NATIONAL RADIO ASTRONOMY OBSERVATORY
SOCORRO, NEW MEXICO
VERY LARGE ARRAY PROGRAM

SGP MEMO NO. 2

VLA COMPUTER MEMORANDUM NO. 145

1. VLA SORTING ENGINE--DESIGN CONSIDERATIONS

by J. Hudson, G. Hunt, D. Ehnebuske, A. Braun
May 2, 1978

The VLA sorting engine has as its task the ordering of spectral line and continuum data in (u,v) sequencing to facilitate gridding the data for Fast Fourier Transforming (FFT) into a map of the radio source. It also has the task of breaking out frequency channels, a job which perhaps can be shared with other processors.

In specifying the architecture of the computer system for this problem we need to consider first the requirements of the gridding machine, since this will put constraints on the ordering and layout of visibility data records. Second, we need to consider the source and ordering of the data records arriving from the spectral line array processor (AP) via the MODCOMP minicomputers "Cora" and "Corbin." Finally, we need to give some thought to the sorting process itself, in order to get an idea of throughput times and memory requirements to be expected.

1.1. A Hypothetical Gridding Engine

It is not our intention to design any part of the mapping system at this time, but we must have better than a vague notion of the requirements of the first stage of that process: the gridding. It is supposed that the gridding machine will prepare a two-dimensional array of data, of size ranging up to 8192 X 8192 complex numbers, perhaps truncated to half that size if we take advantage of the Hermitian property of the visibility function:

$$V(u,v) = V^*(-u,-v).$$

It is also a necessary requirement for the gridding engine that the data be convolved with some function C(u,v) having limited range in the (u,v) domain. If the present continuum mapping system is any guide, that range can be as large as 6 X 6 grid points, meaning that 6 columns at a time of the array must reside in fast memory. For the 8192 map size, this requires fast storage capable of holding 49152 complex numbers. We anticipate the most common map size to be 1024 X 1024, whence it would be practical to organize visibility data records with no more than 8 frequency channels per record. (For the larger maps, this leads to an obvious inefficiency unless the user is content to map only a fragment of his data, or to have the channels averaged together.) If input records contain more than one frequency channel, then mass storage is required to hold the gridded data for all channels but the one currently being mapped. Most probably, mass storage will be required anyway for the three-dimensional transforms (required to correct for the map curvature aberration).

It is also anticipated that calibration will be accomplished by the gridding engine, using a concept similar to the antenna gain tables in current use in the DEC-10 continuum system. Thus, records must bear such information as the antenna pair and a date/time stamp, in addition to the (u,v,w) coordinate. If data records contain 8 frequency channels, with 4 bytes (32 bits) per complex visibility, the overhead for these additional data can be kept to 20% (assuming 8 bytes of overhead per record). If further breakout of frequency channels is desired, one should use instead an indexing scheme whereby the record's position in the data base dictates the location of its identifying information.

From this we conclude that visibility data records to be handled by the sorting and gridding engines should be broken down into at least 32 different data sets (8 frequency channels per record), assuming 256 channels. Further breakdown may be desirable, but at the expense of separating identification data from the visibility data.

1.2. Cora and Corbin

It would be highly desirable for the proposed system to tie in with the two MODCOMP II minicomputers, Cora and Corbin, which will receive visibility data from the Array Processor. Cora and Corbin are limited to 64K words of memory, and are somewhat inhibited by a maximum DMA data transfer rate of ~600 Mbytes/second. Nevertheless, it is felt they can perform useful functions as an adjunct to the various sorting algorithms discussed below. They should be able to accomplish a 32-way split into frequency groups, as an assist to the customary external sorting algorithm; they should also be able to accomplish a random-access store of records to facilitate the "pidgeonholing" process.

It is assumed it will be possible for the AP to feed data to Cora and Corbin in one or two frequency groups (256 or 128 channels), with baselines ordered in any way desired. For instance, they could arrive in order of the desired (u,v) sorting key.

1.3. Some Assumptions

Let us suppose that computers are selected for the sorting machine which are capable of virtual memory addressing, and that context switching between address spaces (64 Kbytes in size, say) requires very little effort. We assume the program can reside outside the address space. We suppose the computers are capable of interfacing to large capacity disk drives having capacities of 400 Mbytes, average seek times of ~40 msec, and byte transfer times of ~1 usec/byte.

1.4. Disk Sorting

Of the various approaches available for achieving the desired (u,v) ordering, perhaps the customary method should be explored first: that of sorting the data into "strings" of correctly sequenced records, followed by merging the strings into one correctly sequenced data set. Let:

Nf = Total number of frequency channels (256 assumed earlier)
Nu = Total number of (u,v) data points in entire data set
S = Data set size, bytes = 4.Nf.Nu, roughly
F = Number of partitions of S into frequency groups
B = Amount of core available for buffers during merge
x = Blocking factor of data records (we will read x at one time)
M = Merge order
p = Number of merge passes through entire data set
L = Logical record size, bytes (holding Nf/F complex numbers + overhead)
Lp = Physical record size, bytes
tc = Compute time to compare 2 sorting keys
ts = Disk seek time
tt = Data transfer time during I/O

We will neglect for the moment the problem of internally sorting the data records. (This will perhaps be accomplished complelety by Cora and Corbin.) Assuming now that we have sorted strings of length x records, we proceed to make p passes through each frequency group (S/F bytes long), merging the strings of length x until there is one string for each frequency group, of length Nu records. To simplify the discussion we assume

$$Nu = M^{p} .$$

Thus

$$p = \log_{M} Nu.$$

In practice (and in the time estimates below), we round p up to the next greater integer. This does not result in the optimum merge pattern for the given number of records, a problem taken up by Knuth (1973, pp. 361--378). The total number of I/O operations for a merge of Nu records is then

$$Nio(\text{per freq. group}) = 2.(Nu/x).\log_{M} Nu,$$

where we count both input and output operations. For the entire data set, we must accomplish this F times, and so we write for the total number of I/O operations

$$Nio = 2.(S/xL).\log_{M} Nu$$

$$= 2.(M+1).(S/B).\log_{M} Nu.$$

where we have assumed the available buffer space, B, is occupied by M+1 buffers (M input, 1 output), each holding x records of length L. Nio is minimized for M=4, a result which is independent of the data set size, buffer size, and number of frequency groups.

The total I/O time for merging is

$$Tio = Nio.ts + 2.p.S.tt$$

$$= 2.(\log_{M} Nu).S.[(M+1).ts/B + tt].$$

The total compute time is

$$Tc = F.p.Nu.(M-1).tc + F.p.Nu.tt$$

$$= (\log_{M} Nu).(S/L).[(M-1).tc + tt],$$

where we assume the same transfer time for core-core transfers as for core-disk. Putting some numbers in for ts, tt, and tc, let:

ts = 40 msec/IO operation,

tt = 1 usec/byte,

tc = 3 usec/byte.

We see that the ratio

$$\frac{2.[(M+1).ts/B + tt]}{[(M-1).tc + tt]/L}$$

is considerably greater than unity, for just about any choice of M, B, and L. Thus, IO time dominates the throughput time in the normal merging process. We can safely assume that most compute time can be overlapped with disk seeking, whence it disappears from consideration, except perhaps for arguing for 1 minicomputer in place of 2. We now arrive at TABLE I, which shows total merging time as a function of merge order, M, and buffer size, B.

From TABLE I, we conclude that it is marginally possible for one

TABLE 1. Merge time for 12 hour data sample.

M	p	B, Kbytes		
		64	256	512
3	13	39.8 hr.	18.6 hr.	15.1 hr.
4	11	39.6	17.2	13.5
5	9	37.2	15.3	11.7
6	8	37.5	14.7	10.9
7	8	41.8	15.8	11.4
8	7	40.4	14.8	10.5
9	7	44.2	15.8	11.0
10	7	47.9	16.7	11.4

minicomputer with the capability of virtually addressing > 512 K bytes of memory to handle the sorting task. In doing so we have glossed over some hardware-dependent considerations: 1) Can the computer perform direct memory access data reads and writes from and to disk of blocks of memory exceeding the range of directly addressable storage? (Probably not. This is only 64 kbytes on some machines.) 2) Is there a penalty assessed in data transfer whenever a disk track boundary is reached? (We are supposing not; there would be a penalty only when the heads must move from cylinder to cylinder.) 3) Since the merge time is heavily dependent upon seek time, we have ignored the saving that would result from having the input data set on several different spindles, with perhaps separate data paths (via separate disk controllers).

Aside from time, one must also consider disk space occupied by the data sets during the sort/merge. If we break the 12 hour sample into F portions, it is necessary to have (F+1) times the space occupied by one portion during the merge. For F=32, this is an extra 3.1%. Also, the approach taken to handling the data in the real world is not to wait until a full 12 hour load has been accumulated before starting the merge; instead, we want to begin the first stage of merging as soon as enough data accumulate to warrant the effort. When M sorted strings of length M records are accumulated, we want to perform pass 2 of the merge operation, creating a data set of size M^2, and so forth. Assuming we can merge as fast as the data flow, as Table I suggests, we will be left at the end of 12 hours with data strings of varying length: let us say, M strings of length M, M-1 strings of length M^2, and so forth, including

M-1 strings of length M^{p-1}. This leaves us with some cleaning-up to do: one M-way merge of M records, one M-way merge of M^2 records, and so forth, up to an M-way merge of the last stage. The fraction of the merging effort left to do after completion of a 12 hour observation is then:

$$\frac{M + M^2 + \ldots + M^p}{p.M^p} = \frac{M^{p+1} - M}{p.(M-1).M^p} \cong \frac{M}{p.(M-1)}$$

which, for M = 8, p = 7 (best case in Table 1) works out to 16.3% of the total time. While this is going on, of course, we can be doing the first few merges required for the next 12 hour observation. But we are left with an extra ~20 per cent. storage requirement, in addition to the 16% delay. Requests to make maps during the merging process we think would require pretty sophisticated coordination in order not to disrupt the proceedings while making most of the pieces of partially merged strings available for processing.

1.5. Keysorting and Pidgeonholes

For both of these techniques (which we shall see are very similar), it is advantageous to keep the frequency channels together during the (u,v) sort, and then to break them out later.

The keysorting process becomes practical when the length of records is large. Then one simply records for each record the place where it has been written, together with the key on which he desires to sort. When all data records are stored, a sort of the key records takes place; whence the data records are simply retrieved according to the pointers kept with the keys. Storage of data records is sequential; retrieval is random, with each read operation requiring time for head seeking. For 1.6×10^6 data records of length 1024 bytes (256 complex numbers), it would require roughly 1/2 hour sequentially to write the data, and 17.8 hours to retrieve them, assuming, as above, tt = 1 usec/Byte, and ts = 40 msec/head seek. Thus, two independent data paths are called for.

The operation of breaking out frequencies into F different groups requires filling F large buffers, to be written into F different places on disk storage. Taking F=32 and assuming, say, 4 KBytes/buffer, the output side of this operation requires 390000 head seeks, at the rate of 40ms/seek, or 4.32 hours. In practice, we would break the input data set into several chunks while data is being collected, and overlap the operation of frequency breakout. This has the virtue of cutting down on extra disk storage to just those disk spindles required to retain the collected data as it comes in, plus those involved in emptying data to the frequency breakout machine.

Equivalent in its demands on the sorting engine is a "pidgeonhole" sort, whereby the data are written selectively on different parts of the disk, so that they can be retrieved sequentially in (u,v) order. This is made possible by a priori knowledge of the baseline projections which determine the sort key (u,v). That is to say, we could, in principle, carry out the key sort ahead of time. We also then allocate exactly what disk area is needed, so pointers can be stored with the keys. A slightly more flexible approach would be not to worry about, say, the v coordinate, and group the data into perhaps 2048 bins, each for a small range in u. (Say, 1024 bins for a half-plane, taking into account the Hermitian property of the visibility function.) Now, as each 256 frequency (1024 Byte) record is received, it is written onto one of the 1024 disk partitions. It is best to allocate clusters of, say, 20 records, as needed. This leads to much less waste than allocation according to estimated storage requirements per bin; statistical fluctuations in the record counts lead to high chance of overflowing one or more bins when the bins are on the average only 60 to 70% filled. It is suggested that one frequency channel be sacrificed to hold the (u,v) coordinate, and another to hold a backward link, linking the record to its predecessors belonging to the same range in u coordinate. One aid can be drawn from a priori knowledge of the sorting keys: in any 10-second interval the optimum seek pattern is very similar to the seek pattern in the preceding interval. It is possible in principle for the pidgeonholing computer to request its input records in the order optimizing head seeks on the output disk drives, thus ensuring that a backlog will not occur. At the rate of 351 records/10 seconds, we can allow a head seek "budget" of 28.5 msec/record.

Pidgeonholing, like keysorting, also requires the F-way breakout of data records after the storage is completed. Penalty in time is slight (~4 hours) for the breakout, and the cost in storage to hold the data during breakout can be kept to the same level as the keysort technique.

1.6. A Hardware Configuration

One configuration which seems to satisfy the needs of all approaches to the sorting problem appears in Figure 1.

The machinery would have differing functions, depending upon the choice of sorting algoritm. For the classical sort/merge, Cora and Corbin would be responsible for writing out the data into F distinct frequency groups on their disk storage units. For F=32, this would mean (assuming a buffer capacity of 60 KBytes for each computer), the 10 second workload of 180 Kbytes would be divided into 3 core loads, requiring 32 disk head seeks apiece, or 96 seeks/10 seconds (a budget of 104 msec/seek allowed). Part of the time, Cora and Corbin access a large group of adjacent data belonging to one frequency group, and feed it to

```
                             ---------------
                             | Array       |
                 ------------| Processor   |------------
                 |           |             |           |
                 |           ---------------           |
         ---------------                       ---------------
         | Cora        |                       | Corbin      |
         |             |                       |             |
         |             |                       |             |
         ---------------                       ---------------
          |           |                         |           |
          | --------- | ---------               | --------- | ---------
          | |49 MBy.| | |49 MBy.|               | |49 MBy.| | |49 MBy.|
          P-|Disk + | P-|Disk + |               P-|Disk + | P-|Disk + |
          | |Control| | |Control|               | |Control| | |Control|
P = Peri- | --------- | ---------               | --------- | ---------
   pheral |           |                         |           |
  Control |           |      ---------------    |           |
   Switch |           |      | Interface   |    |           |
          |           -------| MODCOMP 7810|-----           |
          -------------------| (64 KBytes) |-----------------
                             ---------------
                                    |
                             ---------------
                             | Sorter      |
                             | DEC 11/60   |
   --------------------------| (128-512KBy)|
   |                         ---------------
   |                           |         |
   |     -----------------------         ---------------
   |     |                                             |
   | -------                                        -------
   | |Cont.|-------                                 |Cont.|-------
   | -------      |                                 -------      |
   |          ---------                                     ---------
   |          |300 MBy|-                                    |300 MBy|-
   |          | Disk  ||-                                   | Disk  ||-
   |          ---------||-                                  ---------||-
   |           ---------||-                                  ---------||-
   |            ---------||-                                  ---------||-
   |             ---------||                                   ---------||
   |              ----------                                    ----------
   |                 (5)                                            (5)
   |
   |                         ---------------
   |                         | Gridley?    |
   --------------------------| (Gridding   |
                             |    Engine)  |
                             ---------------
```

Fig. 1. The Sorting Machine

the sorting engine. The data are automatically sorted by (u,v), a service provided by the array processor in conjunction with Boss (this done on a 10 second record basis). The next machine merges the sorted 10 second groups into longer ones on their way out to reside on the large-capacity disk units. We provide 10 spindles of 300 Megabyte capacity, 6 of which are required for holding 12 hours' data, 1 for scratch, and 3 for keeping previous observations. The memory required for the sorter should be on the upper end of the scale 128-512 KBytes for more efficient operation.

For the pidgeonholing process, Cora and Corbin would carry out 176 seeks apiece/10 seconds (budget of 56 msec/seek). (In reality, it would be more like 176 seeks/6 seconds (budget of 34 msec/seek), to allow more flexibility for the array processor.) The AP would provide records of 256 frequency channels in some order (requested by Cora and Corbin) that optimizes disk access. When one entire disk pack (holding 48000 records) is filled (about every 40 minutes) it is turned over to the sorter for emptying; meanwhile the other disk is being written by Cora or Corbin. The sorter then proceeds with the frequency breakout. Assuming a 32-way split, with 4 KBytes/buffer (128 KBytes for buffers), the 12000 writes and 2400 reads would occupy it for 9.6 minutes. It must also perform the same service for the other 48000 records coming over from the other processor, which will require a total of 19.2 minutes, assuming no overlap in the I/O.

The purpose of the MODCOMP 7810 computer is to provide an interface between the MODCOMP systems and the other vendor (DEC, say). Another possibility is to replace the peripheral control switch system with one having dual-port disks, cutting down on the number of controllers. This alternative has the penalty of going to larger disks, since the 49 Mbyte drives are not available with dual ports. Since the sectoring of the disks will differ from manufacturer to manufacturer, it will probably not be possible to configure the system with different manufacturers' computers on either side of the dual-port drives.

1.7. Cost Estimates

The following are cost estimates for two possible configurations involving the MODCOMP computers:

Configuration A -- Dual-port drives

3	MODCOMP 4138 (100 MByte) disks + controllers	$78K
1	MODCOMP 4138 disk, no controller	19K
1	MODCOMP 7810 computer, 32 Kwords MOS memory	6K
1	MODCOMP-DEC CPU link (est. 2 man-months effort)	5K

		$108K

Configuration B -- Single-port drives & peripheral control switches

4 MODCOMP 4134 (49 MByte) disks + controllers	$92K
4 MODCOMP 4906 Peripheral control switches	12K
1 MODCOMP 7810 computer, 32 Kwords MOS memory	6K
1 MODCOMP-DEC CPU link (est. 2 man-months effort)	5K

	$115K

It is not clear that the MODCOMP 4138's can be obtained for the price shown; our estimate is based upon a recent NRAO procurement of single port drives at this cost (OEM supplier: AMPEX). Our figure for the single-port configuration is a little more firm (manufacturer's list prices); hence it is the configuration shown in the diagram.

The sorting machine will have to be equivalent to a DEC 11/60 or MODCOMP 7860, since memory mapping is required, and the rapid data transfer rate obtainable will be useful, though not absolutely necessary. MOS memory is the most economical, and is suggested. The largest capacity disk drives obtainable are the CALCOMP TRIDENT series, model T-300. We find that AMPEX makes an equivalent device, but it is not known whether it interfaces to a minicomputer. The CALCOMP drives have been interfaced to DEC-11's; the availability of interfaces and software may make DEC-11's much more attractive than their competitors. We tentatively suggest the following configuration:

Sorting Engine

DEC 11/60 computer, with 256 KBytes MOS memory	$39K
10 CALCOMP T-300 disk drives, without controllers	120K
2 controllers for the above	14K

	$173K

The total estimated hardwarre cost, assuming Configuration B, is $288K. This does not include software development, which we roughly estimate at about a man year's effort -- this is highly qualified: it makes no provision for additional complexity introduced by the desire of occasional users to share observational time, use of multiple subarrays, processing of calibration sources, and the numerous error checks that would go into a polished, sophisticated system.

Reference

Knuth, D.E. (1973) The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Mass.