Deriving a very simple FLAME algorithm

Robert van de Geijn

Draft
February 2, 2004

Given a vector of n numbers

$$x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_{n-1} \end{pmatrix}$$

the sum of those numbers is given by

$$\alpha = \sum_{i=0}^{n-1} \chi_i$$

Note 1 *We use lower case letters for vectors and lower case Greek letters for scalars.*

Let us define the operation to be performed by

$$\alpha := \mathrm{Sum}(x).$$

We will derive an algorithm for computing α by filling out the worksheet in Fig. 1.

Note 2 *Read this as "α becomes the sum of the elements of vector x".*

Note 3 *Notice that we carefully distinguish between "$=$" which indicates equality and "$:=$" which indicates assignment.*

Note 4 *Below, the step numbers refer to the numbers in the "Steps" column on the left in Fig. 2.*

Step 1: Precondition and postcondition Letting $\hat{\alpha}$ denote the original contents of α, the precondition (state of the variables at the beginning), P_{pre}, for our algorithm will be $\alpha = \hat{\alpha}$ while the postcondition, P_{post}, (state of the variables upon completion) is that $\alpha = \mathrm{Sum}(x)$. This is indicated in Steps 1a and 1b in Fig. 2.

Step 2: Determining loop-invariants As part of the computation of α it is logical to sweep through x in a consistent manner, e.g. from top to bottom. Our notation captures this by partitioning x like

$$x \rightarrow \left(\frac{x_T}{x_B} \right)$$

where x_T and x_B represent the top and bottom part of the vector.

Note 5 *Read x_T and x_B as "x-Top" and "x-Bottom", respectively.*

Note 6 *The idea is that each of the two parts of x at a given state of the algorithm has been updated or used in some (typically different) fashion. For example, we will see next that elements of x_T have already been summed into α while elements of x_B have not.*

Step	Annotated Algorithm: $[D, E, F, \ldots] = \text{op}(A, B, C, D, \ldots)$
1a	$\{P_{\text{pre}}\}$
4	**Partition** **where**
2	$\{P_{\text{inv}}\}$
3	**while** G **do**
2,3	$\{(P_{\text{inv}}) \wedge (G)\}$
5a	**Repartition** **where**
6	$\{Q_{bu}\}$
8	S_U
7	$\{Q_{au}\}$
5b	**Continue with**
2	$\{P_{\text{inv}}\}$
	enddo
2,3	$\{(P_{\text{inv}}) \wedge \neg (G)\}$
1b	$\{P_{\text{post}}\}$

Figure 1: Worksheet for deriving an algorithm.

Step	Annotated Algorithm: $\alpha := \text{Sum}(x)$
1a	$\{\alpha = \hat{\alpha}\}$
4	**Partition** $x \rightarrow \left(\begin{array}{c} x_T \\ \hline\hline x_B \end{array}\right)$ $\alpha = 0$ **where** x_T has 0 elements
2	$\{\alpha = \text{Sum}(x_T)\}$
3	**while** $n(x_T) \neq n(x)$ **do**
2,3	$\{(\alpha = \text{Sum}(x_T)) \wedge (n(x_T) \neq n(x))\}$
5a	**Repartition** $\left(\begin{array}{c} x_T \\ \hline\hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_0 \\ \hline\hline \chi_1 \\ \hline x_2 \end{array}\right)$ **where** χ_1 is a scalar
6	$\{\alpha = \text{Sum}(x_0)\}$
8	$\alpha := \alpha + \chi_1$
7	$\{\alpha = \text{Sum}(x_0) + \chi_1\}$
5b	**Continue with** $\left(\begin{array}{c} x_T \\ \hline\hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline\hline x_2 \end{array}\right)$
2	$\{\alpha = \text{Sum}(x_T)\}$
	enddo
2,3	$\{(\alpha = \text{Sum}(x_T)) \wedge \neg (n(x_T) \neq n(x))\}$
1b	$\{\alpha = \text{Sum}(x)\}$

Figure 2: Worksheet for deriving an algorithm for summing the elements of a vector x (Variant 1).

Next, we take this partitioned vector x and substitute it into the postcondition to find that

$$\alpha = \text{Sum}\left(\left(\frac{x_T}{\overline{x_B}}\right)\right).$$

Manipulating this we notice that

$$\alpha = \text{Sum}(x_T) + \text{Sum}(x_B).$$

Note 7 *The above equality shows how α can be computed from the two separate parts of x.*

Notice that the loop-invariant, P_{inv}, is a predicate that indicates the state of the variables (in this case α) at the top of the *while* loop in Fig. 2. Notice that the loop computes α and thus at the top of the loop we would expect to have computed only some of the final result. For example, at the top of the loop, we may indicate the contents of α as

$$P_{\text{pre}} : \alpha = \text{Sum}(x_T). \tag{1}$$

This condition is now entered in the worksheet in Fig. 2 in four places:

1. At the top of the loop-body.
2. Before entering the loop. The idea is that the first time through the loop, α contains exactly what it contained before entering the loop. Thus, the loop-invariant must already be *true* there.
3. At the bottom of the loop-body. The idea is that no computation happens between the bottom of the loop-body and the next time the top of the loop-body is reached.
4. Upon completion of the loop.

Step 3: Loop-guard Notice that upon completion of the loop

- The loop-invariant is *true.*
- The loop-guard G is *false.*
- The condition

  $$(\alpha = \text{Sum}(x_T)) \wedge \neg G$$

  must imply that $\alpha = \text{Sum}(x)$.

Notice that *if* x_T is all of x, then this is the case. So, if we take $G : n(x_T) \neq n(x)$ (read as: the length of x_T is not the same as the length of x) then $\neg G$ means that $n(x_T) = n(x)$ and therefore

$$((\alpha = \text{Sum}(x_T)) \wedge \neg G) \Rightarrow (\alpha = \text{Sum}(x))$$

This argument determines the required loop-guard G that appears in the worksheet.

Step 4: Initialization Next, we must find an initialization, S_I, that has the following properties:

- No computation is performed.
- The initialization changes the contents of α so that the loop-invariant is satisfied.

We note that if we partition

$$x \rightarrow \left(\frac{x_T}{\overline{x_B}}\right)$$

where x_T has *no* elements (and hence x_B is all of x) *and* we set α equal to zero ($\alpha := 0$), then we put the variables x_T, x_B, and α in a state where the loop-invariant is satisfied. This initialization appears in Step 4 in Fig. 2.

Step 5: Progressing through the vector We now note that as part of the computation x_T starts by having no elements, but must ultimately equal all of x. Thus, as part of the loop elements must be taken from x_B and added to x_T. This is denoted in Fig. 2 by the statements

$$\left(\begin{array}{c} x_T \\ \hline\hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_0 \\ \hline\hline \chi_1 \\ \hline x_2 \end{array}\right)$$

and

$$\left(\begin{array}{c} x_T \\ \hline\hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline\hline x_2 \end{array}\right).$$

This notation simply implies that the top element of x_B, χ_1, migrates from x_B to x_T.

Step 6: Determining the state after repartitioning The repartitioning in Step 5a does not change the contents of α: it is really just an indexing operation. We can thus ask ourselves the question of what the contents of α are in terms of the exposed parts of x (x_0, χ_1, and x_2). We can derive this condition, $Q_{\rm before}$, via *textual substitution*: The repartitioning in Step 5a means that

$$\begin{array}{c} x_T = x_0 \\ \hline\hline x_B = \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array}\right) \end{array}.$$

If we substitute the quantities on the right of the equalities into the loop-invariant we find that

$$\alpha = \mathrm{Sum}(x_T)$$

implies that

$$\alpha = \mathrm{Sum}(x_0)$$

which is entered in Step 6 in Fig. 2.

Step 7: Determining the state after moving the double lines Next, we notice that the moving of the double lines in Step 5b means that now

$$\begin{array}{c} x_T = \left(\begin{array}{c} x_0 \\ \hline \chi_1 \end{array}\right) \\ \hline\hline x_B = x_2 \end{array}.$$

Since the loop-invariant must be true after this redefinition of x_T and x_B, we substitute the quantities on the right of the equalities into the loop-invariant we find that now

$$\alpha = \mathrm{Sum}\left(\left(\begin{array}{c} x_0 \\ \hline\hline \chi_1 \end{array}\right)\right)$$

which implies that

$$\alpha = \mathrm{Sum}(x_0) + \chi_1.$$

This is entered for $Q_{\rm after}$ in Step 7 in Fig. 2.

Step 8 Comparing the contents in Step 6 and Step 7 now tells us that α must be changed from equalling only $\mathrm{Sum}(x_0)$ to equalling $\mathrm{Sum}(x_0)+\chi_1$. We conclude that it must be updated by adding χ_1 (since it already equals $\mathrm{Sum}(x_0)$):

$$\alpha := \alpha + \chi$$

which is entered for S_U in Fig. 2.

Exercise 1. *Derive an algorithm for computing $\alpha := Sum(x)$ by picking the loop-invariant as*

$$\alpha = Sum(x_B)$$