-(n-1)\ell-j} \binom{t-1}{(i-1)\ell-1} \binom{N-t}{(n-1)\ell-1} \binom{j}{(n-1)\ell-1} \binom{N-t-j-1}{(n-1)\ell-1}
\]

\[
= \binom{N-j-1}{(n-1)\ell-1} \binom{j}{(n-1)\ell-1}
\]

where \( q_{i\ell}(t+j+1 \mid Y_{(i-1)\ell} = x_t) \) equals to the probability \( q_{i\ell}(j+1) \) for a sample of size \((n-i+1)\ell - 1\) from a set of size \(N-t_0\).

From this lemma, we get the distribution function \( P_i(j) \), \( j = \ell - 1, \ell, \ldots, N - (n - 1)\ell \). In fact this distribution is called the negative hypergeometric distribution (see Sarndal[8]). The mean of this distribution, or the mean size of \( Q_i \) is

\[ E(j) = \frac{N - n + 1}{n} \]

and the variance of this distribution is

\[ \text{Var}(j) = \frac{(N - n\ell + 1)(n - 1)}{(n\ell + 1)n} \]
Thus an approximate 95\% confidence interval for the size of $Q$ is

$$\frac{N+1}{n} - 1 + 3\sqrt{\frac{(N-n\ell+1)(n-1)}{(n\ell+1) \cdot n}}.$$ 

This holds for all $i$ and also this formula sets an approximate lower limit of the size of core storage of each processor for fast processing without disk I/O delay.

Let $E(C_i)$ be the expected number of comparisons required to sort the sample of size $n\ell - 1$ by using the minimum storage Quicksort, then

$$E(C_i) = \frac{n\ell - 1}{2n\ell} \sum_{i=1}^{n\ell-1} \frac{1}{i+1} - 2(n\ell - 1) \quad (1)$$

Now we can treat $Y_{(i-1)\ell+1} < Y_{(i-1)\ell+2} < \ldots < Y_{i\ell-1}$ as $\ell - 1$ order statistics from a population of size $j$ given that $Q_i$ has size $j$. We can extend the sample sort proposed by Frazer and McKellar to sort $Q_i$. The expected number of comparisons required to sort $Q_i$ given that $Q_i$ has size $j$, $j \geq \ell - 1$ is

$$E[C(Q_i | j)] = E(C_2) + E(C_3)$$

where $C_2$ is the number of comparisons required to insert the sample, and $C_3$ is the number of comparisons to sort the segments of $Q_i$.

Similarly with Frazer and McKellar's analysis, it can be shown that

$$(j-\ell+1)\log_2 \ell \leq E(C_2) \leq (j-\ell+1)(0.0861 + \log_2 \ell),$$
and $E(C_3) = 2(j+1) \sum_{i=1}^{j} \frac{1}{i+1} - 2(j-\ell+1)$.

Thus the expected number of comparisons required to sort $Q_1$ is

$$E[C(Q_1)] = E E[C(Q_1 | j)]$$

and therefore

$$E[C(Q_1)] = E E(C_2) + E E(C_3).$$

After further derivation we obtain

$$\frac{N-n\ell+1}{n} \log_2 \ell \leq E E(C_2) < \frac{N-n\ell+1}{n} \left[ 0.086 + \log_2 \ell \right] \quad (2)$$

and

$$E E(C_3) = \sum_{j=\ell-1}^{N-(n-1)\ell} \frac{(N-j-1) \binom{j}{\ell-1} \left[ 2(j+1) \sum_{i=1}^{j} \frac{1}{i+1} \right]}{\binom{N}{n\ell-1}}$$

and

$$-\sum_{j=\ell-1}^{N-(n-1)\ell} \frac{(N-j-1) \binom{j}{\ell-1}}{\binom{N}{n\ell-1}} \cdot 2(j-\ell+1).$$
To simplify the calculation we need the following genius identity due to Knuth [4].

**LEMMA 2** (Knuth).

\[
\sum_{j=\ell}^{N-a} \binom{N-j-1}{a-1} \binom{j}{\ell-1} \left[ 2(j+1) \sum_{i=\ell}^{j} \frac{1}{i+1} \right] = 2\ell \left( \binom{N+1}{a+\ell} \sum_{a+\ell}^{N} \frac{1}{i+1} \right).
\]

**PROOF.**

\[
\sum_{j=\ell}^{N-a} \binom{N-j-1}{a-1} \binom{j}{\ell-1} \left[ 2(j+1) \sum_{i=\ell}^{j} \frac{1}{i+1} \right]
\]

\[
= 2\ell \sum_{\ell}^{N-a} \binom{N-j-1}{a-1} \binom{j}{\ell} (H_{j+1} - H_{\ell}) \text{ where } H_j = \sum_{1}^{j} \frac{1}{i},
\]

\[
= 2\ell \sum_{0}^{N} \binom{N-j}{a-1} \binom{j}{\ell} (H_{j} - H_{\ell}).
\]

Now \( \left( \frac{k}{a-1} \right) \frac{z^k}{(1-z)^a} \) and

\[
\sum_{0}^{\infty} \left( \frac{j}{\ell} \right) (H_{j} - H_{\ell}) z^j = \frac{z^\ell}{(1-z)^{\ell+1}} \log \left( \frac{1}{1-z} \right).
\]

Multiply these two power series together and look at the coefficient of \( z^N \);

\[
\frac{z^{a+\ell-1}}{(1-z)^{a+\ell+1}} \log \left( \frac{1}{1-z} \right) = \sum_{0}^{\infty} \binom{N+1}{a+\ell} (H_{N+1} - H_{a+\ell}) z^N.
\]

and hence the given sum is \( 2\ell \left( \binom{N+1}{a+\ell} \sum_{a+\ell}^{N} \frac{1}{i+1} \right) \). \( \square \).

By putting \( a = (n - 1)\ell \) in Lemma 2, we have
\[ EE(C_3) = 2 \left( \frac{N \cdot N - N - 1}{n} + \frac{N + 1}{n} \frac{\sum_{1}^{N} \frac{1}{n \cdot i + 1}}{N \cdot N - 1} \right), \]

\[ = 2 \left[ \ell + \frac{N + 1}{n} \left( -1 + \sum_{n \cdot i}^{N} \frac{1}{i + 1} \right) \right]. \quad (3) \]

Since \[ \sum_{n \cdot i}^{N} \frac{1}{i + 1} \leq \log(\frac{N}{(n \cdot i - 1)}) - 1/n \cdot i + 2/(N + 1), \]

\[ EE(C_3) \leq 2 \left[ \ell + \frac{2}{n} + \frac{N + 1}{n} (-1 - \frac{1}{n \cdot i} + \log(\frac{N}{n \cdot i - 1})) \right]. \]

From the above results we have:

**Theorem 1.** The expected number of comparisons (or computing time), \( E(C) \), on processing \( Q_1 \) is given by the sum of Eqs. (1), (2), (3), which is

\[ 2n \cdot i \sum_{1}^{N \cdot i - 1} \frac{1}{i + 1} + \frac{N + 1}{n} (\log_2 \ell - 2 + 2 \sum_{n \cdot i}^{N} \frac{1}{n \cdot i + 1}) - \ell \log_2 \ell + 2(\ell - n \cdot i + 1) \leq E(C) \]

\[ \leq 2n \cdot i \sum_{1}^{N \cdot i - 1} \frac{1}{i + 1} + \frac{N + 1}{n} (\log_2 \ell - 1.9139 + 2 \sum_{n \cdot i}^{N} \frac{1}{n \cdot i + 1}) - \ell \log_2 \ell + 1.9139 \ell \]

\[ + 2(1 - n \cdot i), \]

\[ \leq \frac{N + 1}{n} \left( 2 \log_2 \frac{N}{n \cdot i - 1} + \log_2 \ell - 1.9139 - \frac{2}{n \ell} \right) + 2n \cdot i \log(n \cdot i - 1) - \ell \log_2 \ell \]

\[ + 1.9139 \ell + \frac{4}{n} + 6. \]
Now considering when \( N/n \) and \( \ell \) are large

\[
E(C) = \frac{N+1}{n} \left( 2 \log \frac{N+1}{n\ell} + \log_2 \ell \right) + 2n\ell \log n\ell - \ell \log_2 \ell - 2n\ell,
\]

\[
= \frac{N+1}{n} \left( 2 \log \frac{N+1}{n\ell} \right) \quad \text{when} \quad N > n^2 \ell^2.
\]

**COROLLARY 1.** If \( N > n^2 \ell^2 \) and \( \ell \) is large, then the expected number of comparisons \( E(C) \) on processing \( Q_1 \) is approximately

\[
\frac{N+1}{n} \left( 2 \log \frac{N}{n\ell} \right).
\]

The above procedure analyzes the complexity of parallel sorting of each \( Q_i \). The initial balanced binary tree with \( n - 1 \) nodes and \( n \) buckets on the terminals is used to partition the data set \( X \) and each data is steered to its correct bucket as it descends the tree. The number of operations, say \( CI \), required is:

1. when \( n \) is a number of the form \( 2^k \),
   \[
   CI = (N - n\ell + 1) \log_2 n.
   \]
2. when \( n \) is not of form \( 2^k \), by Lemma 2 of Frazer and McKellar,
   \[
   (N - n\ell + 1) \log_2 n \leq CI \leq (N - n\ell + 1) \left[ 0.0861 + \log_2 n \right] \tag{6}
   \]

In general if the data has to be accessed sequentially from a large secondary storage device, this partitioning time will overlap with the data accessing operation. However, if the data is distributed in a multiple processor environment, the partitioning of data can be performed parallelly with a speed gain of \( n \). The memory contention and the communication overhead problems in such a system are analyzed in Chow and Winslow's paper [10].
Finally after partitioning and parallel sorting, merging takes almost no computing time. We summarize our analysis and discussion in the following theorem.

Theorem 2. For our proposed method of parallel sorting by sampling, the total computation time is given by the sum of equations (1), (2), (3) and (6). If input/output of data are considered then a maximum data transfer or communication overhead of $O(N)$ should be added.
4. Optimal Choice of $\ell$

The choice of $\ell$ is critical to the success of our procedure. Randomness of the sample is also important, but it can be achieved by artificial randomization (see Mendenhall [5]). In general, the system primary storage is limited and we desire to avoid using the low speed secondary storage device unless we have to. Thus we would like to set an upper limit for all sizes of $Q_i$'s. That is, for a given specified $K > 0$ and a small positive number $\alpha$, say .05 or .10, we want to choose the smallest $\ell$ such that

$$\text{Prob. } [ |Q_1| \leq K, Y_i ] \geq 1 - \alpha$$  \hspace{1cm} (7)

where $|Q_1|$ denotes the size of $Q_1$. Now

$$P[ |Q_1| = j_1, |Q_2| = j_2, \ldots, |Q_n| = j_n, \sum_{i=1}^{n} j_i = N-n+1 ]$$

$$= P[ Y_{\ell} = x_{j_1+1}, Y_{2\ell} = x_{j_1+j_2+2}, \ldots, Y_{(n-1)\ell} = x_{j_1+j_2+\ldots+j_{n-1}+n-1} ]$$

$$= P[ Y_{\ell} = x_{j_1+1} ] P[ Y_{2\ell} = x_{j_1+j_2+2} | Y_{\ell} = x_{j_1+1} ] \ldots$$

$$\quad \ldots P[ Y_{(n-1)\ell} = x_{N-j_n} | Y_{(n-2)\ell} = x_{j_1+\ldots+j_{n-2}+n-2} ]$$

$$= \frac{j_1}{j_1-1} \frac{j_2}{j_2-1} \frac{j_3}{j_3-1} \ldots \frac{j_n}{j_n-1} \frac{j_1}{(n-1)\ell-1} \frac{j_2}{(2\ell-1)\ell-1} \frac{j_3}{(3\ell-1)\ell-1} \ldots \frac{j_{n-1}}{(n\ell-1)\ell-1} \frac{j_n}{(2\ell-1)\ell-1}$$

$$= \frac{(N-j_1-1)}{(n\ell-1)(n-1)\ell-1} \frac{(N-j_1-j_2-2)}{(n-2)\ell-1} \ldots \frac{(N-j_1-j_2-j_3-\ldots-j_{n-1}-n-2)}{(2\ell-1)}$$
\[
\binom{j_1}{l-1} \binom{j_2}{l-1} \cdots \binom{j_{n-1}}{l-1} \binom{N-j_1-j_2-\cdots-j_{n-1}}{n-l-1} \binom{N}{n-l-1},
\]

where \( j_i = l - 1, \ldots, N-(n-1) \), for all \( i \), and

\[
\sum_{i=1}^{n-1} j_i \leq N - \frac{n}{\alpha} + 2.
\]

The probability distribution here is \((n-1)\) variate multinomial-beta distribution, or also called \((n-1)\) variate negative hypergeometric distribution (see Sibuya and Shimizu \[9\]). We are interested in developing a computer program to evaluate the probability in Eq.(7) and to find the smallest \( l \) satisfying (7). Because of large \( N \) and its factoria, the computation needs high precision so that each number occupies 400 digits. The program runs on a PDP-11/70 and requires twelve hours computing time. The program is given in Appendix where \( K = 1.2N/n, \alpha = .10 \). Some numerical examples are \( l = 8 \) for \( N = 40 \) and \( n = 4 \), \( l = 20 \) for \( N = 100 \) and \( n = 4 \), \( l = 40 \) for \( N = 200 \) and \( n = 4 \), \( l = 6 \) for \( n = 40 \) and \( n = 6 \), \( l = 12 \) for \( N = 100 \) and \( n = 6 \), \( l = 25 \) for \( N = 200 \) and \( n = 6 \).

Another criterion to choose the optimal \( l \) is to choose \( l \) such that the upper confidence bound (say 97.5% probability) of \(|Q_i|\) is less than or equal to \( K \). Recall that the mean size of \( Q_i \) is

\[
\mathbb{E}(J) = \frac{N+1}{n} - 1.
\]
and its variance is

\[ \text{Var}(j) = \frac{(N-nl+1)(n-1)}{(nl+1)n} . \]

Thus the approximate 97.5% upper confidence bound for \( |Q_i| \) is

\[ \frac{N+1}{n} - 1 + 3 \sqrt{\text{Var}(j)} , \]

and this bound is desired to be less than or equal to \( K \). Thus putting

\[ \frac{N+1}{n} - 1 + 3 \sqrt{\text{Var}(j)} = K, \]

we get

\[ \lambda = \frac{9(N+2)(n-1)}{[N+1-n(K+1)]^2 + 9n(n-1)} - \frac{1}{n} \]

Note that when \( n = 1 \), \( \lambda \) becomes \(-1/n\) which is meaningless, and when \( K = (N+1)/n-1 \), \( \lambda \) is \((N+1)/n\). This is very interesting since the sample size \( n-1 \) is equal to \( N \), i.e., total sampling.
5. Discussion

We have proposed a parallel sorting method by sampling. One critical issue in the method is the parallel partitioning of data into ordered subsets. Detailed computational complexities of the partitioning and sorting are analyzed. Since the size of primary memory is generally limited, its lower bound without excessive I/O to secondary storage is established. The optimal choice of \( k \) which determines the sampling size is also discussed. The analysis will be useful for parallel sorting in local network environment.

ACKNOWLEDGMENTS. We are indebted to Mr. Shing S. Liu of Institute of information science, Academia Sinica, for programming the subroutines ADD, MUL, DIV and DIS.
References


Appendix: Computer Program for finding the Optimal $k$.

**MAIN PROGRAM FOR MULTIVARIATE NEGATION HYPERGEOMETRIC DISTRIBUTION**

```plaintext
INTEGER*4 IA, IL
INTEGER*4 IN, INS
DIMENSION MA(800), MC(800)
COMMON/I0/KPR, NDIM, NDG, FTR, NTR, F0, LUNO
DATA LUNO/4/
'CALL ASSIGN(LUNO, 'MULT2. OUT')
TYPE *, ' N= ' 
ACCEPT *, N
TYPE *, ' NS= ' 
ACCEPT *, NS
K=1.2*N/NS
IN=N
INS=NS
IL=7*(IN+2)*(INS-1)
IA=(IN+1-INS*(K+1))*2+7*INS*(INS-1)
IL=IL/IA
TYPE *, ' IL= ', IL, ' K= ', K
TYPE*, ' ACCEPT L'
ACCEPT*, L
CALL ALLCHO(N, NS, L, K, MC)
NSL1=NS*L-1
DO 80 I=1, NSL1
CALL MUL(MC, MC, I)
NI=N-I+1
80 IF(NI.EQ.0)NI=1
DO 90 I=NSL1, IN
CALL DIV(MC, MC, NI)
90 STOP
END
SUBROUTINE NUMRAT(NS, L, J, MA)

**COMPUTING NS PRODUCT OF BINOMIAL COEFFICIENTS OF THE NUMBER**
**OF NEG-HYPERGEOMETRY**

```
L1=L-1
DO 30 I=1, 800
30 MC(I)=0
MN1=M-N+1
KK=1
IF(MN1-N)4, 2, 1
DO 10 I1=1, N
10 J(I1)=1
IF(J(I1).LT. L1. OR. J(I1).GT. K) GOTO 4
CONTINUE
1000 CONTINUE
DO 12 I1=1, 800
12 MA(I1)=0
CALL NUMRAT(N, L, J, MA)
CALL ADD(MC, MC, MA)
GOTO 03
IF(N-1)4, 5, 6
5 J(1)=MN1
GOTO 1000
6 KK=1
DO 20 I1=2, N
20 J(I1)=L1
J(1)=MN1-(N-1)*L1
IF(J(1).GT. K) GOTO 52
IF(J(1).LT. L1) GOTO 03
DO 13 I1=1, 800
13 MA(I1)=0
CALL NUMRAT(N, L, J, MA)
CALL ADD(MC, MC, MA)
52 LC2=J(1)
JJ=2
GOTO 51
50 IF(JJ. EQ. (N+1)) GOTO 03
NJ=JJ+1
LCJ=MN1-(JJ-1)*L1
IF(JJ. EQ. N) GOTO 22
DO 21 I1=NJ, N
21 LCJ=LCJ-J(I1)
22 IF(J(JJ)-LCJ)7, 8, 7
8 JJ=JJ+1
GOTO 50
7 J(JJ)=J(JJ)+1
JJN=JJ-1
DO 11 I1=2, JJN
11 J(I1)=L1
LC2=LCJ-(J(JJ)-L1)
J(1)=LC2
JJ=2
GOTO 53
51 J(2)=J(2)+1
J(1)=J(1)-1
53 DO 16 I1=1, N
16 IF(J(I1).LT. L1. OR. J(I1).GT. K) GOTO 54
CONTINUE
DO 14 I1=1, 800
14 MA(I1)=0
CALL NUMRAT(N, L, J, MA)
CALL ADD(MC, MC, MA)
SUBROUTINE ADD (MC, MA, MB)
C+
C ARRAY MC = MA + MB
C-
IMPLICIT INTEGER*4 F
COMMON /OP/KFR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MA(1), MB(1), MC(1)
ICARRY = 0
DO 20 I1 = 1, NDIM
  MC(I1) = MA(I1) + MB(I1) + ICARRY
  IF (MC(I1) .LT. NTR) GOTO 10
  ICARRY = 1
  MC(I1) = MC(I1) - NTR
  GOTO 20
10 ICARRY = 0
20 CONTINUE
IF (ICARRY .EQ. 0) RETURN
CALL ERR (1)
END
SUBROUTINE BIT (BC, M, NDG)
C+
C TRANSFORM INTEGER M TO BIT FORM BC
C-
BYTE B0, BC(1)
DATA B0/'0'/
MI = M
DO 10 I1 = 1, 4
  MQ = MI / 10
  MR = MI - MQ*10
  BC(I1) = MR + B0
  MI = MQ
10 CONTINUE
D TYPE*, 'M=', M
D TYPE20, (BC(I1), I1=NDG, 1, -1)
D20 FORMAT(‘,’BC= ‘,’<NDG>’A1)
END
SUBROUTINE DIS (MB, NDAP)
C+
C DISPLAY ARRAY MB WITH NDAP DIGITS AFTER DECIMAL POINT.
C-
IMPLICIT INTEGER*4 F
COMMON /DP/KPR, NDIM, NDC, FTR, NTR, F0, LUN0
INTEGER MB(1), NDAP
BYTE DIG(J2000)
DATA NRSV, N5/0.5/
NPOS = NZR (MB)
D KPRM10 = KPR - 10
D TYPE*, 'NPOS, KPRM10= ', NPOS, KPRM10
D TYPE*, (MB(I1), I1=NPOS, KPRM10, -1)
IF (NPOS .GT. 0) GOTO 20
WRITE(LUN0, 10)
10 FORMAT(/ ' THE NUMBER = ', '.')
RETURN
20 NMIN = MIN0 (NDAP, NDC*KPR)
NDN = KPR - (NMIN-1)/NDC
NST = MAX0 (NPOS, KPR+1)
CALL RST (MB, NST, KPR+1, DIG, NDIG)
NSET = NDIG / 50
IL = 50 * NSET
NR = NDIG - IL
IF (NR .EQ. 0) GOTO 50
NSET = NSET + 1
IL = IL + 50
DO 30 I1 = NDIG+1, IL
   DIG(I1) = '
30 CONTINUE
WRITE(LUN0, 40)
40 FORMAT(/ ' INTEGER PART OF THIS NUMBER := ', '.')
50 DO 70 I1 = NSET, 1, -1
   IR = IL - 49
   WRITE(LUN0, 60) (DIG(I2), I2=IL, IR, -1)
60 FORMAT(IO(I2, 5A1))
   IL = IR - 1
70 CONTINUE
WRITE(LUN0, 80)
80 FORMAT(/ ' DECIMAL PART OF THIS NUMBER := ', '.')
CALL RST (MB, KPR, NDN, DIG, NDIG)
IL = NDIG
90 IF (IL .LE. 0) RETURN
IR = MAX0 (IL-49, 1)
WRITE(LUN0, 60) (DIG(I2), I2=IL, IR, -1)
IL = IR - 1
GOTO 70
END

C+ ARRAY MB = ARRAY MA / NUMBER MDIV
C WHERE MDIV > 0
C-
IMPLICIT INTEGER 4 F
COMMON /DP/KPR, NDIM, NDC, FTR, NTR, F0, LUN0
INTEGER MB(1), MA(1), MDIV
IF (MDIV .LE. 0) GOTO 90
NPOS = NZR (MA)
IF (NPOS .GT. 0) GOTO 10
CALL EQUN (MB, 0)
RETURN
10 IF (NPOS .GE. NDIM) GOTO 30
DO 20 I1 = NDIM, NPOS+1, -1
   MB(I1) = 0
20 CONTINUE
30 FR = F0
FDIV = MDIV
DO 40 I1 = NPOS, 1, -1
   FE = MA(I1)
      FI = FB + FR*FTR
      FQ = FI / FDIV
      FR = FI - FQ*FDIV
      MB(I1) = FQ
40 CONTINUE
RETURN
90 CALL ERR (4)
END
SUBROUTINE EQUA (MB, MA)

C+
C ARRAY MB = ARRAY MA

C-
IMPLICIT INTEGER*4 F
COMMON /OP/KPR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MB(1), MA(1)
DO 10 I1 = 1, NDIM
   MB(I1) = MA(I1)
10 CONTINUE
END
SUBROUTINE EQUN (MB, M)

C+
C ARRAY MB = INTEGER M > 0

C-
IMPLICIT INTEGER*4 F
COMMON /OP/KPR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MB(1), M
IF (M .LT. 0) GOTO 91
DO 10 I1 = 1, NDIM
   MB(I1) = 0
10 CONTINUE
NPOS = KPR
MQ = M
IF (MQ .EQ. 0) GOTO 30
MI = MQ
MQ = MI / NTR
MR = MI - MQ*NTR
NPOS = NPOS + 1
IF (NPOS .GT. NDIM) GOTO 90
MB(NPOS) = MR
GOTO 20
30 IF (NPOS .GE. NDIM) RETURN
RETURN
90 CALL ERR (1)
RETURN
91 CALL ERR (2)
END
SUBROUTINE ERR (IERR)

C+
C OUTPUT ERROR MESSAGE WITH ERROR CODE=IERR

C-
IMPLICIT INTEGER*4 F
COMMON /OP/KPR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MB(1), MA(1), MMUL
DATA KFR, NDIM, NDG, FTR, NTR, F0, LUN0/0, 400, 800, 4, 10000, 10000, 0, 5
DATA ZERO/0./
IF (IERR .EQ. 1) TYPE*, 'DIMENSION TOO SMALL.'
IF (IERR .EQ. 2) TYPE*, 'ARGUMENT < 0.'
IF (IERR .EQ. 3) TYPE*, 'MULTIPLIER < 0.'
IF (IERR .EQ. 4) TYPE*, 'DIVIDER <= 0.'
IF (IERR .EQ. 5) TYPE*, 'MA-MB < 0.'
A = 1. / ZERO
CALL EXIT
END
SUBROUTINE MUL (MB, MA, MMUL)

C+
C ARRAY MB = ARRAY MA * NUMBER MMUL
C WHERE MMUL >= 0

C-
IMPLICIT INTEGER*4 F
COMMON /OP/KFR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MB(1), MA(1), MMUL
IF (MMUL .LT. 0) GOTO 90
NPOS = NZR (MA)
IF (NPOS .GT. 0 .AND. MMUL .GT. 0) GOTO 20
CALL EQUN (MB, 0)
RETURN

20 FQ = F0
FMUL = MMUL
DO 30 I1 = 1 , NPOS
   FB = MA(I1)
   FI = FQ + FB*FMUL
   FQ = FI / FTR
   FB = FI - FQ*FTR
   MB(I1) = FB
30 CONTINUE
IF (NPOS .EQ. NDIM .AND. FQ .GT. F0) GOTO 91
MB(NPOS+1) = FQ
IF (NPOS+1 .GE. NDIM) RETURN
DO 40 I1 = NPOS+2 , NDIM
   MB(I1) = 0
40 CONTINUE
RETURN
90 CALL ERR (3)
RETURN
91 CALL ERR (1)
END

C+
C NZR = THE POSITION OF THE FIRST NONZERO ELEMENT IN ARRAY MB

C-
IMPLICIT INTEGER*4 F
COMMON /OP/KFR, NDIM, NDG, FTR, NTR, F0, LUN0
INTEGER MB(1)
DO 10 I1 = NDIM , 1 , -1
   IF (MB(I1) .NE. 0) GOTO 20
10 CONTINUE
NZR = 0
RETURN
20 NZR = I1
SUBROUTINE RST (MB, IL, IR, DIG, NDIG)

C+ TRANSFORM INTEGER MB(IL..IR) TO BIT-FORM DIG WITH NDIG DI
C-

IMPLICIT INTEGER*4 F
COMMON /CP/KPR, NDIM, NDC, FTR, NTR, F0, LUN0
INTEGER MB(1), IL, IR, NDIG
BYTE BC(10), DIG(1)

NDIG = 0
DO 20 I1 = IR, IL
   CALL BIT (BC, MB(I1), NDG)
   DO 10 I2 = 1, NDG
      NDIG = NDIG + 1
      DIG(NDIG) = BC(I2)
   10 CONTINUE
20 CONTINUE
END