COLLECTING BUTTERFLIES

by

Geraint Jones
Mary Sheeran

Oxford University Computing Laboratory
Programming Research Group

COLLECTING BUTTERFLIES

by

Geraint Jones
Mary Sheeran

Technical Monograph PRG–91
ISBN 0–902928–69–4

February 1991

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD

Geraint Jones
Oxford University Computing Laboratory
11 Keble Road
Oxford OX1 3QD
England

Mary Sheeran
Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
Scotland

Collecting butterflies

This monograph contains three papers about butterfly circuits. Circuits of this form turn up in many signal processing applications, and networks of the same shape are found in parallel algorithms for many sorts of message-passing computers. Unfortunately their presentation is usually bottom-up and consequently difficult to understand. In these papers we give top-down analyses of such circuits in the style of *Ruby* – a language of relations and higher-order functions in which circuits are represented as relations on the signals which pass between them.

The first paper – *The study of butterflies* – introduces the algebra of the combining forms with which butterfly circuits will be described. It goes on to show that butterfly forms arise naturally when certain sorts of problem are tacked by a divide-and-conquer strategy.

Butterfly circuits are probably most familiar from their application to the implementation of the fast Fourier transform. The second paper – *A fast flutter by the Fourier transform* – takes the recursive equation which describes the divide-and-conquer calculation of the Fourier transform and shows how it can be implemented by butterfly circuits and by various related regular layouts.

The third paper – *Sorts of butterflies* – shows how Ruby is used to describe and analyse permutation and comparator networks. It explains a periodic sorting network that is suitable for implementation on silicon.

ACKNOWLEDGMENTS The presentation of divide and conquer algorithms owes much to several attempts to explain it to colleagues, and in particular to Richard Bird. We are grateful to David Murphy and Lars Rossen for many comments and suggestions.

The study of butterflies

Geraint Jones

and

Mary Sheeran

Butterfly networks arise in many signal processing circuits and in parallel algorithms for many sorts of message-passing computers. This paper attempts to explain why this should be, and what butterfly networks are, using a new and elegant formulation based on a language of relations.

Most of the material covered by this paper has appeared in a less tractable form in earlier papers [16, 17]. The novelty here is in the simplicity and elegance of the presentation, which derives from an appropriate choice of high-level structures. These structures are represented by functions which are used to compose circuits from components, and are chosen to have simple mathematical properties.

This presentation makes it easier to explain how the design comes about, showing that butterflies are natural implementations of divide-and-conquer algorithms. We are then able to go on to explain many of the properties of butterfly networks, and of their implementations.

A language of relations

The important things in Ruby [13] are the structuring functions, and the interesting things to know are encapsulated by the mathematical properties of those functions. Nevertheless we will need to have some idea of what the component parts being composed are. These are the things that model the components of a circuit, or the nodes of a network of computers. You can think of these components as being relations: that is the simplest interpretation of what is happening. You should however keep in mind that this is just one interpretation, and that the important things to watch are the functions that put them together and the algebra of those functions.

The principal way of putting components together is (sequential) composition, which we write $R\,;S$. If you are thinking of relations, composition of relations means

$$x\,(R\,;S)\,z \quad \equiv \quad \exists y.\ x\,R\,y\ \&\ y\,S\,z$$

but the thing to keep at the front of your mind is that it is an associative way of putting circuits together, $(R\,;S)\,;T = R\,;(S\,;T)$.

In particular that means that it will make sense to talk about 'reducing' composition over a finite ordered set of indices, and we write

$$\mathop{;}_{i=1}^{n} R_i \quad = \quad R_1\,;R_2\,;\ldots;R_n$$

at least in the case that $n > 0$, and we write R^n for $;_{i=1}^{n} R$.

The other extreme way of putting components together leaves them entirely unconnected. The parallel composition $[R, S]$ is defined by

$$\langle p, q\rangle\,[R, S]\,\langle t, u\rangle \quad \equiv \quad p\,R\,t\ \&\ q\,S\,u$$

and the thing to keep in mind is that sequential and parallel composition have the property $[P, Q]\,;[R, S] = [(P\,;R), (Q\,;S)]$ which Richard Bird [4] calls the *abides* property: that sequential composition abides with parallel composition.

The *inverse* (some people say more properly the 'converse') of a relation, R^{-1}, is defined by

$$x\,R^{-1}\,y \quad \equiv \quad y\,R\,x$$

and we will write R^{-n} for $(R^{-1})^n$ and so on. Beware of doing arithmetic in the exponent! A relation and its inverse cannot necessarily be cancelled, so $R^p\,;R^{-q}$ need not necessarily be the same as R^{p-q}.

Converse distributes over parallel composition, $[R, S]^{-1} = [R^{-1}, S^{-1}]$, and in a modified sense over sequential composition, for $(R\,;S)^{-1} = S^{-1}\,;R^{-1}$.

Because we will want to be using relations and their converses to translate data from one representation to another, we will find useful the abbreviation $R \setminus S = S^{-1}\,;R\,;S$, read 'the *conjugate* of R by S'.

The *sum* of two relations R and S (their relational sum, or their union) is a relation $R + S$ for which

$$x\,(R + S)\,y \quad \equiv \quad x\,R\,y \vee x\,S\,y$$

Most of the operations introduced so far distribute over sum, so that for example $(R + S)\,;T = (R\,;T) + (S\,;T)$. The exceptions are the operations like repeated composition that are not linear: because

$$\begin{aligned}(R + S)^2 \quad &= \quad (R + S)\,;(R + S)\\ &= \quad (R\,;R) + (R\,;S) + (S\,;R) + (S\,;S)\end{aligned}$$

it is not generally the same as $R^2 + S^2$. Similarly the conjugation $R \setminus S$ is not linear in S, although $(P + Q) \setminus S = (P \setminus S) + (Q \setminus S)$.

We write $R : A \rightarrow B$ to mean that R relates things of type A to things of type B, and by this we mean that $R = A\,;R\,;B$. A type is just an equivalence relation, which is to say that it is a relation A for which $A = A^2 = A^{-1}$ and so $A = A^n$ for all positive and negative n. When we speak of a circuit R, we will have in mind particular domain and range types $R^{\leftarrow}$ and $R^{\rightarrow}$, for which $R : R^{\leftarrow} \rightarrow R^{\rightarrow}$, although we may not make them explicit. Do not think of $R^{\leftarrow}$ as being some function of R, it is just one element of a triple $\langle R^{\leftarrow}, R, R^{\rightarrow}\rangle$ which we misleadingly identify with R, on the grounds that it is usually obvious which $R^{\leftarrow}$ and $R^{\rightarrow}$ is meant. When $R^{\leftarrow} = R^{\rightarrow}$ we will write this as R^0, which is suggestive of $R^0;R^n;R^0 = R^{0+n+0} = R^n$. Such an R we will call *homogeneous*.

On the whole we will only need to talk about the types of lists of a given length: we write n for the type of lists of length n, meaning that $x\; n\; y$ if and only if $x = y$ and has n components. There is a notational trap lurking here, for we will write 2^n for the type of lists of length two-to-the-n: it should not be read as the n-times repeated sequential composition of 2. Since 2 is a type, the latter is just 2 and we will never need to write it.

A sum $R + S$ is *disjoint* if $R^{\leftarrow}\,;S^{\leftarrow} = \emptyset = R^{\rightarrow}\,;S^{\rightarrow}$ where $\emptyset$ is the unit of relational sum. In that case both $R^{\leftarrow} + S^{\leftarrow}$ and $R^{\rightarrow} + S^{\rightarrow}$ are types as you can check by calculation. (The sum of two types is not in general a type.) Moreover, since $R + S : R^{\leftarrow} + S^{\leftarrow} \rightarrow R^{\rightarrow} + S^{\rightarrow}$, repeated composition distributes over disjoint sum.

Sum is associative, commutative and idempotent, so we can write $\sum_i$ for the continued sum over any set of indices.

Transposing and shuffling

Most of this paper turns out to be about certain sorts of permutations: those that can be understood in terms of transposition operators. The transposition relation *trn* relates two 'rectangular' lists of lists, in such a way that

$$x\; trn\; y \quad \equiv \quad x_{i,j} = y_{j,i}$$

You can think of it as taking a row-of-columns enumeration of a two-dimensional array and turning it into a column-of-rows enumeration.

The easiest way of describing the relation *halve* is to say that its inverse $halve^{-1}$ relates a pair of equal-length lists to the even-length list obtained by concatenating them

$$\langle x_0, x_1, \ldots x_{2n-1}\rangle \; halve \; \langle\langle x_0, x_1, \ldots x_{n-1}\rangle, \langle x_n, x_{n+1}, \ldots x_{2n-1}\rangle\rangle$$

and similarly the relation *pair* is the converse of $pair^{-1}$ which relates a list of pairs

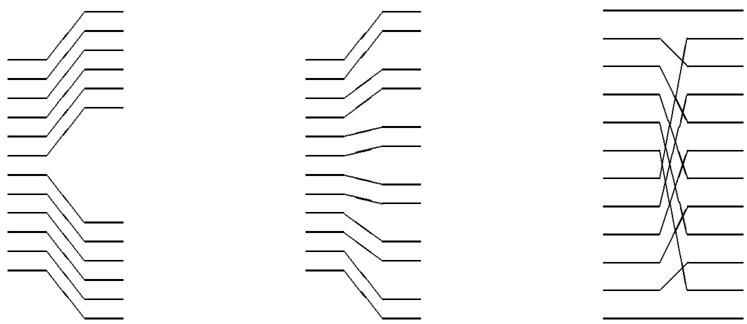

Figure 1: layouts for $12; halve$, $12; pair$, and $12; riffle = (12; halve); trn; (12; pair)^{-1}$

to the even-length list obtained by concatenating the pairs

$$\langle x_0, x_1, \ldots x_{2n-1}\rangle \; pair \; \langle\langle x_0, x_1\rangle, \langle x_2, x_3\rangle, \ldots \langle x_{2n-2}, x_{2n-1}\rangle\rangle$$

The reason we need *halve* and *pair* is to define

$$riffle \quad \equiv \quad halve \; ; trn \; ; pair^{-1}$$

which is a permutation of even-length lists. Think of the professional card-player's shuffling of a pack: the pack is divided in two, *halve*; the corners of the two half-packs are flicked together to interleave them, *trn*; and then the pack is straightened up to give the same status to cards from either half-pack, $pair^{-1}$. This 'riffling' operation is sometimes called a 'perfect shuffle'. It is harder to give a convincing account of how to unriffle a deck of cards, as described by $riffle^{-1}$!

Sometimes we will need to know how wide a list is being permuted, particularly because n successive rifflings of a list of length 2^n will restore it to its original order, which is to say that

$$2^n \; ; riffle^n \quad = \quad 2^n$$

so that

$$2^n \; ; riffle^{n-i} \quad = \quad 2^n \; ; riffle^{-i}$$

Note that this is not directly related to an almost useless fact which any card-sharp will know, that $52 \; ; riffle^8 = 52$.

A language of homogeneous relations

Suppose R is a length-homogeneous circuit, that is one which relates lists of signals only when they have the same length, so that $n \; ; R = R \; ; n$. One way of making

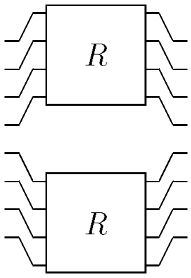

Figure 2: an interpretation of $\mathsf{two}\, R$ as a circuit arrangement

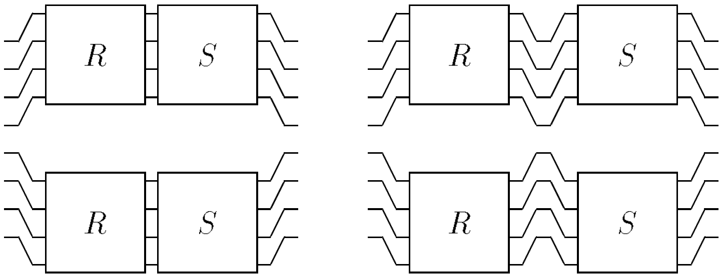

Figure 3: two circuit forms suggesting $\mathsf{two}(R\,;S) = \mathsf{two}\,R\,;\mathsf{two}\,S$

a bigger length-homogeneous circuit is to take two copies of R, and to divide the inputs and outputs of the new circuit equally between the two copies.

$$\mathsf{two}\,R \quad = \quad [R, R] \setminus \mathit{halve}^{-1}$$

So long as we confine ourselves to length-homogeneous relations, two distributes over composition, meaning that

$$\mathsf{two}(R\,;S) \quad = \quad \mathsf{two}\,R\,;\mathsf{two}\,S$$

and so $\mathsf{two}\,R^n = (\mathsf{two}\,R)^n$.

The restriction to length-homogeneous circuits is necessary. Consider the relation R which relates $\langle * \rangle$ to both $\langle * \rangle$ and $\langle *, * \rangle$, and the relation S which relates $\langle * \rangle$ to $\langle a \rangle$ and $\langle *, * \rangle$ to $\langle b \rangle$. Then $R\,;S$ relates $\langle * \rangle$ to both $\langle a \rangle$ and $\langle b \rangle$, so $\mathsf{two}(R\,;S)$ relates $\langle *, * \rangle$ to all four of $\langle a, a \rangle$, $\langle b, a \rangle$, $\langle a, b \rangle$ and $\langle b, b \rangle$. However $\mathsf{two}\,R$ relates $\langle *, * \rangle$ only to $\langle *, * \rangle$ and $\langle *, *, *, * \rangle$, and $\mathsf{two}\,S$ relates each of these to $\langle a, a \rangle$ and $\langle b, b \rangle$, so $\mathsf{two}\,R\,;\mathsf{two}\,S$ is a strictly smaller relation than $\mathsf{two}(R\,;S)$.

A different way of making a length-homogeneous circuit from two components of half its size is suggested by figure 4. The *interleaving* of two components is defined by

$$\mathsf{ilv}\,R \quad = \quad (\mathsf{two}\,R) \setminus \mathit{riffle}$$

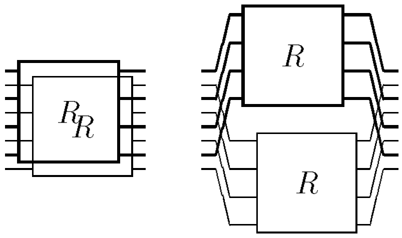

Figure 4: two different interpretations of $\mathsf{ilv}\ R = \mathsf{two}\ R \setminus \mathit{riffle}$

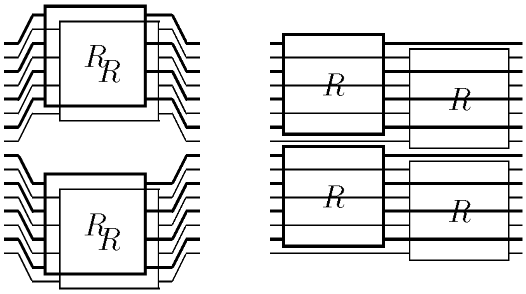

Figure 5: two views of $\mathsf{two}\,\mathsf{ilv}\ R = \mathsf{ilv}\,\mathsf{two}\ R$

and – following from the distribution result for two – if R and S are both length-homogeneous then

$$\mathsf{ilv}(R\,;S) \quad = \quad \mathsf{ilv}\,R\,;\mathsf{ilv}\,S$$

What may be more surprising is that applications of two and ilv commute, for

$$\mathsf{two}\,\mathsf{ilv}\,R \quad = \quad \mathsf{ilv}\,\mathsf{two}\,R$$

This means, by an induction on the number of constructors, that any term consisting of applications of two and ilv to a relation is determined solely by the number of applications of two and the number of applications of ilv, and that the order in which they are applied is immaterial.

The meaning of the equality is suggested by figure 5, and the proof goes something like this

$$\begin{aligned} & \mathit{halve}\,;[\mathit{riffle}^{-1}\,;\mathit{halve}, \mathit{riffle}^{-1}\,;\mathit{halve}] \\ & \quad = \quad \mathit{halve}\,;[\mathit{pair}\,;\mathit{trn}, \mathit{pair}\,;\mathit{trn}] \\ & \quad = \quad \mathit{pair}\,;\mathit{halve}\,;[\mathit{trn}, \mathit{trn}] \\ & \quad = \quad \mathit{pair}\,;\mathit{trn}\,;[\mathit{halve}, \mathit{halve}]\,;\mathit{trn} \\ & \quad = \quad \mathit{riffle}^{-1}\,;\mathit{halve}\,;[\mathit{halve}, \mathit{halve}]\,;\mathit{trn} \end{aligned}$$

so

$$\begin{aligned} \mathsf{two}\,\mathsf{ilv}\,R \quad & = \{\text{ definitions of two and ilv and collecting terms }\} \\ & \quad [[R,R],[R,R]] \setminus ([\mathit{halve}^{-1}\,;\mathit{riffle}, \mathit{halve}^{-1}\,;\mathit{riffle}]\,;\mathit{halve}^{-1}) \end{aligned}$$

$$\begin{aligned}
&= \{\,\text{calculation above, taking inverses on both sides}\,\}\\
&\quad ([[R,R],[R,R]] \setminus \mathit{trn}) \setminus ([\mathit{halve}^{-1}, \mathit{halve}^{-1}]\,;\, \mathit{halve}^{-1}\,;\, \mathit{riffle})\\
&= \{\, [[A,B],[C,D]] \setminus \mathit{trn} = [[A,C],[B,D]] \,\}\\
&\quad [[R,R],[R,R]] \setminus ([\mathit{halve}^{-1}, \mathit{halve}^{-1}]\,;\, \mathit{halve}^{-1}\,;\, \mathit{riffle})\\
&= \{\,\text{collecting terms and replacing definitions}\,\}\\
&\quad \mathsf{ilv}\ \mathsf{two}\ R
\end{aligned}$$

The details are tedious, but we need never see them again: just remember that $\mathsf{two}\ \mathsf{ilv}\ R = \mathsf{ilv}\ \mathsf{two}\ R$.

Divide and conquer algorithms

Suppose you want to solve some problem by a binary divide and conquer strategy: that is, you know how to solve (conquer) some problems by an algorithm C, and you have a technique D for dividing up any problem that is too big to be dealt with by C. A problem divided has then become two smaller problems that can be tackled in the same way. The algorithm is a solution Φ to

$$\Phi \quad = \quad C + (D\,;\, \mathsf{two}\,\Phi)$$

You can read this as an equation in which the unknown is a relation, and in which the + sign means relational sum (union). The solution can be found by unwinding the recursion:

$$\begin{aligned}
\Phi \quad &= \quad C + D\,;\, \mathsf{two}\,\Phi\\
&= \quad C + D\,;\, \mathsf{two}\,C + D\,;\, \mathsf{two}\,D\,;\, \mathsf{two}^2\,\Phi\\
&= \quad C + D\,;\, \mathsf{two}\,C + D\,;\, \mathsf{two}\,D\,;\, \mathsf{two}^2\,C + D\,;\, \mathsf{two}\,D\,;\, \mathsf{two}^2\,D\,;\, \mathsf{two}^3\,\Phi\\
&= \quad \sum_{i=0}^{n} \Big(\mathop{;}_{j=0}^{i-1} \mathsf{two}^j\,D \Big)\,;\, \mathsf{two}^i\,C + \Big(\mathop{;}_{j=0}^{n} \mathsf{two}^j\,D \Big)\,;\, \mathsf{two}^{n+1}\,\Phi
\end{aligned}$$

and because (at least if there are no empty lists in the range of Φ) the range of $\mathsf{two}^i\,\Phi$ contains only lists of length at least 2^i long, this unfolding eventually defines Φ, by

$$\Phi \quad = \quad \sum_{i=0}^{\infty} \Big(\mathop{;}_{j=0}^{i-1} \mathsf{two}^j\,D \Big)\,;\, \mathsf{two}^i\,C$$

We will suppose that C and D are length-homogeneous, and that $C : k \rightarrow k$ for some small number k. There is no harm in supposing that we can only conquer small problems: that is of the essence of how divide-and-conquer works. Of course there remains the problem of how to divide very large problems.

Suppose that D can itself be implemented by divide-and-conquer, and that $D = R + S \,;\, \mathsf{two}\, D$. If we are to make progress S had better be simple: we could assume that S was the identity relation. In that case $D = \sum_i \mathsf{two}^i\, R$ and if $R : k \to k$ as well as C, it follows that $\Phi = \sum_i (\mathsf{two}^i\, R)^i \,;\, \mathsf{two}^i\, C = \sum_i \mathsf{two}^i (R^i \,;\, C)$. This is not very interesting, because it says that Φ can be applied to a list of a give size just by allocating each k-wide piece to a calculation independent of all the others.

Butterflies arise in the case where large division problems can be tackled by *interleaving* smaller division algorithms, for suppose that $D = R + \mathsf{ilv}\, D$, then under the same assumptions

$$\Phi \;=\; \sum_{i=0}^{\infty} \Big(\mathop{;}_{j=0}^{i-1} \mathsf{two}^j\, \mathsf{ilv}^{i-j}\, R \Big) \,;\, \mathsf{two}^i\, C$$

and if $R = C$

$$\Phi \;=\; \sum_{i=0}^{\infty} \bowtie_i R$$

$$\text{where } \bowtie_i R = \mathop{;}_{j=0}^{i} \mathsf{two}^j\, \mathsf{ilv}^{i-j}\, R$$

The right-hand side of this definition suggests a way of laying out the circuit which is illustrated in figure 8 for the case of $\bowtie_3 R$ where $R : 2 \to 2$. We define the butterfly of R by the sum

$$\bowtie R \;=\; \sum_{i=0}^{\infty} \bowtie_i R$$

The sum is disjoint, at least if $R^0 = k$ for some fixed number k, an assumption which we make in what follows.

(If you are comparing this paper with the discussion of butterflies in reference [16], notice that in that paper the definition is slightly different, being $\bowtie R = 1 + \sum_{i=0}^{\infty} \bowtie_i R$. The difference is unimportant, and only slightly alters the discussion in the following section.)

Recursive decomposition of butterflies

Because we arrived at the butterfly by solving a recursion equation, it comes as no surprise that it has a recursive decomposition. There are however a great number of other decompositions. Suppose p and q are at least zero, then

$$\begin{aligned} \bowtie_{p+q+1} R \;&=\; \mathop{;}_{i=0}^{p+q+1} \mathsf{two}^i\, \mathsf{ilv}^{(p+q+1)-i}\, R \\ &=\; \mathop{;}_{i=0}^{p} \mathsf{two}^i\, \mathsf{ilv}^{p-i}\, \mathsf{ilv}^{q+1}\, R \;;\; \mathop{;}_{i=0}^{q} \mathsf{two}^{p+1}\, \mathsf{two}^i\, \mathsf{ilv}^{q-i}\, R \end{aligned}$$

$$\begin{aligned} &= \mathop{;}_{i=0}^{p} \mathsf{two}^i\,\mathsf{ilv}^{p-i}(\mathsf{ilv}^{q+1}\,R)\ ;\ \mathop{;}_{i=0}^{q} \mathsf{two}^i\,\mathsf{ilv}^{q-i}(\mathsf{two}^{p+1}\,R) \\ &= \bowtie_p \mathsf{ilv}^{q+1}\,R\ ;\ \bowtie_q \mathsf{two}^{p+1}\,R \qquad (1) \end{aligned}$$

and

$$\bowtie_{p+q+1} R \;=\; \mathsf{ilv}^{q+1} \bowtie_p R\ ;\ \mathsf{two}^{p+1} \bowtie_q R \qquad (2)$$

In particular, by taking one or other of p and q to be zero in each of equations 1 and 2, it follows that

$$\begin{aligned} \bowtie_{n+1} R &= \mathsf{ilv}^{n+1}\,R\,;\bowtie_n \mathsf{two}\,R && (3)\\ &= \mathsf{ilv}^{n+1}\,R\,;\mathsf{two}\bowtie_n R && (4)\\ &= \bowtie_n \mathsf{ilv}\,R\,;\mathsf{two}^{n+1}\,R && (5)\\ &= \mathsf{ilv}\bowtie_n R\,;\mathsf{two}^{n+1}\,R && (6) \end{aligned}$$

each of which suggests a layout for the implementation. The four decompositions of $\bowtie_3 R$, for a component $R : 2 \to 2$ that takes pairs to pairs, are illustrated in figures 6 to 9.

Results about the general $\bowtie$ follow from taking sums on both sides of each of these equations, for example from equation 3

$$\begin{aligned} \bowtie R &= \bowtie_0 R \;+\; \sum_{i=1}^{\infty} \bowtie_i R \\ &= R \;+\; \sum_{i=0}^{\infty}(\mathsf{ilv}^{i+1}\,R\,;\bowtie_i \mathsf{two}\,R) \\ &= \mathsf{ilv}^0\,R\,;R^0 \;+\; (\sum_{i=0}^{\infty}\mathsf{ilv}^{i+1}\,R)\,;(\sum_{i=0}^{\infty}\bowtie_i \mathsf{two}\,R) \\ &= \mathsf{ilv}^0\,R\,;R^0 \;+\; (\sum_{i=1}^{\infty}\mathsf{ilv}^{i}\,R)\,;\bowtie \mathsf{two}\,R \\ &= (\sum_{i=0}^{\infty}\mathsf{ilv}^{i}\,R)\ ;\ (R^0 + \bowtie \mathsf{two}\,R) \end{aligned}$$

because the various cross-terms are empty and so disappear from the sums. In the same way it can be shown that

$$\bowtie R \;=\; (R^0 + \bowtie \mathsf{ilv}\,R)\ ;\ \sum_{i=0}^{\infty}\mathsf{two}^i\,R$$

an so on.

Shuffle networks

Although the recursive decompositions of butterflies are elegant and easy to reason about, when it comes to laying out circuits they have the disadvantage of having

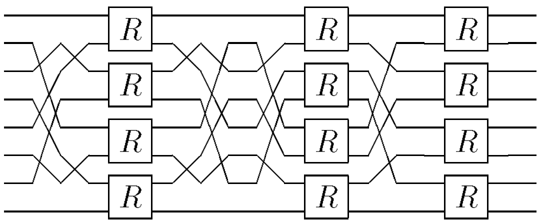

Figure 6: $\bowtie_2 R = \mathsf{ilv}^2\, R\,;\, \bowtie_1 \mathsf{two}\, R = \mathsf{ilv}^2\, R\,;\, \mathsf{ilv}\ \mathsf{two}\, R\,;\, \bowtie_0 \mathsf{two}^2\, R$

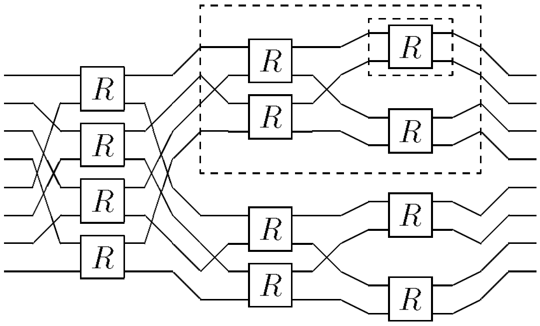

Figure 7: $\bowtie_2 R = \mathsf{ilv}^2\, R\,;\, \mathsf{two} \bowtie_1 R = \mathsf{ilv}^2\, R\,;\, \mathsf{two}(\mathsf{ilv}^1\, R\,;\, \mathsf{two} \bowtie_0 R)$

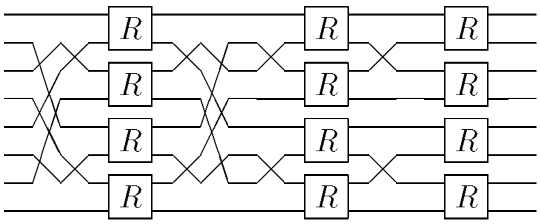

Figure 8: $\bowtie_2 R = \bowtie_1 \mathsf{ilv}\, R\,;\, \mathsf{two}^2\, R = \bowtie_0 \mathsf{ilv}^2\, R\,;\, \mathsf{two}\ \mathsf{ilv}\, R\,;\, \mathsf{two}^2\, R$

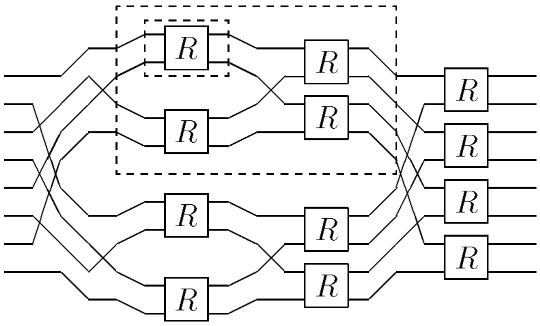

Figure 9: $\bowtie_2 R = \mathsf{ilv} \bowtie_1 R\,;\, \mathsf{two}^2\, R = \mathsf{ilv}(\mathsf{ilv} \bowtie_0 R\,;\, \mathsf{two}\, R)\,;\, \mathsf{two}^2\, R$

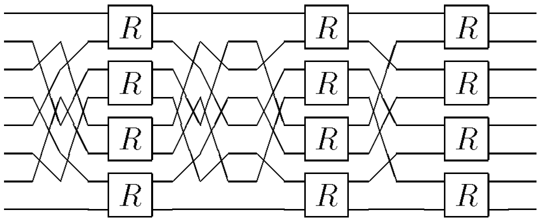

Figure 10: $\bowtie_2 R = \mathop{;}_{i=0}^{2} \mathsf{two}^2\, R \setminus \mathit{riffle}^{2-i}$

differently shaped wiring in different places. Even if the R components can be replicated and laid out in a regular way, each column of wiring is different and there is an amount of work about 16^n involved in laying out the differently shaped parts of it.

Recall that because $\mathsf{two}\,\mathsf{ilv}\, R = \mathsf{ilv}\,\mathsf{two}\, R$, the only thing that matters in a term like $\mathsf{two}^p\, \mathsf{ilv}^q\, R$, or the equivalent $\mathsf{ilv}^q\, \mathsf{two}^p\, R$, is the number of applications of ilv and two. This is encapsulated in the equality

$$\mathsf{two}^p\, \mathsf{ilv}^q\, R \quad = \quad (\mathsf{two}^{p+q}\, R) \setminus \mathit{riffle}^q$$

which can be proved by an induction on q. The case of $q = 0$ is easy, and

$$\begin{aligned}
\mathsf{two}^p\, \mathsf{ilv}^{q+1}\, R \quad &= \{\text{ commuting terms }\} \\
&\ \ \mathsf{ilv}\, \mathsf{two}^p\, \mathsf{ilv}^q\, R \\
&= \{\text{ definition of } \mathsf{ilv} \text{ }\} \\
&\ \ (\mathsf{two}\, \mathsf{two}^p\, \mathsf{ilv}^q\, R) \setminus \mathit{riffle} \\
&= \{\text{ commuting terms }\} \\
&\ \ (\mathsf{two}^p\, \mathsf{ilv}^q\, \mathsf{two}\, R) \setminus \mathit{riffle} \\
&= \{\text{ inductive hypothesis }\} \\
&\ \ (\mathsf{two}^{p+q}\, \mathsf{two}\, R) \setminus (\mathit{riffle}^q \,;\, \mathit{riffle}) \\
&= \ \ \mathsf{two}^{p+q+1}\, R \setminus \mathit{riffle}^{q+1}
\end{aligned}$$

This now suggests that the composition of terms that make up a butterfly has an expression in terms of *riffle* and $\mathsf{two}^n\, R$.

$$\begin{aligned}
\bowtie_n R \quad &= \quad \mathop{;}_{i=0}^{n} \mathsf{two}^i\, \mathsf{ilv}^{n-i}\, R \\
&= \quad \mathop{;}_{i=0}^{n} (\mathsf{two}^n\, R \setminus \mathit{riffle}^{n-i}) \\
&= \quad \mathop{;}_{i=0}^{n} (\mathit{riffle}^{-(n-i)} \,;\, \mathsf{two}^n\, R \,;\, \mathit{riffle}^{n-i})
\end{aligned}$$

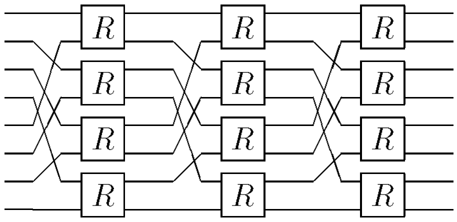

Figure 11: $\mathit{riffle}^3 \,; \bowtie_2 R = (\mathit{riffle} \,; \mathsf{two}^2\, R)^3$

in which the columns of Rs are all the same, but the wiring between them, as illustrated in figure 10, is different for each column and unnecessarily complex.

By the associativity of sequential composition one of the three parts of each column can be carried forward to the next, and

$$\begin{aligned}
\mathit{riffle}^{(n+1)} \,; \bowtie_n R &= \mathit{riffle}^{(n+1)} \;;\; \mathop{;}_{i=0}^{n} (\mathit{riffle}^{-(n-i)} \,; \mathsf{two}^n\, R \,; \mathit{riffle}^{n-i}) \\
&= \mathop{;}_{i=0}^{n} (\mathit{riffle}^{n+1-i} \,; \mathit{riffle}^{-(n-i)} \,; \mathsf{two}^n\, R) \;;\; \mathit{riffle}^0 \\
&= \mathop{;}_{i=0}^{n} (\mathit{riffle} \,; \mathsf{two}^n\, R) \\
&= (\mathit{riffle} \,; \mathsf{two}^n\, R)^{n+1}
\end{aligned}$$

in which each column is the same, and each is wired in the same way to its neighbours, as illustrated in figure 11. This arrangement of components is commonly known as a 'shuffle network'.

Since if $R : k \to k$, any term like $\mathsf{two}^i\, \mathsf{ilv}^j\, R$ has width $2^{i+j}k$, and in case $k = 2$, it is immediate from its definition that $\bowtie_n R : 2^{n+1} \to 2^{n+1}$, and the riffle^{n+1} on the left-hand side can be cancelled yielding

$$\bowtie_n R \;=\; (\mathit{riffle} \,; \mathsf{two}^n\, R)^{n+1}$$

Although there is still a great number of wire crossings in the resulting circuit – about 4^n in each of the $n+1$ columns – it has the advantage that each column is the same as all of the others, so only one column's worth of the circuit need be laid out and replicated.

By a symmetrical argument, it is also true that

$$\bowtie_n R \;=\; (\mathsf{ilv}^n\, R \,; \mathit{riffle})^{n+1}$$

A fast flutter by the Fourier transform

Geraint Jones

This paper explains some familiar but intricate circuit forms that are used to implement the fast Fourier transform. They are shown to be solutions to a recursion equation that defines the transform. An earlier paper [12] showed that the essence of the fast Fourier transform is captured by an equation characteristic of divide-and-conquer algorithms. Butterfly circuits have been shown [14] to be solutions to such equations, and in this paper solutions are derived to the particular equation defining the fast Fourier transform.

Introduction

Twenty-five years ago Cooley and Tukey rediscovered an optimising technique usually attributed to Gauss, who used it in hand calculation. They applied the technique to the discrete Fourier transform, reducing an apparently $O(n^2)$ problem to the almost instantly ubiquitous $O(n \log n)$ 'fast Fourier transform' [7]. The fast Fourier transform is not of course a different transform, but a fast implementation of the discrete transform.

Its greatest virtue lies in that it can be executed in $O(\log n)$ time on $O(n)$ processors in a uniform way – which is to say that it lends itself to a low-latency high-throughput pipelined hardware implementation. Indeed, a footnote to the Cooley–Tukey paper records that a hardware implementation was underway as the paper was published, specifically that a component for evaluating a four-point transform had been 'designed by R. E. Miller and S. Winograd of the IBM Watson Research Centre'.

The unfortunate disadvantage of the fast algorithm is that although the fundamental idea is simple, the detail of its efficient implementation is very hard to understand. That efficiency depends on intricate permutations which rearrange data to maximise the sharing of work done in calculating intermediate results.

Presentations of the algorithm abound in mysterious artefacts like the reversal of bits in subscripts [1], and the translation of parts of subscripts from time space to frequency space [18]. More recent descriptions of implementations seem to gloss over the problem, either referring the reader back to older presentations [21], or apparently assuming that the algorithm – because it is well known – must be well understood [6].

An earlier paper [12] reports the derivation of the Cooley–Tukey fast Fourier algorithm from the specification of the discrete Fourier transform. A functional programming notation was used to express the discrete transform, and an equation describing the fast algorithm calculated from it. That recursion equation shows that the 'fast transform' is an application of a divide-and-conquer strategy. In this paper we take the derivation further by finding a solution to the recursion equation, a solution which is the well-known butterfly circuit.

The discrete Fourier transform

The discrete Fourier transform is defined in terms of the arithmetic on an integral domain. You can think of arithmetic on complex numbers, for a definite example, although there are applications where finite fields or vector spaces over integral domains are appropriate. The derivation depends only on the algebraic properties of the arithmetic, not on the underlying arithmetic itself, so everything said here about the algorithm will be true for finite fields and vector spaces as well.

The discrete Fourier transform of a vector x of length n is a vector y of the same length for which

$$y_j \quad = \quad \sum_{k:0\leq k<n} \omega^{j\times k} \times x_k$$

where ω is a principal n-th root of unity. (In the example of complex numbers, you can think of $\omega = e^{2\pi i/n}$.) The result, y, is sometimes called the 'frequency spectrum' of the sample x.

Even if the powers of ω are pre-calculated, it would appear that $O(n^2)$ multiplications are required to evaluate the whole of y for any x. The fast algorithm avoids many of these by making use of the fact that $\omega^n = 1$. The discovery made by Cooley and Tukey was that if n is composite, the calculation can be divided into what amounts to a number of smaller Fourier transforms. Suppose $n = p \times q$, then by a change of variables

$$\begin{aligned} y_{pa+b} &= \sum_{c:0\leq c<p} \ \sum_{d:0\leq d<q} \omega^{(pa+b)(qc+d)} x_{qc+d} \\ &= \sum_{c:0\leq c<p} \ \sum_{d:0\leq d<q} (\omega^{pq})^{ac}(\omega^{p})^{ad}(\omega^{q})^{bc}\omega^{bd} x_{qc+d} \end{aligned}$$

$$= \sum_{d:0\leq d<q} (\omega^p)^{ad}\omega^{bd} \sum_{c:0\leq c<p} (\omega^q)^{bc} x_{qc+d}$$

Since ω^q is a p-th root of unity, and ω^p is a q-th root of unity, it is not surprising that the above calculation leads to an implementation in which p-sized and q-sized transforms appear.

In particular, if $p = 2$ there is an implementation involving only transforms of size 2 – which are particularly simple – and a pair of transforms of size $n/2$. Repeated division by two permits of an implementation consisting solely of transforms of size two, for any transform which has a width that is a power of two. It is however rather difficult to see from the above calculations what these implementations might be.

In reference [12] the divide-and-conquer strategy is revealed by a calculation in which the expressions are algorithms, rather than data values. For this we will need the notation from a companion paper [14] and a small amount of extra notation specific to this problem.

Triangles

With the constructors introduced in reference [14], any path from the domain to the range has to go through the same number of components. In order to deal with a wider class of circuits we introduce

$$\mathsf{one}\, R \;=\; [id, R] \setminus halve^{-1}$$

where id is the identity relation, the unit of sequential composition. This constructor behaves very like two, for example, remembering that the variables range over only length-homogeneous relations

$$\begin{aligned} \mathsf{one}(R\,;S) &= \mathsf{one}\,R\,;\mathsf{one}\,S \\ \mathsf{ilv}\,\mathsf{one}\,R &= \mathsf{one}\,\mathsf{ilv}\,R \end{aligned}$$

but be careful because $\mathsf{two}\,\mathsf{one}\,R \neq \mathsf{one}\,\mathsf{two}\,R$.

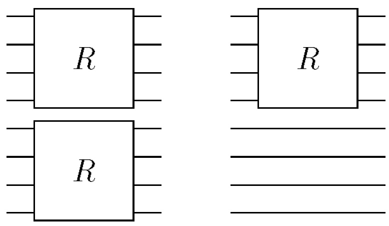

Figure 12: circuit arrangements for $\mathsf{two}\,R$ and $\mathsf{one}\,R$

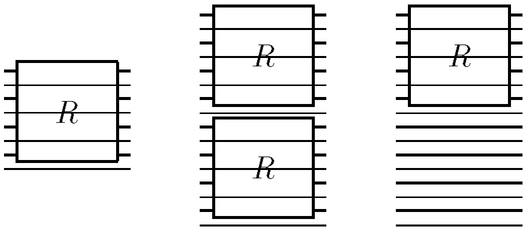

Figure 13: $\mathsf{thw}\ R$, $\mathsf{two}\,\mathsf{thw}\ R = \mathsf{thw}\,\mathsf{two}\ R$, and $\mathsf{one}\,\mathsf{thw}\ R = \mathsf{thw}\,\mathsf{one}\ R$

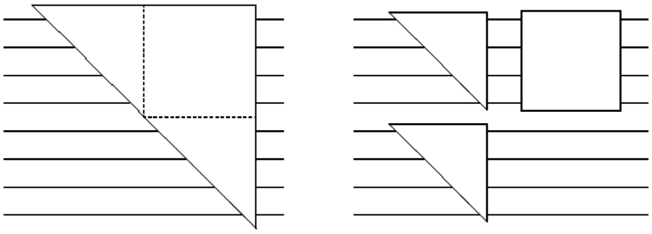

Figure 14: $\mathsf{tri}_{n+1}\ R = \mathsf{two}\,\mathsf{tri}_n\ R\ ;\ \mathsf{one}\,\mathsf{block}_n\ R$

Of course, you can riffle together the two halves of a $\mathsf{one}\ R$. Define

$$\mathsf{thw}\ R \quad = \quad (\mathsf{one}\ R) \setminus \mathit{riffle}$$

for 'through-wire', and it should come as no surprise that

$$\begin{aligned} \mathsf{thw}\,\mathsf{one}\ R &= \mathsf{one}\,\mathsf{thw}\ R \\ \mathsf{thw}\,\mathsf{two}\ R &= \mathsf{two}\,\mathsf{thw}\ R \end{aligned}$$

although in general $\mathsf{thw}\,\mathsf{ilv}\ R \neq \mathsf{ilv}\,\mathsf{thw}\ R$.

There are two families of these constructors, the *straight* ones: one and two, and the *shuffled* ones: ilv and thw. Just as before we were able to say that the only thing that mattered in a term made by applying ilv and two was the number of each, so now we can say that the term is determined by the number and order of the straight constructors, and the number and order of the shuffled ones. The order of the constructors matters within a family, but not the way in which the constructors from the two families are interleaved. The shuffled constructors pass through the straight ones like ghosts though walls, but behave quite reasonably with respect to each other.

You can think of $\mathsf{one}\ R$ as a small triangular-shaped circuit, and figure 14 suggests that larger triangular-shaped circuits can be made by a recursion similar to that for butterflies.

$$\begin{aligned} \mathsf{tri}_{n+1}\ R &= \mathsf{two}\,\mathsf{tri}_n\ R\ ;\ \mathsf{one}\,\mathsf{block}_n\ R \\ &= \mathsf{one}\,\mathsf{block}_n\ R\ ;\ \mathsf{two}\,\mathsf{tri}_n\ R \\ &\qquad \text{where } \mathsf{block}_n = \mathsf{two}^n\ R^{2^n} \end{aligned}$$

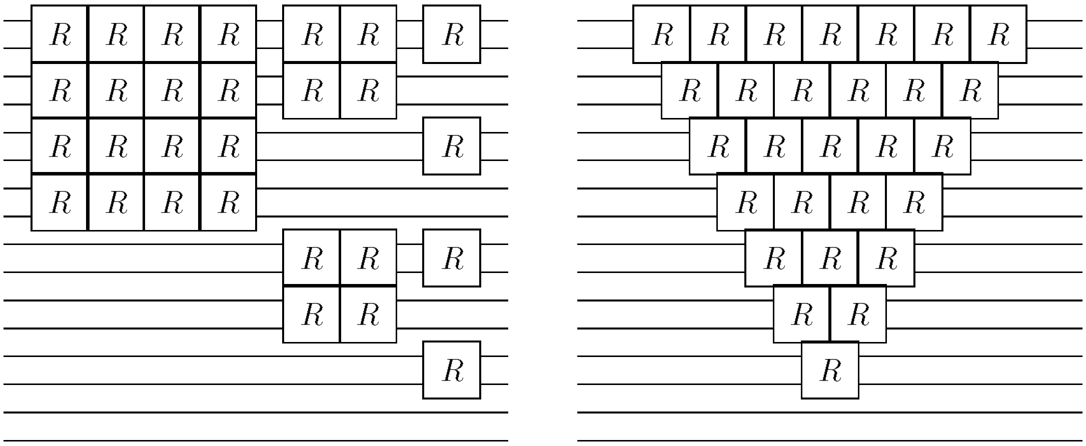

Figure 15: $\mathsf{tri}_3\, R = \mathsf{one}\,\mathsf{two}^2\, R^4 \,;\, \mathsf{two}\,\mathsf{one}\,\mathsf{two}\, R^2 \,;\, \mathsf{two}^2\,\mathsf{one}\, R$

where this time $\mathsf{tri}_0\, R = id$ is the identity relation, $\mathsf{tri}_1\, R = \mathsf{one}\, R$, and so on. You can define a tri of general width and depth by

$$\mathsf{tri}\, R \;=\; \sum_{i=0}^{\infty} \mathsf{tri}_i\, R$$

which is again a disjoint sum in case R has a fixed width.

An iterative solution to the recursion for triangle is given by

$$\mathsf{tri}_n\, R \;=\; \mathop{\large ;}_{i=1}^{n} \mathsf{two}^{i-1}\, \mathsf{one}\, \mathsf{two}^{n-i}\, R^{2^{n-i}}$$

and a layout suggested by this equation is shown in figure 15. Because each of the constructors in a triangle is straight, it follows that $\mathsf{ilv}\,\mathsf{tri}\, R = \mathsf{tri}\,\mathsf{ilv}\, R$ so triangle itself has straight properties. The proof goes like

$$\begin{aligned}
\mathsf{ilv}\,\mathsf{tri}\, R &= \mathsf{ilv} \sum_{i=0}^{\infty} \mathop{\large ;}_{j=1}^{i} \mathsf{two}^{j-1}\, \mathsf{one}\, \mathsf{two}^{i-j}\, R^{2^{i-j}} \\
&= \sum_{i=0}^{\infty} \mathop{\large ;}_{j=1}^{i} \mathsf{two}^{j-1}\, \mathsf{one}\, \mathsf{two}^{i-j}\, \mathsf{ilv}\, R^{2^{i-j}} \\
&= \sum_{i=0}^{\infty} \mathop{\large ;}_{j=1}^{i} \mathsf{two}^{j-1}\, \mathsf{one}\, \mathsf{two}^{i-j} (\mathsf{ilv}\, R)^{2^{i-j}} \\
&= \mathsf{tri}\,\mathsf{ilv}\, R
\end{aligned}$$

and similarly $\mathsf{thw}\,\mathsf{tri}\, R = \mathsf{tri}\,\mathsf{thw}\, R$.

(If you are comparing this paper with earlier presentations such as that in reference [13], beware that this is not quite the same definition of triangle: that paper defines a triangular constructor which assumes that the component is $R : 1 \to 1$.)

The fast Fourier transform

At the end of reference [12] it is suggested that, at least for certain factorisations, the algorithm admits of an implementation which is like a butterfly network. The substance of that claim can now be explained. In the reference it is eventually shown that the transform of size $2n$ can be implemented by two calculations of size n by the algorithm

$$\mathcal{F}_f \,;\, 2n \;=\; \mathit{riffle} \,;\, \mathsf{two}^n(\mathcal{F}_{f^n} \,;\, 2) \,;\, \mathit{riffle}^{-1} \,;\, \mathsf{tri}_1\, \mathsf{tri}_n\, f \,;\, \mathsf{two}(\mathcal{F}_{f^2} \,;\, n) \,;\, \mathit{riffle}$$

where the kernel operation $f : 1 \to 1$, multiplication by a $2n$-th root of unity, is such that f^{2n} is the identity on singletons.

The component $\varphi = \mathcal{F}_{f^n};2$ takes two inputs to two outputs and will be assumed to be directly implementable. The other part, $\mathcal{F}_{f^2} \,;\, n$ is also a Fourier transform because $(f^2)^n$ is also the identity. If n is even the division can be repeated, and in particular if n is a power of two it can be continued until the only $\mathcal{F}$ components are all φ.

Let $\Phi_n = \mathcal{F}_{f_n} \,;\, 2^n$ where for each n the operation f_n is such that ${f_n}^{2^n}$ is the identity, and $f_n = f_{n+1}^2$. Then at least for $n > 1$

$$\begin{aligned}
\Phi_n \;&=\; \mathit{riffle} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathit{riffle}^{-1} \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-1}\, f_{n-1} \,;\, \mathsf{two}\, \Phi_{n-1} \,;\, \mathit{riffle} \\
&= \{\, \mathit{riffle}^n \text{ can be cancelled on } 2^n \,\} \\
&\quad\; \mathit{riffle}^{-(n-1)} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathit{riffle}^{n-1} \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-1}\, f_{n-1} \,;\, \mathsf{two}\, \Phi_{n-1} \,;\, \mathit{riffle} \\
&= \{\, \mathsf{two}\, R \setminus \mathit{riffle} = \mathsf{ilv}\, R \text{ and } \mathsf{two}\, \mathsf{ilv}\, R = \mathsf{ilv}\, \mathsf{two}\, R \text{ and then by induction} \,\} \\
&\quad\; \mathsf{ilv}^{n-1}\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-1}\, f_{n-1} \,;\, \mathsf{two}\, \Phi_{n-1} \,;\, \mathit{riffle} \\
&= \{\, \text{unwinding the recursion, then by induction} \,\} \\
&\quad\; \mathop{;}_{i=1}^{n-1} \mathsf{two}^{i-1}(\mathsf{ilv}^{n-i}\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i}) \;\;;\;\; \mathsf{two}^{n-1}\, \varphi \;\;;\;\; \mathop{;}_{i=2}^{n} \mathsf{two}^{n-i}\, \mathit{riffle}
\end{aligned}$$

The term in the middle can be written, rather perversely, as

$$\mathsf{two}^{n-1}\, \varphi \;=\; \mathsf{two}^{n-1}(\mathsf{ilv}^0\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_0\, f_0) \,;\, \mathsf{two}^{n-1}\, \mathit{riffle}$$

by adding in some extra terms that happen to be identities, so

$$\Phi_n \;=\; \mathcal{B}_n \,;\, \mathcal{S}_n$$

where

$$\begin{aligned}
\mathcal{B}_n \;&=\; \mathop{;}_{i=1}^{n} \mathsf{two}^{i-1}(\mathsf{ilv}^{n-i}\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i}) && (7) \\
\mathcal{S}_n \;&=\; \mathop{;}_{i=1}^{n} \mathsf{two}^{n-i}(2^i \,;\, \mathit{riffle})
\end{aligned}$$

As in the decompositions of the butterfly, the $\mathcal{B}$ and $\mathcal{S}$ terms can be summed separately, since $\mathcal{B}_i \,;\, \mathcal{S}_j$ is empty unless $i = j$. Let $\mathcal{B} = \sum_{i=0}^{\infty} \mathcal{B}_i$ and $\mathcal{S} = \sum_{i=0}^{\infty} \mathcal{S}_i$ then $\Phi = \sum_{i=0}^{\infty} \Phi_i = \mathcal{B} \,;\, \mathcal{S}$. It is normal to implement the required part of $\mathcal{B}$ in a machine, and to leave the corresponding part of $\mathcal{S}$ to the way that the machine is connected to the outside world.

The butterfly

The part of the decomposition of Φ_n that looks like a butterfly circuit is $\mathcal{B}_n$, which is like a butterfly – specifically, like $\bowtie_{n-1} \varphi$ – in which to each column $\mathsf{two}^{i-1}\, \mathsf{ilv}^{n-i}\, \varphi$ has been added a term $\mathsf{two}^{i-1}\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i}$. This is made with only straight constructors and powers of the kernel operation: in implementations it would be turned into a single column of multipliers.

For example, following the development of the shuffle network for a butterfly given in the companion paper [14], there is a shuffle network for the Fourier transform. Each column of $\mathcal{B}_n$ in equation 7 has the form

$$
\begin{aligned}
&\mathsf{two}^{i-1}(\mathsf{ilv}^{n-i}\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i}) \\
&\quad = \quad \mathsf{two}^{i-1}\, \mathsf{ilv}^{n-i}\, \varphi \,;\, \mathsf{two}^{i-1}\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i} \\
&\quad = \{ \text{ unriffling the } \mathsf{ilv}^{n-i}\, \varphi \,\} \\
&\qquad \mathit{riffle}^{-(n-i)} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathit{riffle}^{n-i} \,;\, \mathsf{two}^{i-1}\, \mathsf{one}\, \mathsf{tri}_{n-i}\, f_{n-i} \\
&\quad = \{ \text{ riffling the } \mathsf{two}^{i-1}\, \mathsf{one}\, R \,\} \\
&\qquad \mathit{riffle}^{-(n-i)} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathit{riffle}^{n} \,;\, \mathsf{ilv}^{i-1}\, \mathsf{thw}\, \mathsf{tri}_{n-i}\, f_{n-i} \,;\, \mathit{riffle}^{n-i} \\
&\quad = \{ \text{ } \mathit{riffle}^n \text{ can be cancelled on } 2^n\text{-lists, promoting straight operators} \,\} \\
&\qquad \mathit{riffle}^{-(n-i)} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathsf{tri}_{n-i}\, \mathsf{ilv}^{i-1}\, \mathsf{thw}\, f_{n-i} \,;\, \mathit{riffle}^{n-i}
\end{aligned}
$$

but the term in the triangle

$$
\begin{aligned}
\mathsf{ilv}^{i-1}\, \mathsf{thw}\, f_{n-i} \quad &= \{ \text{ unriffling } \} \\
&\quad (\mathsf{two}^{i-1}\, \mathsf{one}\, f_{n-i}) \setminus \mathit{riffle}^{i} \\
&= \{ \text{ } \mathit{riffle}^{i} \text{ can be cancelled on } 2^i\text{-lists } \} \\
&\quad \mathsf{two}^{i-1}\, \mathsf{one}\, f_{n-i}
\end{aligned}
$$

Re-assembling these columns in equation 7 and cancelling,

$$
\begin{aligned}
\mathcal{B}_n \quad &= \quad \mathop{;}_{i=1}^{n} \mathsf{two}^{i-1}(\mathsf{ilv}^{n-i}\, \varphi \,;\, \mathsf{tri}_1\, \mathsf{tri}_{n-i}\, f_{n-i}) \\
&= \quad \mathop{;}_{i=1}^{n} (\mathit{riffle}^{-(n-i)} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathsf{tri}_{n-i}\, \mathsf{two}^{i-1}\, \mathsf{one}\, f_{n-i} \,;\, \mathit{riffle}^{n-i}) \\
&= \quad \mathit{riffle}^{-n} \,;\, \mathop{;}_{i=1}^{n} (\mathit{riffle} \,;\, \mathsf{two}^{n-1}\, \varphi \,;\, \mathsf{tri}_{n-i}\, \mathsf{two}^{i-1}\, \mathsf{one}\, f_{n-i})
\end{aligned}
$$

Now the term in the triangle is entirely straight, in fact it is

$$\begin{aligned}
&\mathsf{tri}_{n-i}\,\mathsf{two}^{i-1}\,\mathsf{one}\,f_{n-i} \\
&\quad= \mathop{;}_{j=1}^{n-i} \mathsf{two}^{j-1}\,\mathsf{one}\,\mathsf{two}^{n-(i+j)}(\mathsf{two}^{i-1}\,\mathsf{one}\,f_{n-i})^{2^{n-(i+j)}} \\
&\quad= \mathop{;}_{j=1}^{n-i} \mathsf{two}^{j-1}\,\mathsf{one}\,\mathsf{two}^{(n-j)-1}\,\mathsf{one}\,f_j
\end{aligned}$$

so

$$\mathcal{B}_n = \mathop{;}_{i=1}^{n} (\mathit{riffle} ; \mathcal{C}_i)$$

$$\text{where } \mathcal{C}_i = \mathsf{two}^{n-1}\,\varphi ; \mathop{;}_{j=1}^{n-i} \mathsf{two}^{j-1}\,\mathsf{one}\,\mathsf{two}^{(n-j)-1}\,\mathsf{one}\,f_j$$

The column $\mathcal{C}_i$ is a group of 2^{n-1} independent circuits, each of which is $\varphi ; \mathsf{one}\, f^k_{n-i}$ for some k. It would be nice to conclude by showing this, but we have not yet found an elegant and convincing way of doing this within the notation.

The shuffle

Returning to the remaining part of the algorithm, an induction from $\mathsf{two}\,R ; \mathit{riffle} = \mathit{riffle} ; \mathsf{ilv}\,R$ will show that

$$\mathcal{S}_n = 2^n ; \mathop{;}_{i=1}^{n} \mathsf{two}^{n-i}\,\mathit{riffle} = 2^n ; \mathop{;}_{i=0}^{n-1} \mathsf{ilv}^i\,\mathit{riffle}$$

This is just a permutation on lists of length 2^n. It is that very thorough shuffle that appears mysteriously in many presentations of this algorithm: $x\,\mathcal{S}_n\,y$ if and only if x and y are both of length 2^n and $x_i = y_j$ where the (n-bit long) binary representations of i and of j are each the reverse of the other.

It is its own inverse, and is closely related to the butterfly since if $R : 2^k \to 2^k$ then $(\mathsf{ilv}\,R) \setminus \mathcal{S}_{k+1} = \mathsf{two}(R \setminus \mathcal{S}_k)$ and $(\mathsf{two}\,R) \setminus \mathcal{S}_{k+1} = \mathsf{ilv}(R \setminus \mathcal{S}_k)$, and so also $(\bowtie_n(R \setminus \mathcal{S}_k)) \setminus \mathcal{S}_{n+k} = (\bowtie_n(R^{-1}))^{-1}$. Proofs of these, and the discovery of many other pleasant properties are left for the reader's idle moments.

Sorts of butterflies

Mary Sheeran

This paper shows how Ruby is used to describe and analyse permutation and comparator networks. It describes two merging networks, the bitonic merger and the balanced merger, and shows how they are related. Both of these networks can be used to build recursive sorters. The balanced merger is also the building block of a periodic sorting network that is suitable for implementation on silicon. The correctness of this sorter is demonstrated. As always the key to success in understanding a circuit or algorithm is in finding suitable structuring functions and studying their mathematical properties.

Permutation networks

As well as the wiring permutation *riffle*, we will need some other permutations. The basic building blocks are $[id, id]$ and swp where $\langle a, b\rangle\ swp\ \langle b, a\rangle$. The permutation $\mathsf{two}^n\, swp$ swaps adjacent pairs in a list of length 2^{n+1}. For example, $\langle 0,1,2,3,4,5,6,7\rangle$ is related by $\mathsf{two}^2\, swp$ to $\langle 1,0,3,2,5,4,7,6\rangle$. The permutation $\mathsf{ilv}^n\, swp$ switches the two halves of a list so that

$$\mathsf{ilv}^n\, swp \quad = \quad 2^{n+1} \,;\, halve \,;\, swp \,;\, halve^{-1} \tag{8}$$

For example, $\langle 0,1,2,3,4,5,6,7\rangle$ is related by $\mathsf{ilv}^2\, swp$ to $\langle 4,5,6,7,0,1,2,3\rangle$. The relation $\mathsf{ilv}^n\, swp$ commutes with $\mathsf{two}\, R$ for any homogeneous R.

$$\begin{aligned}
\mathsf{ilv}^n\, swp \,;\, \mathsf{two}\, R \;&= \{\text{ equation 8 and definition } \mathsf{two}\ \}\\
&\quad\; 2^{n+1} \,;\, halve \,;\, swp \,;\, halve^{-1} \,;\, halve \,;\, [R,R] \,;\, halve^{-1}\\
&= \{\ halve \,;\, swp \,;\, halve^{-1} \,;\, halve = halve \,;\, swp\ \}\\
&\quad\; 2^{n+1} \,;\, halve \,;\, swp \,;\, [R,R] \,;\, halve^{-1}\\
&= \{\ swp \,;\, [R,R] = [R,R] \,;\, swp^{-1} \text{ and } R \text{ homogeneous }\}\\
&\quad\; halve \,;\, [R,R] \,;\, swp^{-1} \,;\, halve^{-1} \,;\, 2^{n+1}
\end{aligned}$$

$$\begin{aligned} &= \{\text{ reversing the above calculation }\} \\ &\quad (\mathsf{ilv}^n\, swp\, ;\, \mathsf{two}\, R^{-1})^{-1} \\ &= \{\text{ taking inverses, } swp^{-1} = swp \,\} \\ &\quad \mathsf{two}\, R\, ;\, \mathsf{ilv}^n\, swp \end{aligned} \tag{9}$$

For any $R : 2 \to 2$, the relations $\mathsf{two}^n\, R$ and $\mathsf{ilv}^n\, R$ are related by

$$\mathsf{two}^n\, R \quad = \quad (\mathsf{ilv}^n\, R) \setminus \mathit{riffle}$$

since $2^{n+1}\, ;\, \mathit{riffle}^n = 2^{n+1}\, ;\, \mathit{riffle}^{-1}$. So we can take the *riffle* conjugate of each side of equation 9 to get

$$\mathsf{two}^n\, swp\, ;\, \mathsf{ilv}\, R \quad = \quad \mathsf{ilv}\, R\, ;\, \mathsf{two}^n\, swp$$

The relation prm, for 'permute', defined by

$$prm = [id, id] + swp$$

relates a 2-list to each of its two permutations (and vice versa). Since $prm = prm^{-1} = prm^2$, it is the type of unordered 2-lists.

Switching networks can be built from prm. For example, $\mathsf{two}\, prm$ relates a list of length four to each of the four permutations that are obtained by choosing whether or not to swap adjacent pairs. These four possibilities are shown in figure 16. Similarly, $\mathsf{two}^n\, prm$ relates a list of length 2^{n+1} to each of 2^{2^n} permutations since each prm can be either $[id, id]$ or swp. Note that while $\mathsf{two}^n[id, id]$ and $\mathsf{two}^n\, swp$ both commute with $\mathsf{ilv}\, R$ for homogeneous R, $\mathsf{two}^n\, prm$ does not.

The network $\bowtie_n prm$ is an interesting one that has been much studied. For example, it is presented and analysed in reference [3] where it is called the *omega network*. It has $(n+1)2^n$ prm elements each of which has two possible settings.

Comparator networks

A two-input comparator is a permuting element whose range is constrained to be sorted. Let inc_i be the identity on sorted lists of length 2^i and $up = inc_1$ be the identity on sorted two-lists. Then $inc = \sum_{i=0}^{\infty} inc_i$ is the identity on sorted lists. Define

$$cmp = prm\, ;\, up$$

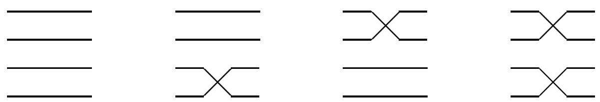

Figure 16: The four permutations realised by $\mathsf{two}\, prm$

Then because *prm* and *up* are both types,

$$prm \,;\, cmp = cmp = cmp \,;\, up$$

so the type of *cmp* is $prm \to up$ which says that it relates an unordered 2-list to an ordered one. Because *up* is strictly smaller than the identity, $cmp < prm$.

The number of pairs in a sequence that are in order ($x_i \leq x_j$ for $i < j$) is a measure of the sortedness of the sequence. The relation $\mathsf{two}^n\ cmp$ increases the sortedness of a sequence by swapping the value at index $2i$ with the value at index $2i + 1$ if necessary. For example, the sequence $\langle 7, 6, 5, 4, 3, 2, 1, 0\rangle$ is related by $\mathsf{two}^2\ cmp$ to $\langle 6, 7, 4, 5, 2, 3, 0, 1\rangle$. If the sequence in the domain of $\mathsf{two}^n\ cmp$ consists of two interleaved sorted sequences, then the related sequence in the range also consists of two interleaved sorted sequences. We write this as

$$\mathsf{ilv}\ inc \,;\, \mathsf{two}^n\ cmp \quad = \quad \mathsf{ilv}\ inc \,;\, \mathsf{two}^n\ cmp \,;\, \mathsf{ilv}\ inc \tag{10}$$

The relation $\mathsf{ilv}\ inc$ is the identity on sequences whose even-numbered elements and odd-numbered elements both form sorted sequences. Here we are using restricted identities as predicates. We will say that a sequence *satisfies* an identity if it is in the domain of the identity. The equation $Pre \,;\, R = Pre \,;\, R \,;\, Post$ says that if an element in the domain of R satisfies Pre then the related element in the range of R satisfies $Post$.

It can be proved that if $k \leq n$

$$\mathsf{ilv}^{k+1}\ inc \,;\, \mathsf{two}^{n+1}\ cmp \quad = \quad \mathsf{ilv}^{k+1}\ inc \,;\, \mathsf{two}^{n+1}\ cmp \,;\, \mathsf{ilv}^{k+1}\ inc \tag{11}$$

from equation 10 and the properties of permutations.

Batcher's bitonic merger

Perhaps the best known comparator network of all is Batcher's bitonic merger. It is a butterfly of comparators and it sorts some but not all sequences. In particular, Batcher notes that $\mathcal{B}_n = \bowtie_n cmp$ sorts any sequence (of length 2^{n+1}) whose two halves are sorted into opposite orders (see references [2, 20]). It sorts many other sequences, but that does not matter. Knowing that it sorts sequences of that particular form gives us the classic recursive bitonic sorter.

The interesting properties of the bitonic merger derive from the fact that it is a butterfly. For example,

$$\begin{aligned} \mathcal{B}_0 &= cmp \\ \mathcal{B}_{n+1} &= \mathsf{ilv}^{n+1}\ \mathcal{B}_0 \,;\, \mathsf{two}\ \mathcal{B}_n \\ &= \mathsf{ilv}\ \mathcal{B}_n \,;\, \mathsf{two}^{n+1}\ \mathcal{B}_0 \end{aligned}$$

These are the two standard recursive decompositions often presented in the literature. The properties of $\bowtie$ give us many more, including

$$\mathcal{B}_{p+q+1} \quad = \quad \mathsf{ilv}^{q+1}\,\mathcal{B}_p \,;\, \mathsf{two}^{p+1}\,\mathcal{B}_q$$

This is the equation that underlies the K-way bitonic sort which is presented in [15]. It is not really a new algorithm, but another way of decomposing an old one.

We can build networks with the same behaviour as $\mathcal{B}_n$ but with a different connection pattern by putting the wiring relation swp in front of selected comparators. This transformation preserves behaviour since $swp \,;\, cmp = cmp$. Replacing every cmp by $swp \,;\, cmp$ turns out to be uninteresting but we can replace the $\mathsf{two}^n\ cmp$ in the rightmost column by $\mathsf{two}^{n-1}\,\mathsf{one}\,swp \,;\, \mathsf{two}^n\ cmp$ since

$$\begin{aligned}
\mathsf{two}^{n-1}\,\mathsf{one}\,swp \,;\, \mathsf{two}^n\ cmp &= \mathsf{two}^{n-1}(\mathsf{one}\,swp \,;\, \mathsf{two}\ cmp)\\
&= \mathsf{two}^{n-1}\,\mathsf{two}\ cmp\\
&= \mathsf{two}^{n}\ cmp
\end{aligned}$$

Abbreviate $\mathsf{two}^{n-1}\,\mathsf{one}\,swp$ to alt_n and let $alt = \sum_{i=1}^{\infty} alt_i$. For $n > 0$

$$\mathcal{B}_n \quad = \quad \mathsf{ilv}\,\mathcal{B}_{n-1} \,;\, alt_n \,;\, \mathsf{two}^n\ cmp$$

We want to move the alt leftwards so that it appears as a wiring relation on the domain. Define a new structuring function vee by

$$\mathsf{vee}\,R \quad = \quad (\mathsf{ilv}\,R) \setminus alt$$

We can compose alt on the left of both sides of this equation to give $alt \,;\, \mathsf{vee}\,R = \mathsf{ilv}\ R \,;\, alt$. Now

$$\mathcal{B}_n \quad = \quad alt_n \,;\, \mathsf{vee}\,\mathcal{B}_{n-1} \,;\, \mathsf{two}^n\ cmp$$

and by induction (using properties of vee that are discussed in the next section)

$$\mathcal{B}_n \quad = \quad (\mathop{\large ;}_{i=0}^{n-1} \mathsf{vee}^i\ alt_{n-i}) \,;\, \mathop{\large ;}_{i=0}^{n} \mathsf{vee}^{n-i}\,\mathsf{two}^i\ cmp$$

We have shown that the bitonic merger can be rewritten as the composition of a wiring permutation $\mathop{;}_{i=0}^{n-1} \mathsf{vee}^i\ alt_{n-i}$ with something that looks very like a butterfly except that it is made with vee instead of with ilv. The butterfly-like thing is the balanced merger proposed in reference [11] as the building block of a periodic sorter.

Networks built using vee

The next step is to study the properties of vee. Assume that R and S are length-homogeneous. Because ilv distributes over composition, so does vee (see figure 17).

$$\mathsf{vee}(R\,;S) = \mathsf{vee}\,R\,;\mathsf{vee}\,S$$

Because $alt_{n+1} = \mathsf{two}\,alt_n$ and ilv commutes with two

$$\mathsf{two}\,\mathsf{vee}\,R = \mathsf{vee}\,\mathsf{two}\,R$$

It is altogether more surprising to find that (for $R : 2^n \to 2^n$)

$$\mathsf{vee}\,\mathsf{ilv}\,R = \mathsf{ilv}\,\mathsf{ilv}\,R$$

Instances of these two equalities are shown in figures 18 and 19, for $R : 2 \to 2$.

If a sequence in the domain of $\mathsf{two}^{n+1}\,cmp$ satisfies $\mathsf{vee}\,inc$ then the related sequence in the range satisfies $\mathsf{ilv}\,inc$ since

$$\begin{array}{ll}
\mathsf{vee}\,inc\,;\mathsf{two}^{n+1}\,cmp & = \{\text{ definition } \mathsf{vee}\,\} \\
& \quad alt^{-1}\,;\mathsf{ilv}\,inc\,;alt\,;\mathsf{two}^{n+1}\,cmp \\
& = \{\ alt\,;\mathsf{two}^{n+1}\,cmp = \mathsf{two}^{n+1}\,cmp\ \} \\
& \quad alt^{-1}\,;\mathsf{ilv}\,inc\,;\mathsf{two}^{n+1}\,cmp \\
& = \{\text{ equation } 10\ \} \\
& \quad alt^{-1}\,;\mathsf{ilv}\,inc\,;\mathsf{two}^{n+1}\,cmp\,;\mathsf{ilv}\,inc \\
& = \{\text{ reversing the steps in the above calculation }\} \\
& \quad \mathsf{vee}\,inc\,;\mathsf{two}^{n+1}\,cmp\,;\mathsf{ilv}\,inc
\end{array} \qquad (12)$$

Each comparator 'operates' on one value from each of the sorted sequences in the domain. An example of a sequence that satisfies $\mathsf{vee}\,inc$ (but not $\mathsf{ilv}\,inc$) is $\langle 0,4,5,1,2,6,7,3\rangle$; one that satisfies $\mathsf{ilv}\,inc$ (but not $\mathsf{vee}\,inc$) is $\langle 0,4,1,5,2,6,3,7\rangle$. These two sequences are related by $\mathsf{two}^2\,cmp$.

We have now proved

$$\mathsf{vee}\,\mathsf{ilv}^k\,inc\,;\mathsf{two}^{n+1}\,cmp \quad = \quad \mathsf{vee}\,\mathsf{ilv}^k\,inc\,;\mathsf{two}^{n+1}\,cmp\,;\mathsf{ilv}^{k+1}\,inc \qquad (13)$$

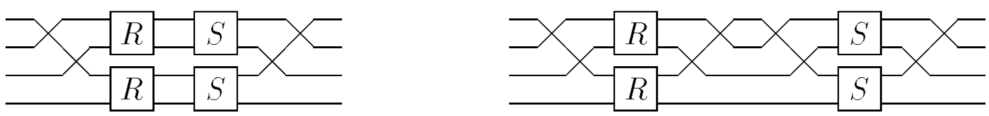

Figure 17: $\mathsf{vee}(R\,;S)$ and $\mathsf{vee}\,R\,;\mathsf{vee}\,S$

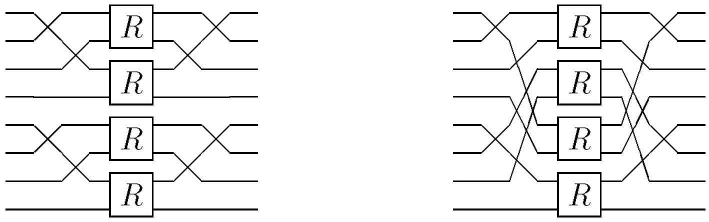

Figure 18: $\mathsf{two}\,\mathsf{vee}\,R$ and $\mathsf{vee}\,\mathsf{two}\,R$

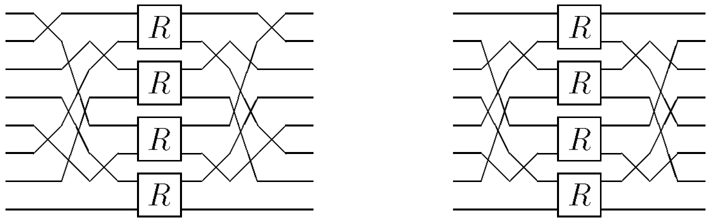

Figure 19: $\mathsf{vee}\,\mathsf{ilv}\,R$ and $\mathsf{ilv}\,\mathsf{ilv}\,R$

because if $k = 0$ it reduces to equation 12, and if $k > 0$, since $\mathsf{vee}\,\mathsf{ilv}\,R = \mathsf{ilv}\,\mathsf{ilv}\,R$, it reduces to equation 11.

Let rev be the relation between each sequence and the corresponding sequence with the same elements in the reverse order. The relation $\mathsf{vee}^n\,swp$ reverses a sequence of length 2^{n+1}

$$\mathsf{vee}^n\,swp = \underline{2^{n+1}}\,;\,rev$$

because it swaps the first and last elements, second and second last, and so on. Similarly, $\mathsf{vee}^n\,cmp$ compares the first and last elements of a sequence, the second and second last, and so on. For example, the sequence $\langle 0,4,1,5,2,6,3,7\rangle$ is related by $\mathsf{vee}^2\,cmp$ to $\langle 0,3,1,2,5,6,4,7\rangle$. For $R : 2 \to 2$, the relations $\mathsf{vee}^n\,R$ and $\mathsf{ilv}^n\,R$ are related by

$$\mathsf{vee}^n\,R \quad = \quad (\mathsf{ilv}^n\,R) \setminus \mathsf{one}\,rev \tag{14}$$

If you want to think about binary representations of indices, then $\mathsf{ilv}\,R$ divides elements of its domain and range (between instances of R) according to the least significant bit of the index, while $\mathsf{two}\,R$ divides according to the most significant bit. Amazingly enough, $\mathsf{vee}\,R$ divides according to the parity of the two least significant bits! It is best to stop thinking about bits as soon as possible.

The butterfly-like structure that arose in the discussion of the bitonic merger is defined by

$$\mathsf{W}_n\,R \quad = \quad \mathop{;}\limits_{i=0}^{n} \mathsf{vee}^{n-i}\,\mathsf{two}^i\,R$$

We read this as 'veefly R'. Because vee is so much like ilv the structure has a great many recursive decompositions like those of the butterfly, including

$$\begin{aligned} \mathbb{W}_{p+q+1}\,R &= \mathbb{W}_p\,\mathsf{vee}^{q+1}\,R\,;\mathbb{W}_q\,\mathsf{two}^{p+1}\,R \\ &= \mathsf{vee}^{q+1}\,\mathbb{W}_p\,R\,;\mathsf{two}^{p+1}\,\mathbb{W}_q\,R \end{aligned}$$

and choosing p or q to be zero,

$$\begin{aligned} \mathbb{W}_{n+1}\,R &= \mathsf{vee}^{n+1}\,R\,;\mathbb{W}_n\,\mathsf{two}\,R \\ &= \mathsf{vee}^{n+1}\,R\,;\mathsf{two}\,\mathbb{W}_n\,R \\ &= \mathbb{W}_n\,\mathsf{vee}\,R\,;\mathsf{two}^{n+1}\,R \\ &= \mathsf{vee}\,\mathbb{W}_n\,R\,;\mathsf{two}^{n+1}\,R \end{aligned}$$

each of which suggests a layout for the network. The four decompositions of $\mathbb{W}_3\,R$ for a component $R : 2 \to 2$ are shown in figures 20 to 23.

The wiring permutation $\mathop{;}_{i=0}^{n-1} \mathsf{vee}^i\,alt_{n-i}$ that arose in the discussion of the bitonic merger is itself the inverse of a veefly.

$$\begin{aligned} \mathop{;}_{i=0}^{n-1} \mathsf{vee}^i\,alt_{n-i} &= \mathop{;}_{i=0}^{n-1} \mathsf{vee}^i\,\mathsf{two}^{n-1-i}\,\mathsf{one}\,swp \\ &= (\mathop{;}_{i=0}^{n-1} \mathsf{vee}^{n-1-i}\,\mathsf{two}^i\,\mathsf{one}\,swp)^{-1} \\ &= (\mathbb{W}_{n-1}\,\mathsf{one}\,swp)^{-1} \end{aligned}$$

It is also a butterfly. It can be shown by induction that

$$\begin{aligned} \mathop{;}_{i=0}^{n-1} \mathsf{vee}^i\,alt_{n-i} &= \mathop{;}_{i=0}^{n-1} \mathsf{ilv}^{n-1-i}\,alt_{i+1} \\ &= \mathop{;}_{i=0}^{n-1} \mathsf{ilv}^{n-1-i}\,\mathsf{two}^i\,\mathsf{one}\,swp \\ &= \bowtie_{n-1}\,\mathsf{one}\,swp \end{aligned}$$

We can conclude that

$$\begin{aligned} \mathcal{B}_n &= (\mathbb{W}_{n-1}\,\mathsf{one}\,swp)^{-1}\,;\mathbb{W}_n\,cmp \\ &= \bowtie_{n-1}\,\mathsf{one}\,swp\,;\mathbb{W}_n\,cmp \end{aligned}$$

The balanced merger

In reference [11] the original designers of the balanced merger present it as a modification to the bitonic merger.

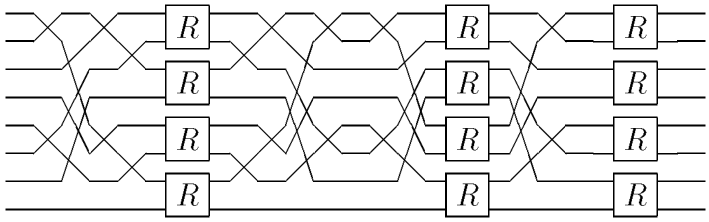

Figure 20: $\mathbb{W}_2\, R = \mathsf{vee}^2\, R \,;\, \mathbb{W}_1\, \mathsf{two}\, R = \mathbb{W}^2\, R \,;\, \mathsf{vee}\,\mathsf{two}\, R \,;\, \mathbb{W}_0\, \mathsf{two}^2\, R$

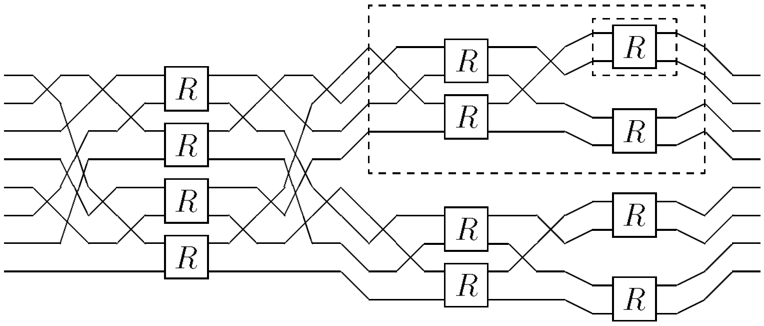

Figure 21: $\mathbb{W}_2\, R = \mathsf{vee}^2\, R \,;\, \mathsf{two}\, \mathbb{W}_1\, R = \mathsf{vee}^2\, R \,;\, \mathsf{two}(\mathsf{vee}^1\, R \,;\, \mathsf{two}\, \mathbb{W}_0\, R)$

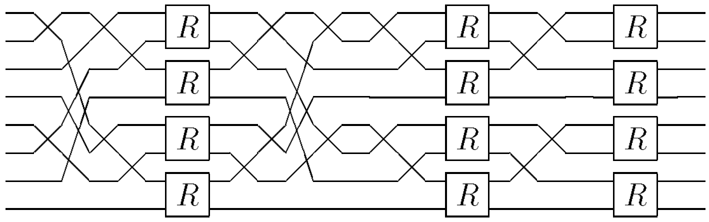

Figure 22: $\mathbb{W}_2\, R = \mathbb{W}_1\, \mathsf{vee}\, R \,;\, \mathsf{two}^2\, R = \mathbb{W}_0\, \mathsf{vee}^2\, R \,;\, \mathsf{two}\,\mathsf{vee}\, R \,;\, \mathsf{two}^2\, R$

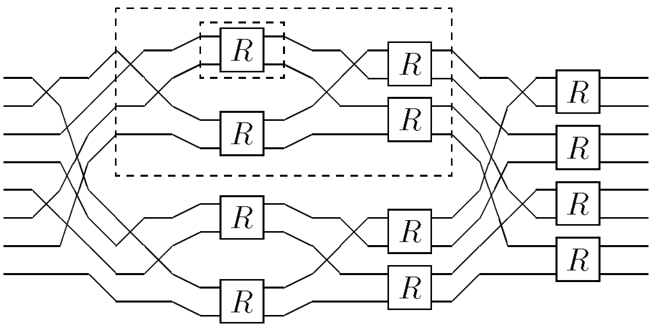

Figure 23: $\mathbb{W}_2\, R = \mathsf{vee}\, \mathbb{W}_1\, R \,;\, \mathsf{two}^2\, R = \mathsf{vee}(\mathsf{vee}\, \mathbb{W}_0\, R \,;\, \mathsf{two}\, R) \,;\, \mathsf{two}^2\, R$

> We apply the permutation $(n/2-1,\ n/2-2,\ \ldots,\ 1,\ 0,\ n/2,\ n/2+1,\ \ldots,\ n-2,\ n-1)$ to the first phase of the bitonic merging network to obtain the new first phase comparing elements $x(0)$ with $x(n-1)$, $x(1)$ with $x(n-2)$, $\ldots$ $x(n/2-1)$ and $x(n/2)$, where x is the input vector, that is, comparing the first element with the last one, the second with the second to last, etc. Applying this permutation to the following phases of the bitonic merging network does not change those phases. Instead, we follow the bitonic merging network in assuming the partition of the elements into two halves of the smaller and the larger elements and applying in the second phase the same structure of the first phase for both halves. We continue recursively for the consecutive phases.

The authors write sequences of numbers $x = (x_0, \ldots, x_i, \ldots, x_{n-1})$ to name the permutation that takes i to x_i. They also number the sequences in their diagrams from top to bottom, so the permutation that they write as $(n/2-1, n/2-2, \ldots, 1, 0, n/2, n/2+1, \ldots, n-2, n-1)$ is written $n\,;\mathsf{one}\ rev$ in our notation. It reverses the top half of a sequence of length n.

To construct the balanced merger from the bitonic merger, we transform the first rank of comparators from $\mathsf{ilv}^{p+1}\ cmp$ to $\mathsf{vee}^{p+1}\ cmp$ using the properties of the permutation $\mathsf{one}\ rev$ and the fact that rev is a left-identity of the bitonic merger.

$$\begin{aligned}
\mathsf{one}\ rev\ ; \mathcal{B}_{p+1} &= \{\text{ definition } \mathcal{B}\ \} \\
&\quad\ \mathsf{one}\ rev\ ; \mathsf{ilv}^{p+1}\ cmp\ ; \mathsf{two}\ \mathcal{B}_p \\
&= \{\text{ equation 14 }\} \\
&\quad\ \mathsf{vee}^{p+1}\ cmp\ ; \mathsf{one}\ rev\ ; \mathsf{two}\ \mathcal{B}_p \\
&= \{\ rev\ ; \mathcal{B}_p = \mathcal{B}_p\ \} \\
&\quad\ \mathsf{vee}^{p+1}\ cmp\ ; \mathsf{two}\ \mathcal{B}_p
\end{aligned}$$

The relation $\mathsf{vee}^{p+1}\ cmp$ compares the first and last elements of a sequence, the second and second last elements, and so on, as required.

We also want to replace each of the recursive calls of $\mathcal{B}_p$ by $\mathsf{one}\ rev\ ; \mathcal{B}_p$ in the same way, and so on recursively. It can be shown by induction that

$$\begin{aligned}
(\mathop{;}_{i=0}^{p} \mathsf{two}^i\ \mathsf{one}\ rev)\ ; \mathcal{B}_{p+1} &= \mathop{;}_{i=0}^{p+1} \mathsf{vee}^{p+1-i}\ \mathsf{two}^i\ cmp \\
&= \mathbb{W}_{p+1}\ cmp
\end{aligned}$$

So the balanced merger, $\mathcal{M}_{p+1}$, is just the network $\mathbb{W}_{p+1}\ cmp$ that we have already seen, and it is related to the bitonic merger by

$$\mathcal{M}_{p+1} = (\mathop{;}_{i=0}^{p} \mathsf{two}^i\ \mathsf{one}\ rev)\ ; \mathcal{B}_{p+1}$$

The wiring permutation $\mathop{;}_{i=0}^{p} \mathsf{two}^i\, \mathsf{one}\, rev$, when it operates on sequences of length 2^{p+2} as it does here, is $\mathbb{W}_p\, \mathsf{one}\, swp$, which we saw above.

$$
\begin{aligned}
\mathop{;}_{i=0}^{p} \mathsf{two}^i\, \mathsf{one}\, rev \;;\; 2^{p+2} &= \{\; rev \;;\; 2^{j+1} = \mathsf{vee}^j\, swp \;\} \\
&\quad \mathop{;}_{i=0}^{p} \mathsf{two}^i\, \mathsf{one}\, \mathsf{vee}^{p-i}\, swp \\
&= \{\; \mathsf{one}\, \mathsf{vee}\, R = \mathsf{vee}\, \mathsf{one}\, R \;\} \\
&\quad \mathop{;}_{i=0}^{p} \mathsf{two}^i\, \mathsf{vee}^{p-i}\, \mathsf{one}\, swp \\
&= \{\; \text{definition } \mathbb{W} \;\} \\
&\quad \mathbb{W}_p\, \mathsf{one}\, swp
\end{aligned}
$$

This is the permutation τ that appears mysteriously in reference [3] when the balanced merger is discussed. The natural language description of the balanced merger quoted above is typical of the way in which networks are described in the literature. Our formal description is much more precise, and it captures the designers' intuition in a satisfying way.

Knowing that the balanced merger is a veefly of comparators gives us numerous recursive decompositions of that network. In particular,

$$
\begin{aligned}
\mathcal{M}_0 &= cmp \\
\mathcal{M}_{n+1} &= \mathsf{vee}^{n+1}\, \mathcal{M}_0 \;;\; \mathsf{two}\, \mathcal{M}_n \\
&= \mathsf{vee}\, \mathcal{M}_n \;;\; \mathsf{two}^{n+1}\, \mathcal{M}_0
\end{aligned}
$$

The designers of the periodic balanced sorter show [11] that

$$
\mathsf{ilv}\; inc \;;\; \mathcal{M}_n \;=\; \mathsf{ilv}\; inc \;;\; \mathcal{M}_n \;;\; inc \tag{15}
$$

That is, the balanced merger sorts a sequence consisting of two interleaved sorted sequences. Applying the function ilv^k to each side of equation 15 gives

$$
\mathsf{ilv}^{k+1}\, inc \;;\; \mathsf{ilv}^k\, \mathcal{M}_n \;=\; \mathsf{ilv}^{k+1}\, inc \;;\; \mathsf{ilv}^k\, \mathcal{M}_n \;;\; \mathsf{ilv}^k\, inc \tag{16}
$$

To build a sorter for sequences of length 2^{n+1}, we need to relate an unsorted sequence (which satisfies $\mathsf{ilv}^{n+1}\, inc$) to its sorted permutation (which satisfies $\mathsf{ilv}^0\, inc$). We can do this by progressing through permutations that obey $\mathsf{ilv}^n\, inc$, $\mathsf{ilv}^{n-1}\, inc$ and so on. The network

$$
\mathcal{S}_n \;=\; \mathop{;}_{i=0}^{n} \mathsf{ilv}^{n-i}\, \mathcal{M}_i \tag{17}
$$

sorts in this way. The proof that it is a sorter is by induction on n, using equation 16. For a given size of input, $\mathcal{S}_n$ has the same number of comparators as the bitonic sorter.

The periodic balanced sorting network

What makes the balanced merger interesting is that the composition of $n+1$ copies of $\mathcal{M}_n$, that is $\mathcal{M}_n^{n+1}$, is also a sorter. For a VLSI implementation, the resulting periodic circuit is attractive because only one copy of $\mathcal{M}_n$ need actually be laid out and its outputs can be fed back to its inputs. Thus space, a scarce resource, is traded off against time.

To prove the periodic sorter correct, we need to show that (for $0 \le k \le n$)

$$\mathsf{ilv}^{k+1}\, inc\, ; \mathcal{M}_n \quad = \quad \mathsf{ilv}^{k+1}\, inc\, ; \mathcal{M}_n\, ; \mathsf{ilv}^{k}\, inc \tag{18}$$

because then an induction, and the fact that $\mathsf{ilv}^{n+1}\, inc$ is the identity on sequences of length 2^{n+1}, gives

$$\mathcal{M}_n^{n+1} \quad = \quad \mathcal{M}_n^{n+1}\, ; inc$$

which is the desired result.

The proof of equation 18 is by induction. The base case is equation 15, which is proved in reference [11]; we will not prove it here. For the step:

$$\begin{aligned}
&\mathsf{ilv}^{k+2}\, inc\, ; \mathcal{M}_{n+1}\\
&\qquad = \{\, \mathsf{ilv}^2\, R = \mathsf{vee}\, \mathsf{ilv}\, R \text{ and definition } \mathcal{M} \,\}\\
&\qquad\quad \mathsf{vee}\, \mathsf{ilv}^{k+1}\, inc\, ; \mathsf{vee}\, \mathcal{M}_n\, ; \mathsf{two}^{n+1}\, cmp\\
&\qquad = \{\, \text{homogeneity} \,\}\\
&\qquad\quad \mathsf{vee}(\mathsf{ilv}^{k+1}\, inc\, ; \mathcal{M}_n)\, ; \mathsf{two}^{n+1}\, cmp\\
&\qquad = \{\, \text{inductive hypothesis} \,\}\\
&\qquad\quad \mathsf{vee}(\mathsf{ilv}^{k+1}\, inc\, ; \mathcal{M}_n\, ; \mathsf{ilv}^{k}\, inc)\, ; \mathsf{two}^{n+1}\, cmp\\
&\qquad = \{\, \text{homogeneity and equation 13} \,\}\\
&\qquad\quad \mathsf{vee}(\mathsf{ilv}^{k+1}\, inc\, ; \mathcal{M}_n)\, ; \mathsf{vee}\, \mathsf{ilv}^{k}\, inc\, ; \mathsf{two}^{n+1}\, cmp\, ; \mathsf{ilv}^{k+1}\, inc\\
&\qquad = \{\, \text{reversing the steps in the above calculation} \,\}\\
&\qquad\quad \mathsf{ilv}^{k+2}\, inc\, ; \mathcal{M}_{n+1}\, ; \mathsf{ilv}^{k+1}\, inc
\end{aligned}$$

This demonstrates the correctness of the periodic sorter.

To compare the sizes of $\mathcal{S}_n$ and the periodic sorter, note that we have replaced each $\mathsf{ilv}^{n-i}\, \mathcal{M}_i$ in equation 17 by the larger $\mathcal{M}_n$. In $\mathcal{S}_n$, the ith column of mergers has $2^{n-i}(i+1)2^i = (i+1)2^n$ comparators while in the periodic sorter, each column of mergers has $(n+1)2^n$ comparators. This means that the complete periodic sorter has roughly twice as many comparators. For such a small constant factor, one might consider laying out the complete periodic network on silicon, instead of the smaller but less regular $\mathcal{S}_n$.

Conclusion

The work on permutation and comparator networks is only just starting. The approach looks promising, especially when compared with standard methods, which tend to make obscure appeals to the binary representations of indices. Our proof of the periodic sorter is appealingly simple, largely because we were able to use exactly the right recursive decomposition of the balanced merger. Our first attempt at the proof had the same structure as the original proof in reference [11]. It used an inappropriate recursive decomposition of the merger, and so was long and complicated. The fact that we can express alternative recursive decompositions easily is an important advantage of our use of structuring functions. It is to be hoped that it will also be useful in the mapping of algorithms onto structured networks.

There is clearly a whole family of structuring functions like vee waiting to be investigated; in particular, there is the structuring function that matches vee in the same way that two matches ilv. This will lead to a family of butterfly-like networks for different forms of divide-and-conquer algorithms.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The design and analysis of computer algorithms*, Addison–Wesley, 1974.

[2] K. E. Batcher, *Sorting networks and their applications*, in Proc. AFIPS Spring Joint Comput. Conf., Vol. 32, April 1968.

[3] G. Bilardi, *Merging and sorting networks with the topology of the omega network*, IEEE Transactions on Computers, Vol. 38, No. 10, October 1989.

[4] R. S. Bird, *Lectures on constructive functional programming*, in [5]. (Programming Research Group technical monograph PRG–69)

[5] M. Broy (ed.), *Constructive methods in computing science*, NATO advanced study institutes, Series F: Computer and systems sciences, Springer-Verlag, 1989.

[6] K. M. Chandy and J. Misra, *Parallel program design – a foundation*, Addison–Wesley, 1988.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine computation of complex Fourier series*, Mathematics of Computation, **19**, pp. 297–301, 1965.

[8] G. David, R. T. Boute and B. D. Shriver (eds.), *Declarative systems*, North-Holland, 1990.

[9] K. Davis and J. Hughes (eds.), *Functional programming, Glasgow 1989*, Springer Workshops in Computing, 1990.

[10] P. Denyer and D. Renshaw, *VLSI signal processing; a bit-serial approach*, Addison–Wesley, 1985.

[11] M. Dowd, Y. Perl, L. Rudolph and M. Saks, *The periodic balanced sorting network*, Journal of the ACM, Vol. 36, No. 4, October 1989.

[12] G. Jones, *Deriving the fast Fourier algorithm by calculation*, in [9]. (Programming Research Group technical report PRG–TR–4–89)

[13] G. Jones and M. Sheeran, *Circuit design in Ruby*, in [19].

[14] G. Jones and M. Sheeran, *The study of butterflies*, in this volume.

[15] T. Nakatani, S.-T. Huang, B. W. Arden and S. T. Tripathi, *K-Way Bitonic Sort*, IEEE Transactions on Computers, Vol. 38, No. 2, February 1989.

[16] M. Sheeran, *Describing hardware algorithms in Ruby*, in [8]. (Revised form appears as [17])

[17] M. Sheeran, *Describing butterfly networks in Ruby*, in [9].

[18] S. G. Smith, *Fourier transform machines*, pp. 147–199 in [10].

[19] Jørgen Staunstrup (ed.), *Formal methods for VLSI design*, North-Holland, 1990.

[20] H. S. Stone, *Parallel processing with the perfect shuffle*, IEEE Transactions on Computers, Vol. C-20, No. 2, February 1971.

[21] J. D. Ullman, *Computational aspects of VLSI*, Computer Science Press, 1984.

It may be said, therefore, that on these expanded membranes Nature writes, as on a tablet, the story of the modifications of species, so truly do all changes of the organisation register themselves thereon. Moreover the same colour patterns of the wings generally show, with great regularity, the degrees of blood-relationship of the species. As the laws of nature must be the same for all beings, the conclusions furnished by this group of insects must be applicable to the whole organic world; therefore, ...the study of butterflies – creatures selected as the types of airiness and frivolity – instead of being despised, will some day be valued a one of the most important branches of Biological science.

W. H. Bates (1864) *The Naturalist on the River Amazons*