Sortnet: A Program for Building Sorting Networks

Kenneth E. Batcher, Sherenaz W. Al-Haj Baddar

Kent State University
Department of Computer Science
Kent, Ohio 44240
batcher@cs.kent.edu
salhajba@cs.kent.edu

***Abstract*— Sorting networks are cost-effective multistage interconnection networks with sorting capabilities. The fastest implementable sorting networks built so far consume $\Theta(N\log^2 N)$ comparisons, and generally, use the Merge-sorting strategy to sort the input. In order to help analyze and synthesize sorting networks, a software tool needs to be developed. This technical report introduces Sortnet, a software tool developed by Kenneth Batcher, to help build better sorting networks.**

***Index Terms*—Sorting Networks, 0/1 principle, Partial Ordering.**

I. INTRODUCTION

Several interconnection networks schemes have been developed so far among which are multistage interconnection networks. These widely used networks combine the cost effectiveness of bus-based interconnection networks with the connectivity of crossbar-based networks. Several multistage interconnection networks were developed including sorting networks.

Van Voorhis [1] defines a sorting network as a circuit with N inputs and N outputs such that for any set of inputs $\{I_1,I_2,\ldots,I_N\}$, the resulting output is the set $\{O_1,O_2,\ldots,O_N\}$. The output set must be a permutation of the input set $\{I_1,I_2,\ldots,I_N\}$. Moreover, for every two elements of the output set O_j and O_k, O_j must be less than or equal to O_k whenever $j \leq k$.

Sorting networks are constructed using stages (steps) of basic cells called Comparator Exchange(CE) modules. A CE is a 2-element sorting circuit. It accepts two inputs via two input lines, compares them and outputs the larger element on its high output line, whereas the smaller is output on its low output line. It is assumed that two comparators with disjoint inputs can operate in parallel. A typical CE is depicted in Figure 1[2].

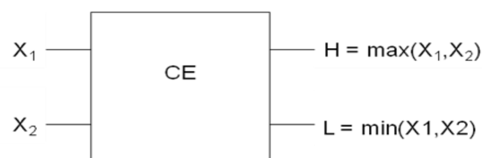

Figure 1. A comparator exchange module

Theoretically optimal N-element sorting networks that deploy $\theta(N\log N)$ comparisons have been designed, namely the AKS networks. However, such networks have not been implemented yet due to their impracticality[3]. On the other hand, several practical $\theta(N\log^2 N)$ techniques for building sorting networks exist among which is the Merge-sorting technique developed by Batcher[2]. This technique has been generally deployed for building the best performing sorting networks known so far. Thus, it's vital to develop a software tool that can help sorting networks designers build better sorting networks. This technical report introduces Sortnet, a software tool, developed by Kenneth Batcher to help analyze and synthesize N-element sorting networks, where $N \leq 32$. Sortnet has already facilitated discovering an 11-step 18-element sorting network that outperforms the 12-step best known solution for this problem[4].

Section 2 reviews some basic mathematical concepts necessary for discussing the subset of Sortnet commands highlighted in this report. Section 3 describes Sortnet depicting its basic functionality and introducing a subset of its commands. Section 4 shows how some parts of Sortnet work. Finally, Section 5 concludes the technical report and highlights future work.

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar

2. MATHEMATICAL BACKGROUND

This section illustrates some basic mathematical concepts necessary for discussing Sortnet commands. These concepts include: Knuth diagrams, 0/1-principle, partial ordering, and Haase diagrams.

2.1 Knuth diagrams

Knuth diagrams are pictorial representations of sorting networks that help distinguish the several steps constituting the investigated network[5]. In a typical Knuth diagram, each input element is represented by a horizontal line and each CE is represented by a vertical line connecting the two elements being compared. The elements being sorted are assumed to have numeric labels. Thus, an N-element network will have its input elements labeled from 0 through (N-1). Moreover, the network's top most element is assigned the largest label and the bottom most element is assigned the smallest. It is also assumed that the elements are fed into the left most end of the network and received sorted at the other end, with the maximum element being the top most element. Figure 2 illustrates a typical Knuth diagram for sorting 4 elements[5].

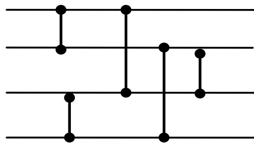

Figure 2. A Knuth diagram for a 4-element sorting network

2.2. The 0/1 principle

The 0/1 principle plays a vital role in building and verifying sorting networks. It states that a given N-element sorting network sorts N inputs correctly if it sorts all the 2^N binary strings of length N [5].

A 0/1 case is a sequence of length N where the value of each entry in the sequence is either 0 or 1. If a given 0/1 case A, of length N, has j zeros then it has N-j ones. When a comparator element compares two entries in A, it might swap the values of the compared entries. However, the number of zero and one entries in A remains the same. Hence, a series of comparator elements sorts case A iff it rearranges A's entries such that the first j locations hold zeros and the next N-j locations hold ones.

The number of zeros(ones) in a given binary sequence of N bits ranges between 0 and N which implies that there are N+1 0/1 sorted cases. Hence, an N-element sorting network, that rearranges any of the 2^N input permutations into one of the N+1 possible sorted 0/1 cases, can sort any arbitrary sequence of N elements.

2.3. Partial ordering and Haase Diagrams

A total order relation is imposed on the set of elements sorted by a sorting network. This total ordering is achieved by the end of the sorting process. However, only a partial order relation exists on this set at any arbitrary step in the sorting process prior to the last step, and such a set is called a **p**artial **o**rder **set** (poset)[6]. Having a partial ordering relation, denoted by R, on a set of elements implies that there may exist a pair of elements x and y, such that there is at least one 0/1 case where x=0 and y=1, and at least another 0/1 case, where x = 1 and y = 0.

Haase diagrams are used in order to visualize the progress of a sorting network [6]. In a conventional Haase diagram, elements are represented by vertices and relations among them are represented by edges. An edge running from vertex x to vertex y exists iff there does not exist a 0/1 case in which x=0 and y=1. It is noticed that the relative positioning of an edge's two endpoints implies its direction. More preciously, if vertex x appears above vertex y, then the direction of the edge running between both is assumed to be from x to y.

Figure 3 depicts the progress of the sorting network illustrated in Figure 2 using Haase diagrams. In the first step, two partial ordering relations are formed and such diagram is called a multi-segment poset. In the second step, the two segments are combined into a one-segment poset, or simply a poset. Finally, step 3 transforms this poset into a totally ordered chain of elements.

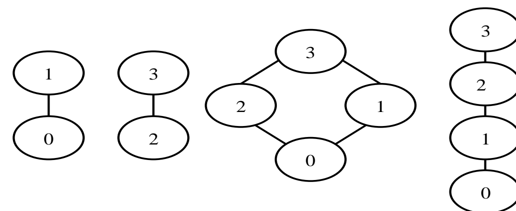

Figure 3. Haase diagrams tracking the sorting of 4 elements

3. SORTNET: A SORTING NETWORKS DESIGN TOOL

Some of Sortnet commands can be used to build, manipulate, and save a CE-list; an ordered list of CEs. Other Sortnet commands can then be used to: generate the corresponding set of 0/1-cases; count the number of these cases; see the poset of elements arising from this set of cases; and see the Shmoo chart that corresponds to the current stage of the sorting network. This section describes Sortnet and highlights a subset of its commands.

3.1 The interface of Sortnet

Whenever Sortnet is waiting to receive a command it prompts the user with **sortnet->**. A Sortnet command consists of:

1. A mnemonic specifying the operation to be performed;

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar

2. A list of zero or more arguments; and
3. A semi-colon(;), if necessary, to indicate the end of the argument list.

Whitespace(s)(one or more spaces, tabs, or new lines) must separate the mnemonic from the first argument, each argument from the next argument, and the last argument from the semi-colon at the end of the argument list. Each command should be followed by a newline(return key) to make sure that the Operating System(OS) sends it to Sortnet. Each mnemonic has one, two, or three elements with a dot(.) separating each element from the next one. The first element specifies an action, the second element(if present) specifies the object to be acted upon, and the last element(if present) is a modifier.

3.2 Sortnet's commands for manipulating CE-lists

The commands listed in this subsection are useful for creating and modifying the CE-list.

3.2.1 ENT.CE num_{lo} num_{hi} ... num_{lo} num_{hi}

This command enters the list of CEs from the keyboard into the CE-list. To enter K comparators into the CE-list there must be 2K numbers in the argument list-the first two numbers specify the first low and high indices of the first CE, the next two numbers specify the low and high indices of the second CE, etc. For example, the Sortnet command **ENT.CE 0 1 2 3 0 2 1 3 1 2 ;** will enter the CE-list that corresponds to the Knuth diagram illustrated in Figure 2.

3.2.2 SHOW.CE

This command displays the current CE-list on the monitor, with comments to show the comparators of each step. For example, Figure 4 shows the output of executing the SHOW.CE command after executing the ENT.CE command described in subsection 3.2.1.

```
sortnet-> SHOW.CE
/* STEP 1 */
    0 1    2 3
/* STEP 2 */
    0 2    1 3
/* STEP 3 */
    1 2
/* 5 comparators in 3 steps. *
sortnet->
```

Figure 4. The SHOW.CE display of the five CEs illustrated in Figure 2.

3.2.3 WR.CE file_name

This command writes the CE-list into the specified file. The contents of the file are the same as the output of the SHOW.CE command.

3.2.4 RD.CE file_name

This command reads the CE-list stored in the specified file. This file can be created by the WR.CE command and the comments this file contain are ignored.

3.2.5 CUT.CE.STEPS numlaststep

This command removes all comparators in the CE-list that are in steps past numlaststep. For example, applying the CUT.CE.STEPS 1 command on the CE-list shown in Figure 3 will remove the comparators in STEP 2 and STEP 3 leaving the two comparators in STEP 1.

3.2.6 CLR.CE

This command simply clears out all comparators in the CE-list.

3.3 Sortnet's commands for manipulating 0/1-cases

The commands listed in this subsection are used for generating and analyzing 0/1-cases.

3.3.1 GEN.CASES

This command generates a set of 0/1-cases from the current CE-list. After entering a CE-list, using either the ENT.CE or the RD.CE commands, the GEN.CASES command is used to generate the corresponding set of 0/1 cases. This enables other Sortnet commands like, SHOW.POSET and SHOW.SHMOO , to execute and produce their designated outputs.

3.3.2 SHOW.POSET

This command displays on the monitor, a 5-column table describing the poset of the elements created by the current set of 0/1-cases. There is a row in the table for each element with:

- column 1 showing the index of the element, k;
- column 2 showing the number of elements that are greater than element-k;
- column 3 showing the number of elements that are less than element-k;
- column 4 showing the indices of elements that cover element-k; and
- column 5 showing the indices of elements that are covered by element-k.

```
sortnet-> SHOW.POSET

POSET CONSIDERING ALL 0/1-CASES
+-----+-----------------+----------+----------+
|     | Number of Keys  |   Keys   |   Keys   |
|  k  +-------+---------+ covering | covered  |
|     |  > k  |   < k   |    k     |   by k   |
+-----+-------+---------+----------+----------+
|  0  |   3   |    0    |  1  2    |          |
|  1  |   1   |    1    |  3       |  0       |
|  2  |   1   |    1    |  3       |  0       |
|  3  |   0   |    3    |          |  1  2    |
+-----+-------+---------+----------+----------+
```

Figure 5. The poset table generated after the first two step of the network depicted in Figure 2

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar

The rows are ordered by segment (with horizontal lines separating the segments) with the elements in each segment ordered by their height in the segment. For example, Figure 5 shows the table of the poset generated after executing the first two steps of the network depicted in Figure 2.

3.3.3 SHOW.SHMOO

A Shmoo chart is a two-dimensional diagram where each column shows all 0/1 cases with the same number of zeros and each row shows an element. Figure 6 illustrates the Shmoo chart generated after applying step 2 of the 4-element network depicted in Figure 2.

```
    Number of Zeroes in Case
    00000    No. of Cases
    43210    where key = 1

3: 01111 :          5
2: 00-11 :          3
1: 00-11 :          3
0: 00001 :          1
```

Figure 6. The Shmoo chart of the 4-element network after step 2

The columns of a typical Shmoo chart are ordered according to the number of zeros in the set of 0/1 cases with the case of all zeros at the left of the chart and the one with all ones at the right of it. The elements in the rows are ordered according to the number of the 0/1 cases in which that particular element is 1. The entry in the chart for a given row and a given column can be either:

- 0: if the value of the element is 0 for all 0/1 cases in a given column,
- 1: if the value of the element is 1 for all 0/1 cases in a given column, or
- -: if the value of the element is equal to 0 at least for one 0/1 case and equal to 1 at least for another 0/1 case in a given column.

A shmoo chart becomes dash-free when all elements get sorted , i.e. all entries in the Shmoo chart are either zeroes or ones.

3.3.4 SHOW.GOODCE

This command considers all possible comparisons and for each it calculates the number of Shmoo chart dashes that will be removed in case that comparison is made. The output of this command is displayed in the form of an upper triangular (N-1)x(N-1) table. The entry at the ith row and j^{th} column contains the number of dashes removed from the Shmoo chart when comparing elements i and j, and a dot will be displayed whenever such a comparison removes no dashes at all. Since the comparisons are commutative, the table needs to be printed in an upper triangular format to avoid repeating commutative GOODCES values. Figure 7 illustrates the output of the SHOW.GOODCE command when it is applied after generating the cases of the first step of the network depicted in Figure 2.

```
sortnet-> SHOW.GOODCE

Number of Shmoo Chart Dashes Removed by Each Possible CE
for all cases.

High:  3  2  1
 Low
   0:  2  3  .
   1:  3  2
   2:  .
```

Figure 7. The output of SHOW.GOODCE after applying the first step of the network depicted in Figure 2.

3.3.5 SHOW.DIFF

This command considers all possible comparisons and for each it calculates the number of 0/1-cases that get affected by the designated comparison. The output of this command is displayed in the form of an upper triangular (N-1)x(N-1) table. The entry at the ith row and j^{th} column contains the number of cases affected by comparing elements i and j, and a zero designates that such a comparison affects no 0/1-cases at all. Since the comparisons are commutative, the table needs to be printed in an upper triangular format to avoid repeating commutative DIFF values. Figure 8 depicts the output of the SHOW.DIFF command when it is executed after applying the first step of the network illustrated in Figure 2.

```
sortnet-> SHOW.DIFF

Number of 0/1-Cases Affected by each Possible CE for all cases.

Maximum Value = 2

High:  3  2  1
 Low
   0:  1  2  0
   1:  2  1
   2:  0
```

Figure 8. The output of SHOW.DIFF after applying the first step of the network depicted in Figure 2.

3.3.6 SHOW.BESTCE

This command considers the outcomes of the SHOW.DIFF and SHOW.GOODCE commands and consequently finds the CEs with the best GOODCE as well as DIFF values and prints them along with the number of the earliest step in which they can be deployed. It is then up to the network designer to pick the CEs she/he thinks are the best and add them to the CE-list. Figure 9 describes the output of the SHOW.BESTCE command after applying it to the cases generated by the first step of the network depicted in Figure 2.

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar

```
sortnet-> SHOW.BESTCE

Best Comparators considering all cases.

Step Dashes CasesSwapped   Comparators

   2      3            2   0:2 1:3
```

Figure 9. The output of SHOW.BESTCE after applying the first step of the network depicted in Figure 2.

4. HOW SORTNET WORKS

This section describes how some parts of Sortnet work.

4.1 Each 0/1 Case

For an N-element sorting network, the program maintains a set of 0/1-cases. Each 0/1-case is a sequence of N bits with each bit showing the value, 0 or 1, of one of the elements for that case. An integer in C contains 32 bits so it's convenient to limit N to 32.

4.2 The Number of 0/1 Cases

At the start, before any comparators are treated, the N elements are not sorted at all so theoretically the program starts with 2^N 0/1-cases. If N = 32 then $2^N = 2^{32} = 4{,}294{,}967{,}296$. Does Sortnet really start with that many 0/1-cases? No. If the poset has two or more segments, Sortnet only has to maintain the 0/1-cases for each segment in a separate linked-list.

4.3 Treating Posets

At the start, the poset has N segments with each segment containing just one element so there are only two 0/1-cases for that segment: one with a 0 in that element and another with a 1 in it. Thus, Sortnet starts with N linked-lists with each list containing only two cases.

4.4 Treating Each Comparator

Sortnet treats the comparators in the CE-list one at a time. To treat comparator C(Lo, Hi), Sortnet checks to see if Lo and Hi are in the same segment of the poset or in two different segments:

- If Lo and Hi are in the same segment of the poset, then Sortnet runs through all 0/1-cases for that segment and whenever it finds a case where Lo = 1 and Hi = 0 it swaps the two values.
- If Lo and Hi are in two different segments of the poset, then Sortnet uses an outer loop and an inner loop to combine every 0/1-case of one segment with every 0/1-case of the other segment. Whenever it finds a combination where Lo = 1 and Hi = 0 it swaps the two values. The two segments are now combined into one segment containing both Lo and Hi.

4.5 Eliminating Duplicate Cases

Hashing is used to eliminate duplicate cases. The new cases generated when a comparator is treated are temporarily stored in P separate linked-lists where P is a prime number. Case C is treated like an integer and stored in linked-list C mod P only if it's not the same as any other case in the same linked-list.

4.6 Example – Treating a 32-Element Network

To illustrate how Sortnet limits the number of cases it handles we use the CE-list depicted in Figure 10 that forms a one segment poset of 32 elements.

```
sortnet-> RD.CE START32

sortnet-> SHOW.CE

/* STEP 1 */

   0  1    2  3    4  5    6  7    8  9   10 11   12 13   14 15
  16 17   18 19   20 21   22 23   24 25   26 27   28 29   30 31

/* STEP 2 */

   0  2    1  3    4  6    5  7    8 10    9 11   12 14   13 15
  16 18   17 19   20 22   21 23   24 26   25 27   28 30   29 31

/* STEP 3 */

   0  4    1  5    2  6    3  7    8 12    9 13   10 14   11 15
  16 20   17 21   18 22   19 23   24 28   25 29   26 30   27 31

/* STEP 4 */

   0  8    1  9    2 10    3 11    4 12    5 13    6 14    7 15
  16 24   17 25   18 26   19 27   20 28   21 29   22 30   23 31

/* STEP 5 */

   0 16    1 17    2 18    3 19    4 20    5 21    6 22    7 23
   8 24    9 25   10 26   11 27   12 28   13 29   14 30   15 31

/* 80 comparators in 5 steps. */

sortnet->
```

Figure 10. A CE-list that forms a one-segment poset of 32 elements in 5 steps.

- Initially each element is in a separate segment so there are only 64 cases instead of $2^{32} = 4{,}294{,}967{,}296$.
- The 16 comparators in Step 1 form 16 segments with three cases in each segment; 48 cases instead of 3^{16} = 43,046,721.
- The 16 comparators in Step 2 form 8 segments with six cases in each segment; 48 cases instead of 6^8 = 1,679,616.
- The 16 comparators in Step 3 form 4 segments with twenty cases in each segment; 80 cases instead of 20^4 = 160,000.
- The 16 comparators in Step 4 form two segments with 168 cases in each segment; 336 cases instead of 168^2 = 28,224.
- The 16 comparators in Step 5 form one segment with 7,581 cases.

5. CONCLUSION AND FUTURE WORK

Sorting networks are cost-effective multistage interconnection networks with sorting capabilities. The fastest practical sorting consume $\theta(N\log^2 N)$ comparisons[2], whereas the optimal ones developed so far are imparctical. Here we

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar

describe Sortnet,a software tool developed by Kenneth Batcher to help build better sorting networks.

This report describes the purpose of developing Sortnet and highlights a subset of its commands. Sortnet has been undergoing several modifications and extensions to improve its ability to help analyze and synthesize sorting networks. It also has already facilitated developing an 11-step solution for sorting18 elements which outperforms the best known 12-step solution for this problem[4]. Experimentaions for building better networks consisting of 12, 24, and 32 elements have been carried out. Sortnet is undergoing a major update to allow it handle numbers of elements gretaer than 32.

6. REFERENCES

[1] Van Voorhis, D. C., Efficient Sorting Networks, Stanford University, CA, USA, Doctoral Thesis, 1972.

[2] K. E.Batcher, “Sorting Networks and their Applications”, *Spring Joint Computer Conference*, AFIPS Proc., vol. 32, 1968, pp. 307-314.

[3] M. Ajtai, J. Komlos, and E. Szemerdi, “Sorting in nlogn Steps”, *Combinatorica*, Vol. 3, 1983, pp. 1-19.

[4] S. Al-Haj Baddar and K. Batcher, "An 11-Step Sorting Network for 18 Elements", Technical Report TR-KSU-CS-2007-06, Department of Computer Science, Kent State University, Kent OH, USA, September 19th 2007.

[5] Knuth, D. E., The Art of Computer Programming: Volume 3: Sorting and Searching, Addison-Wesley Longman, USA, 2nd Edition, 1998, Chapter 5, pp. 225-228.

[6] Rosen, K. H., Discrete Mathematics and its Applications, McGraw-Hill Companies, USA, 5th Edition, 2003, Chapter 7, pp. 520-525.

© 2008 Kenneth E. Batcher and Sherenaz W. Al-Haj Baddar