A heuristic explanation of Batcher's Baffler

[This is intended to be my final synthesis of EWD's 932b, 935a, 937, 939, and 940.]

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. (End of Abstract.)

Batcher's Baffler - so named by David Gries after K.E. Batcher, who designed it in 1968- is a sorting algorithm. Its building block considers 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

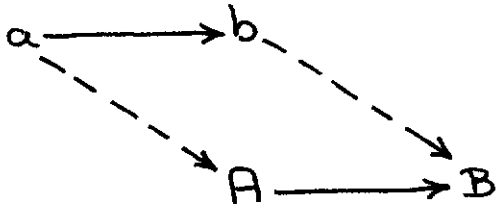
already in order.

We shall present the relevant lemmata graphically, in the form of directed graphs with dotted arrows and solid arrows. A dotted arrow $x \dashrightarrow y$ stands for the "comparison swap"

$$\{\text{true}\}\ x, y := x \underline{\min} y, x \underline{\max} y\ \{x \leq y\}\ .$$

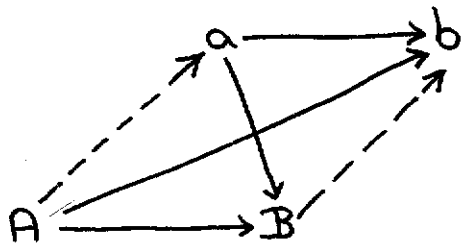
A solid arrow $x \rightarrow y$ stands for the invariant relation $x \leq y$; more precisely, the graphs representing our lemmata should be read as follows: if initially the inequalities corresponding to the solid arrows hold, they are maintained by the execution of the operations corresponding to the dotted arrows (whose inequalities eventually hold as well).

Lemma 0



Proof The maximum of the four values is initially an element of the set $\{b, B\}$, hence finally equal to B; the inequality $A \leq B$ is therefore maintained. Similarly, the minimum of the four values is initially an element of the set $\{a, A\}$, hence finally equal to a; the inequality $a \leq b$ is therefore maintained. (End of Proof.)

Lemma 1



Proof The four solid arrows are together equivalent to a max A ≤ b min B ; this relation is maintained by each of the operations corresponding to the dotted arrows. (End of Proof.)

So much for the little theory we need.

* * *

Our purpose is to sort array $f(i: 0 \leq i < N)$. For simplicity's sake, this finite array is mentally extended in both directions to infinity, i.e. with values "$-\infty$" to its left and values "$+\infty$" to its right (the direction from left to right being the order of increasing subscript value).

With the predicate OK being given by

$$OK.i.j \equiv f.i \leq f.j$$

our purpose is to establish relation R given by

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

by rearranging the values of $f(i: 0 \leq i < N)$. Note that, thanks to our extension, OK.i.j trivially holds for $i < 0 \vee j \geq N$.

The algorithm manipulates array f only by means of the operation Ord.i.j with $i < j$ and given by

$Ord.i.j = \underline{\text{if}}\ OK.i.j \to skip \ [\!]\ \neg OK.i.j \to swap.f.i.j\ \underline{\text{fi}}$

– the swap interchanges the values of $f.i$ and $f.j$ – . The operation $Ord.i.j$ establishes $OK.i.j$. Note that, thanks to our extension,

$$i<0 \ \vee\ j \geq N \ \Rightarrow\ Ord.i.j = skip \quad ,$$

independently of the value of $f(i: 0 \leq i < N)$.

In view of R we choose as invariant P_0 given by

$$P_0 \equiv \quad (\underline{A}i :: OK.i.(i+t)) \quad ,$$

which is easily established since $t \geq N \Rightarrow P_0$. This choice suggests for Batcher's Baffler the form

"establish $t \geq N$" $\{P_0\}$
; <u>do</u> $t \neq 1 \to$ "decrease t under invariance of P_0"
<u>od</u> $\{R\}$.

The guiding principle of our development is that, once an OK relation has been established, it will be maintained, i.e. if "decrease t under invariance of P_0" involves the transition from $t=t'$ to $t=t''$, we require

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

an implication whose validity requires (in view of OK's transitivity) t' to be a multiple of t''. Under that constraint, the most modest decrease of t – i.e. the one that strengthens P_0 as little as possible – is halving it, and we therefore propose to restrict t to powers of 2. (At this stage this proposal is tentative; its wisdom will

transpire shortly.)

Explicit incorporation of the manipulations on t yields for Batcher's Baffler

$t := 1$; **do** $t < N \rightarrow t := 2 \cdot t$ **od** $\{P_0 \wedge t \text{ is a power of } 2\}$

; **do** $t \neq 1 \rightarrow t := t/2$

; $\{P_1 : (\underline{A}i :: OK.i.(i + 2 \cdot t))\}$

"restore P_0"

$\{P_0 : (\underline{A}i :: OK.i.(i + t))\}$

od $\{R\}$.

The rest of this note is concerned with the development of the subalgorithm for "restore P_0" as specified above by its pre- and postconditions. (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 groups of Ord operations with disjoint argument pairs, because such Ord operations could be executed concurrently. As each Ord operation establishes the corresponding OK relation, we are invited to select from the postcondition P_0 a group of OK relations with disjoint argument pairs. Such a group occurs in

$P_2 \equiv \quad (\underline{A}i : e.i : OK.i.(i + t))$

if we can find a predicate e such that

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

There are many such predicates, all variations on the same theme; the simplest one is

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

Remark It is the factor of 2 in the above formula that will justify our earlier choice to restrict t to powers of 2. (End of Remark.)

In order to capture the remaining OK relations of P_0 we write P_0 as $P_2 \wedge P_3$, thus finding

$P_3 \equiv \quad (\underline{A}i : \neg e.i : OK.i.(i+t))$.

Note that (0) is maintained when e is replaced by $\neg e$; in other words: also the OK relations occurring in P_3 have disjoint argument pairs.

Using $\|$ to denote the potentially concurrent combination of statements, we have indeed

S_2: $\quad \{true\}\ (\|i : e.i : Ord.i.(i+t))\ \{P_2\}$ and

S_3: $\quad \{true\}\ (\|i : \neg e.i : Ord.i.(i+t))\ \{P_3\}$.

But we cannot achieve $P_2 \wedge P_3$ -i.e. P_0- by performing S_2 and S_3 consecutively (in some order), for in general the second one will destroy what the first one has accomplished. So we have to proceed more carefully, e.g. first establishing P_2 -the choice is irrelevant, as we were free to call either polarity of the partitioning predicate e - and then establishing P_3 with a repetition for which P_2 is an invariant, i.e. we may expect that repetition to establish $P_2 \wedge P_3$ under invariance of $P_2 \wedge P_4$, where P_4

is a suitable generalization of P_3 .

In order to be a suitable generalization of P_3, P_4 too should be composed of OK-relations with disjoint argument pairs. In view of (0) this can be achieved by replacing t in $OK.i.(i+t)$ by an odd multiple of t. Since the general even value has a slightly simpler form than the general odd value, we propose to rewrite P_3 -see(0)- as

$$P_3 \equiv (\underline{A}i : e.i : OK.(i+t).(i+2\cdot t))$$

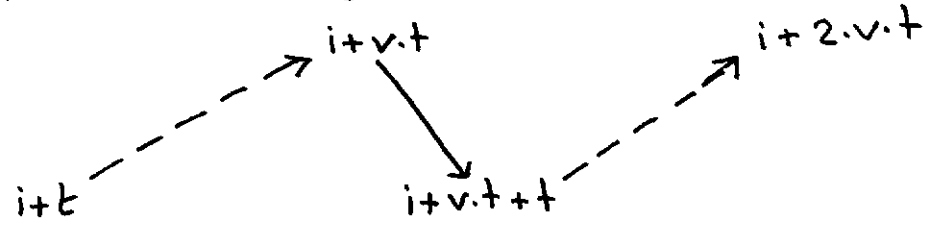
and propose for P_4

$$P_4 \equiv (\underline{A}i : e.i : OK.(i+t).(i+v\cdot t)) \quad \text{with even } v,$$

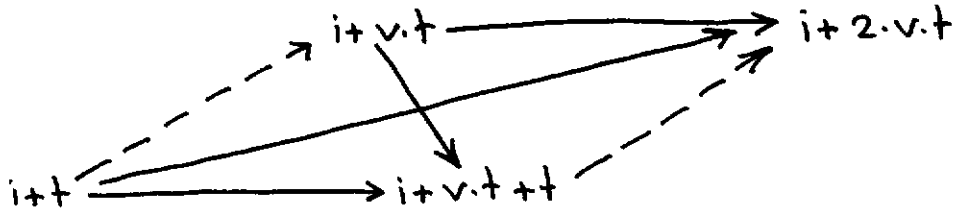
the latter to ensure that all argument pairs in P_4 are disjoint. It now stands to reason to consider

$$S4: \quad \{true\}\ (||i : e.i : Ord.(i+t).(i+v\cdot t))\ \{P_4\}$$

and to investigate under which conditions S4 maintains P_2. Because v is even, it suffices to investigate the fate of $OK.(i+v\cdot t).(i+v\cdot t+t)$ with i satisfying $e.i$. With its incident Ord operations it yields the picture



which is certainly not a lemma, but we can recognize the sequence --→⟶--→ in Lemma1, redrawn for the purpose:



Of the three solid arrows added, the two horizontal ones – v being even! – are implied by P_1. The third one is implied by $P4^{\check{}}_{2.v}$. In other words, for statement $S4$ we have established the theorem

$$\{P_1 \wedge P_2 \wedge P4^{\check{}}_{2.v}\}\ S4\ \{P_2 \wedge P_4\} .$$

Taking the invariance of P_1 for the time being for granted, this theorem tells us, as

$$P_4 \wedge v=2 \Rightarrow P_3 ,$$

that we can establish P_3 under invariance of P_2 by first establishing P_4 with v equal to a sufficiently high power of 2, and then repeatedly halving v while maintaining $P_2 \wedge P_4$ by executing $S4$.

Independently of e, P_4 holds if $(v-1).t \geq N$; with e as given in (1), however, the weaker $v.t \geq N$ suffices. With that choice for e (and still under the assumption of the invariance of P_1) we get the following program for Batcher's Baffler.

$$
\begin{array}{l}
\|[\ t, v0 : \mathrm{int}\ ;\ \ t, v0 := 1, 1 \\
\ ;\ \underline{\mathrm{do}}\ t < N \rightarrow\ t := 2\cdot t\ \underline{\mathrm{od}}\ \{P0 \wedge v0\cdot t \geq N\} \\
\ ;\ \underline{\mathrm{do}}\ t \neq 1 \rightarrow\ t, v0 := t/2\ ,\ 2\cdot v0\ \{P1 \wedge v0\cdot t \geq N\} \\
\qquad ;\ (\|i : e.i : \mathrm{Ord}.i.(i+t))\ \{P1 \wedge P2 \wedge v0\cdot t \geq N\} \\
\qquad ;\ \|[\ v : \mathrm{int};\ \ v := v0\ \{P1 \wedge P2 \wedge P4\} \\
\qquad\ ;\ \underline{\mathrm{do}}\ v \neq 2 \rightarrow\ v := v/2\ \{P1 \wedge P2 \wedge P4^{\check{}}_{2\cdot v}\} \\
\qquad\qquad ;(\|i : e.i : \mathrm{Ord}.(i+t).(i+v\cdot t)) \\
\qquad\qquad\qquad\qquad \{P1 \wedge P2 \wedge P4\} \\
\qquad\ \underline{\mathrm{od}}\ \{P2 \wedge P3\} \\
\qquad \|\ \{P0 \wedge v0\cdot t \geq N\} \\
\ \underline{\mathrm{od}}\ \{P0 \wedge t = 1\} \\
\|\ \{R\}\ .
\end{array}
$$

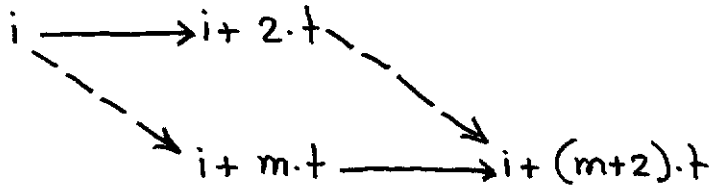
We are left with the obligation to verify that $P1$, i.e.

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

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

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

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



,

however, we recognize Lemma 0. With either choice

for p, each OK relation of P_1 occurs, because m is odd, in precisely one such diagram, and this settles the invariance of P_1.

And this completes our derivation of Batcher's Baffler.

History and acknowledgements

We are first and foremost indebted to K.E. Batcher, who invented this algorithm in 1968. I saw it for the first time explained and proved to be correct in a manuscript by David Gries. His explanation is in two steps, dealing with "sort two-ordered" first, and then reducing the complete sorting task to repeated applications of the former; in the above I have followed his way of breaking down the invariant.

In the fall semester of 1985, my target was the derivation of programs rather than just their explanation. The leitmotiv of that course was that, in the design of a program, it often pays to develop the underlying theory first. It had been chosen because of my experience that such a theory could provide a strong heuristic guidance. Because I did not know it too well and it looked sufficiently ambitious, Batcher's Baffler was one of the examples selected as proving ground for this approach. As we did not know what would be needed, our initial theory encompassed more than the two lemmata given here.

The ATAC (= Austin Tuesday Afternoon Club) focussed its attention on the theory and the graphical notation of its lemmata; this led to a completion of the theory and an improvement of its presentation.

The ETAC (= Eindhoven Tuesday Afternoon Club) focussed its attention on the argument that led to my design and suggested several improvements. There, C.S. Scholten suggested the extension of the finite array to an infinite one; this suggestion deserves to be mentioned explicitly since it did away with the last case analyses in the argument.

Finally, an oversight -in the initialization of P4- was pointed out to me in late December '85, when I had the privilege of presenting Batcher's Baffler to the Eindhoven Informatics Colloquium.

All these contributions are gratefully acknowledged.

Austin, 19 February 1986

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
United States of America.