A heuristic explanation of Batcher's Baffler

by Edsger W. Dijkstra *)

Abstract Batcher's Baffler — so named by David Gries — is a sorting algorithm that is of interest because many of its "comparison swaps" can be executed concurrently. It is also of interest because it used to be hard to explain.

This note explains Batcher's Baffler by designing it. Besides including all heuristics, it has two distinguishing features, both contributing to its clarity and brevity:

(0) the (little) theory the algorithm relies upon is dealt with in isolation;

(1) by suitable abstractions, all case analyses have been removed from the argument.

*) Author's address:

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
United States of America

Batcher's Baffler — so named by David Gries [1] after K.E. Batcher [0], who published his design in 1968— is a sorting algorithm. Its building block treats a set of disjoint pairs of elements, swapping each pair of values that is out of order; the pairs of the set being disjoint, they will be treated as if dealt with concurrently. Since eventually all pairs have to be in order, we are interested in theorems about sets of "comparison swaps" that maintain for some other pairs the fact that they are already in order.

We shall present the relevant lemmata graphically. A dotted arrow $x \dashrightarrow y$ stands for the comparison swap

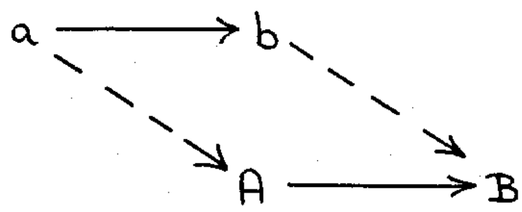
$$x, y := x \underline{\min} y ,\ x \underline{\max} y \qquad ;$$

a solid arrow $x \longrightarrow y$ stands for the relation

$$x \leq y \qquad .$$

The graphs representing our lemmata should be read as follows: if the inequalities corresponding to the solid arrows initially hold, they are maintained by the execution of the comparison swaps corresponding to the dotted arrows (whose inequalities eventually hold as well).

Lemma 0

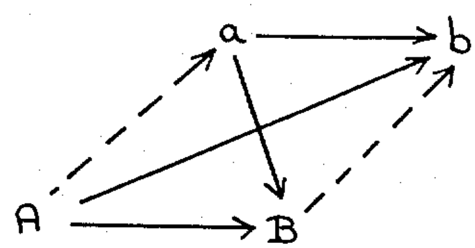


Proof According to the axiom of assignment, the postcondition $a \leq b \wedge A \leq B$ is guaranteed by the precondition

$$a \ \underline{\min} \ A \leq b \ \underline{\min} \ B \quad \wedge \quad a \ \underline{\max} \ A \leq b \ \underline{\max} \ B \quad ,$$

which is implied by the initial $a \leq b \wedge A \leq B$. (End of Proof.)

Lemma 1



Proof The four solid arrows are together equivalent to

$$a \ \underline{\max} \ A \leq b \ \underline{\min} \ B \quad ;$$

this relation is maintained by (each of) the operations corresponding to the dotted arrows, since the values of both its sides remain unaffected. (End of Proof.)

So much for the little theory we need.

* * *

Our purpose is to sort array $f(i: 0 \le i < N)$ in increasing order. For simplicity's sake, this finite array is mentally extended in both directions to infinity:

$$i<0 \Rightarrow f.i = \text{"}-\infty\text{"} \quad \text{and} \quad i \ge N \Rightarrow f.i = \text{"}+\infty\text{"} .$$

For brevity's sake, we introduce the transitive predicate OK given by

$$OK.i.j \equiv f.i \le f.j \quad ;$$

note that, thanks to the array extension, we have

$$i<0 \lor j \ge N \Rightarrow OK.i.j \quad .$$

Our purpose is to establish relation R given by

$$R \equiv \quad (\underline{A}\, i :: OK.i.(i+1))$$

by rearranging the values in $f(i: 0 \le i < N)$. (The advantage of the array extension is that the above universal quantification is over all integers, i.e. that we don't need to bother anymore about subscript bounds.)

The algorithm will manipulate array f only by means of the operation Ord given by

$$\begin{aligned} Ord.i.j = \ & \underline{if}\ OK.i.j \rightarrow skip \\ & [\!]\ \neg OK.i.j \rightarrow f.i, f.j := f.j, f.i \\ & \underline{fi} \quad . \end{aligned}$$

Operation Ord.i.j establishes OK.i.j ; note that, thanks to the array extension, we have

$$i<0 \lor j \geq N \Rightarrow Ord.i.j = skip$$

independently of the values in $f(i: 0 \leq i < N)$. The algorithm will invoke $Ord.i.j$ only with $i<j$, thus ensuring that the sequence $f(i: 0 \leq i < N)$ remains a permutation of its initial value.

After these preliminaries we can begin with the design of the algorithm. In view of R we choose as invariant P0, given by

$$P0 \equiv \quad (\underline{A}i :: OK.i.(i+t)) \quad ,$$

which is easily established since $t \geq N \Rightarrow P0$. Since —by construction!— $P0 \land t=1 \Rightarrow R$, our choice of invariant suggests for Batcher's Baffler the form

"establish $t \geq N$" {P0}
; <u>do</u> $t \neq 1 \rightarrow$ "reduce t under invariance of P0"
<u>od</u> {R} .

The guiding principle of our development is that, once an OK relation has been established, it will be maintained. This means that, if "reduce t under invariance of P0" involves the transition from $t=t'$ to $t=t''$, we require t' and t'' to satisfy

$$(\underline{A}i :: OK.i.(i+t'')) \Rightarrow (\underline{A}i :: OK.i.(i+t')) \quad ,$$

an implication whose validity requires (in view of

OK's transitivity) t'' to be a divisor of t'. Under that constraint the most modest decrease of t —i.e. the one that strengthens P0 as little as possible— is halving it. We propose to reduce t by halving it (and, hence, to restrict t to powers of 2). (Note that, at this stage of our analysis, this proposal is tentative; its wisdom, however, will transpire shortly.)

Explicit incorporation of the manipulation of t yields for Batcher's Baffler a program of the form

$t:= 1$; <u>do</u> $t < N \rightarrow t:= t\cdot 2$ <u>od</u> {P0 $\wedge$ (t is a power of 2)}
; <u>do</u> $t \neq 1 \rightarrow t:= t/2$ {P1}
 ; "restore P0" {P0}
<u>od</u> {R}

with P1 given by

$$P1 \equiv \quad (\underline{A}i:: OK.i.(i+2\cdot t)) \quad .$$

The rest of this note is concerned with the development of the subalgorithm for "restore P0" as specified by pre- and postcondition:

{P1} "restore P0" {P0} .

(For this subalgorithm it is no longer relevant that t is a power of 2.)

The design of Batcher's Baffler is driven by the desire to find sets of Ord operations with disjoint arguments because such Ord operations can be executed concurrently. Since each Ord operation establishes the corresponding OK relation, we are invited to consider – as our sweetly reasonable "units of establishment", so to speak – conjunctions of OK relations with disjoint arguments. Is it, for instance, possible to write postcondition P0 as $P2 \wedge P3$ such that in each of P2 and P3 the OK relations have disjoint arguments?

We can ensure $P0 \equiv P2 \wedge P3$ with

$P2 \equiv \quad (\underline{A}i : e.i : \ OK.i.(i+t))$ and

$P3 \equiv \quad (\underline{A}i : \neg e.i : \ OK.i.(i+t))$

with any boolean function e. Requiring the OK relations in P2 to have disjoint arguments boils down to requiring

$$e.i \Rightarrow \neg e.(i+t) \qquad ;$$

for P3 the analogous requirement is

$$\neg e.i \Rightarrow e.(i+t) \qquad .$$

Combining the two requirements, we conclude that with e satisfying

(0) $\qquad e.i \equiv \neg e.(i+t)$

$P2$ and $P3$ can each be established by a set of concurrent Ord operations. From now on, e denotes a predicate satisfying (0). Note that there are many such predicates, all variations on the same theme; the simplest one is

(1) $$e.i \equiv (i \ \underline{\text{mod}}\ 2\cdot t) < t \quad .$$

<u>Remark</u> It is the factor 2 in the above formula that will justify our earlier choice of reducing t by halving it. (End of Remark.)

Using $\|$ to denote the potentially concurrent combination of statements, we define $S2$ and $S3$ by

$$S2: \quad (\| i : e.i : Ord.i.(i+t)) \qquad \text{and}$$

$$S3: \quad (\| i : \neg e.i : Ord.i.(i+t)) \qquad .$$

<u>Remark</u> In these quantifications, i ranges over infinitely many values, but this presents no unsurmountable implementation problems since $Ord.i.(i+t)$ differs from skip for only a finite number of values of i. (End of Remark.)

Statement $S2$ establishes $P2$ and statement $S3$ establishes $P3$, but we cannot establish $P2 \wedge P3$ – i.e. $P0$ – by performing $S2$ and $S3$ (in some order) consecutively, for in general the second one will destroy what the first one has established. So after the execution of the first one, we have to proceed more carefully.

Let "restore P0" start with S2 establishing P2 . This choice of S2 is irrelevant since – see (0)– we are free to call either polarity of the partitioning predicate e . Proceeding from thereon "more carefully" means establishing P3 while maintaining P2 . The alternative to establishing P3 directly (i.e. by S3) is establishing P3 by means of a repetition with some invariant P4 , where P4 is a suitable generalization of P3 ; our purpose is to construct that repetition such that it has the stronger $P2 \wedge P4$ as invariant.

Before proceeding we rewrite P3 for simplicity's sake in such a way that its dummy is controlled by the same range as the dummy in P2 . In view of (0) , we can do so by renaming (with i+t replacing i):

$$P3 \equiv \quad (\underline{A}i : e.i : OK.(i+t).(i+2\cdot t)) \quad .$$

We generalize P3 by replacing the constant 2 by the variable u , i.e. we propose

$$P4 \equiv \quad (\underline{A}i : e.i : OK.(i+t).(i+u\cdot t)) \quad \text{with even } u.$$

The latter constraint on u ensures that the OK relations in P4 are disjoint so that P4 can be established by S4, given by

$$S4: \qquad (\| i: e.i: \mathrm{Ord}.(i+t).(i+u\cdot t)) \quad .$$

In view of our aim that the stronger $P2 \wedge P4$ be an invariant, it now stands to reason to investigate under which conditions S4 maintains P2. That is, for i satisfying $e.i$ we have to investigate the fate of $OK.i.(i+t)$; on account of (0) and because u is even, this is the same as investigating the fate of $OK.(i+u\cdot t).(i+u\cdot t+t)$ for any i satisfying $e.i$. With its incident Ord operations from S4 it yields the picture –nodes now being labelled by subscripts–

$$\begin{tikzcd}
 & i+u\cdot t \arrow[dr] & & i+2\cdot u\cdot t \\
i+t \arrow[ur, dashed] & & i+u\cdot t+t \arrow[ur, dashed] &
\end{tikzcd}$$

which is certainly not a lemma, but we can recognize the sequence $\dashrightarrow \longrightarrow \dashrightarrow$ in Lemma 1, redrawn for the purpose:

$$\begin{tikzcd}
 & i+u\cdot t \arrow[r] \arrow[dr] & i+2\cdot u\cdot t \\
i+t \arrow[ur, dashed] \arrow[urr] \arrow[rr] & & i+u\cdot t+t \arrow[u, dashed]
\end{tikzcd}$$

.

Of the three solid arrows added, the two horizontal ones are implied by $P1$ because u is even and the OK relation is transitive. The third one is implied by $P4(2\cdot u/u)$. (Here we have used the notation "$R(E/x)$" for the expression R in which E has been substituted for x.) In other words, for statement $S4$ we have established the theorem

$$\{P1 \wedge P2 \wedge P4(2\cdot u/u)\}\ S4\ \{P2 \wedge P4\} \quad .$$

Remembering that for the design of "restore $P0$" we could rely on the precondition $P1$ and taking the invariance of $P1$ for the time being for granted, we see from our last theorem, since —by construction—

$$P4 \wedge u=2 \Rightarrow P3 \quad ,$$

that we can establish $P3$ under invariance of $P2$ by first establishing $P4$ with u equal to a sufficiently high power of 2, and then repeatedly halving u while each time maintaining $P2 \wedge P4$ by an execution of $S4$.

Thus the fully annotated version of "restore $P0$" becomes

$\{P1\}$ S2 $\{P1 \wedge P2\}$
; "u := suitable power of 2" $\{P1 \wedge P2 \wedge P4\}$
; <u>do</u> $u \neq 2 \rightarrow u := u/2$ $\{P1 \wedge P2 \wedge P4(2 \cdot u/u)\}$
   ; S4 $\{P1 \wedge P2 \wedge P4\}$
<u>od</u> $\{P2 \wedge P4 \wedge u = 2$, hence $P0\}$.

We are left with two obligations: determining a "suitable power of 2" and showing the invariance of P1 .

Since

$$u \cdot t - t \geq N \Rightarrow (\underline{A} i :: OK.(i+t).(i+u \cdot t))$$

and the consequent implies P4, a u satisfying the antecedent would do the job. With for e the specific choice (1), the weaker $u \cdot t \geq N$ will do because

$ut \geq N$
$\Rightarrow$ { consider $i < -t$, $-t \leq i < 0$, $0 \leq i$ }
$(\underline{A} i :: i+t < 0 \vee \neg e.i \vee i + u \cdot t \geq N)$
$=$ { predicate calculus }
$(\underline{A} i : e.i : i+t < 0 \vee i + u \cdot t \geq N)$
$\Rightarrow$ { by the extension to an infinite array }
$(\underline{A} i : e.i : OK.(i+t).(i+u \cdot t))$
$=$ { definition of P4 }
P4 .

.../11a

Finally we have to show that $P1$, i.e.

$$(\underline{A}i :: OK.i.(i+2\cdot t))$$

is maintained by S2 and by S4 . The latter two are both of the form

$$(\|i : p.i : Ord.i.(i+m\cdot t))$$

with odd m and $p.i \equiv e.i$ or $p.i \equiv \neg e.i$, i.e. with

$$p.i \equiv \neg p.(i+m\cdot t) \qquad .$$

Hence, each OK relation of P1 occurs once as a solid arrow in the following diagrams with i satisfying p.i :

$$\begin{tikzcd}
i \arrow[r] \arrow[dr, dashed] & i+2\cdot t \arrow[dr, dashed] & \\
 & i+m\cdot t \arrow[r] & i+(m+2)\cdot t
\end{tikzcd} \qquad .$$

In this diagram, we recognize Lemma 0; hence neither S2 nor S4 falsifies P1. This concludes our heuristic explanation of Batcher's Baffler.

Concluding Remarks

Several aspects of the above are worth noting.

- For the expression $f.i \leq f.j$ we introduced the transitive predicate OK.i.j "for brevity's sake". More important than the physical abbreviation is that in the notation OK.i.j we only retained what matters in the sequel, viz. the two index values; not only the references to f, but —more importantly— the subexpressions f.i and f.j have disappeared; so has the relational operator $\leq$, and rightly so, for we could have wished to sort in descending order.

• The way in which the invariants have been derived from the postconditions —P0 from R and P4 from P3— is absolutely standard; it is known as "replacing a constant by a variable". The choice of which constant is to be replaced by a variable is usually severely constrained by the requirement that we can think of an initial value for that variable with which to establish the invariant.

• We have introduced two variables, t and u, constrained to be a power of 2. We could have been more explicit by representing in our analysis their values by 2^h and 2^{k+1} respectively, i.e. we could have introduced the natural variables h and k instead. It seems a minor notational variation, but I would like to point out that it makes all the difference. The difference is not so much that the identifiers t and u are shorter than the alternatives 2^h and 2^{k+1}. The difference is that with the latter notation their being a power of 2 would have permeated ~~all through~~ our formalism, even where their being a power of 2 had not yet been decided ~~yet~~ or did no longer matter. The nomenclature provided by t and u enables us to do justice to the latter disentanglement of the argument.

• The extension of the finite array to an infinite one was presented as a way of simplifying the postcondition and the intermediate assertions, but note that it has bought us much more. Without it, our two lemmata would not have sufficed and our invariance proofs would have been burdened by case analyses to take care of all sorts of boundary effects due to "missing elements".

• The extension to an infinite array has also protected us from the introduction of expressions like

$$2^{\lceil {}^2\log.N \rceil}$$

and from the suggestion that the algorithm is really designed for N of the form 2^n.

Acknowledgements

I am first and foremost indebted to four persons:

- I am indebted to K.E. Batcher for having designed this ingenious algorithm.
- I am indebted to D. Gries for drawing my attention to this algorithm by sending me a manuscript in which this algorithm was explained and proved to be correct. In the above I have followed his way of breaking down the invariant.
- I am indebted to C.S. Scholten for simplifying my original derivation by extending the finite array to an infinite one.
- I am indebted to A.J.M. van Gasteren for raising my standards of disentanglement, for making me more aware of the issues involved, and for teaching me the relevant techniques.

Without these four persons, the above heuristic explanation would not have been written.

Finally I mention in gratitude my opportunity of presenting (in successive stages) my explanation to about half a dozen audiences, for each time my audience did not act as sponge but as whetstone. I also thank the referees.

[0] K.E. Batcher, "Sorting networks and their applications", Proc. Spring Joint Computer Conference, 1968

[1] D. Gries, Private Communication