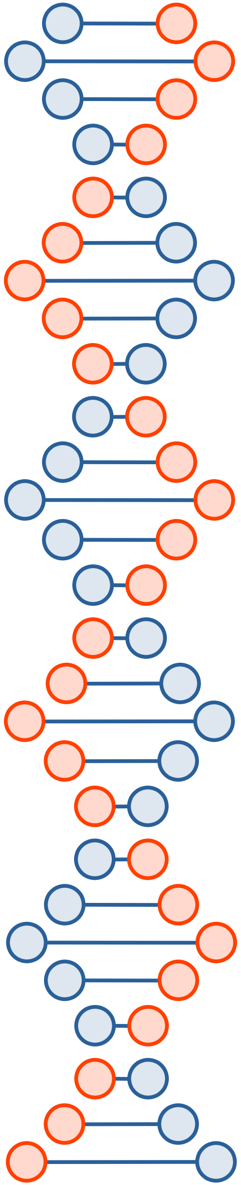
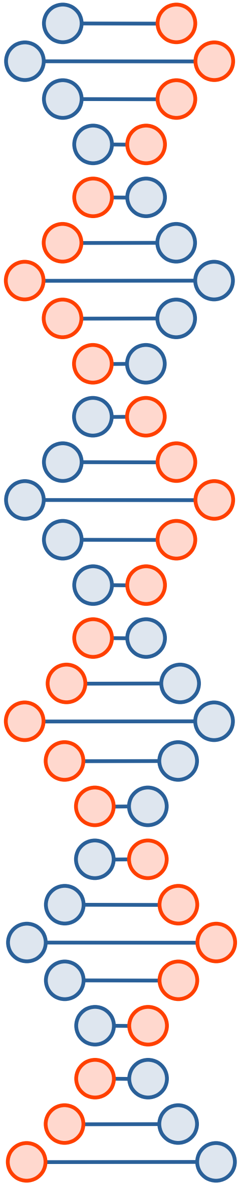


High Performance Computing

Ch. 2
Basic optimization techniques for serial code

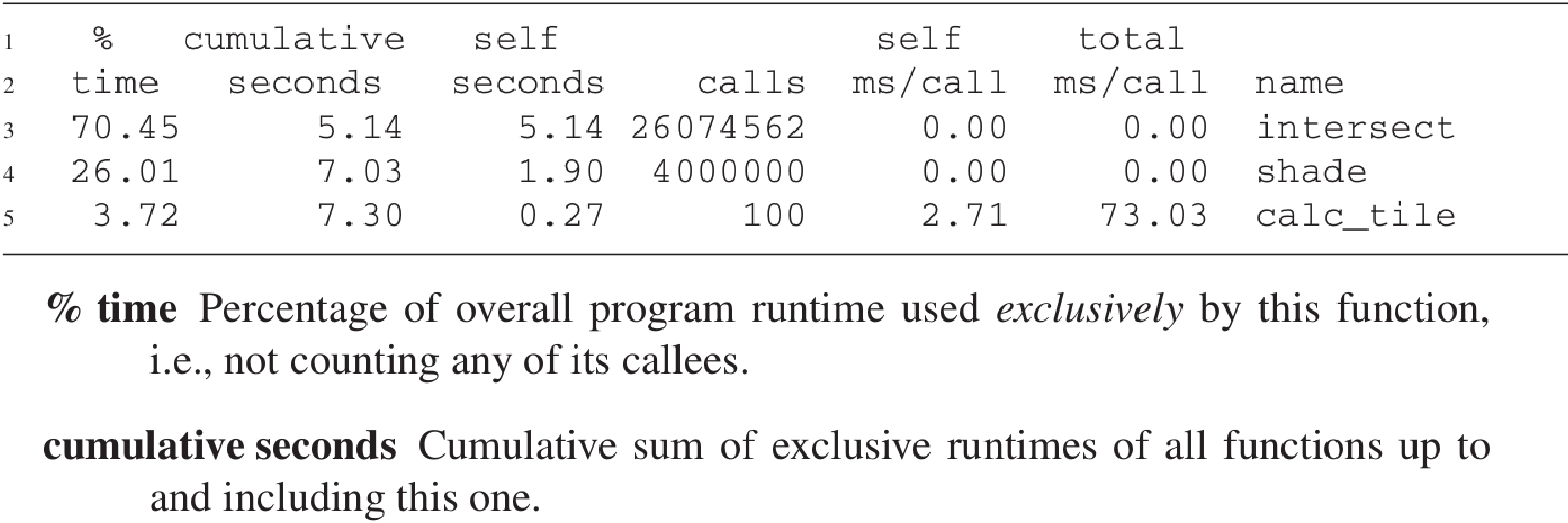
Computer Eng., KMITL
Assoc. Prof. Dr. Surin. K.



Basic optimization techniques for serial code

- 2.1 Scalar profiling . . . .
- 2.2 Common sense optimizations . . . .
- 2.3 Simple measures, large impact . . . . . .
- 2.4 The role of compilers . . . .
- 2.5 C++ optimizations . . . .

2.1 Scalar Profiling

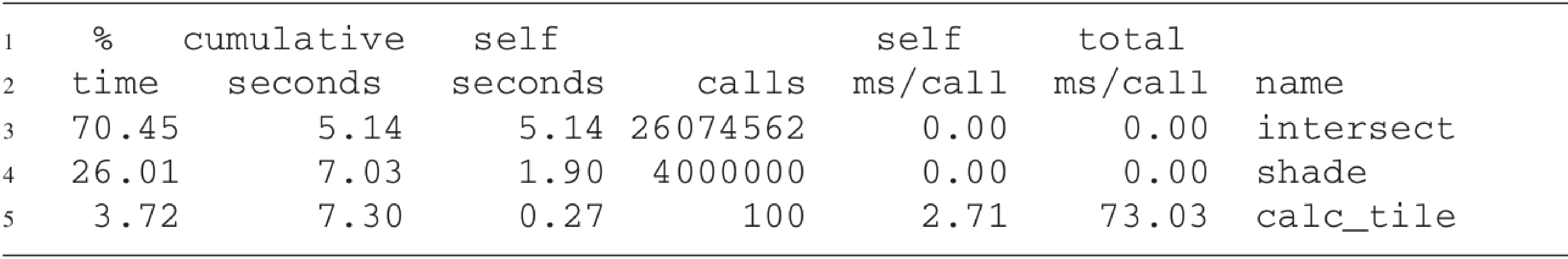


```
  %    cumulative   self              self     total
 time    seconds   seconds    calls  ms/call  ms/call  name
 70.45      5.14     5.14 26074562     0.00     0.00  intersect
 26.01      7.03     1.90  4000000     0.00     0.00  shade
  3.72      7.30     0.27      100     2.71    73.03  calc_tile
```

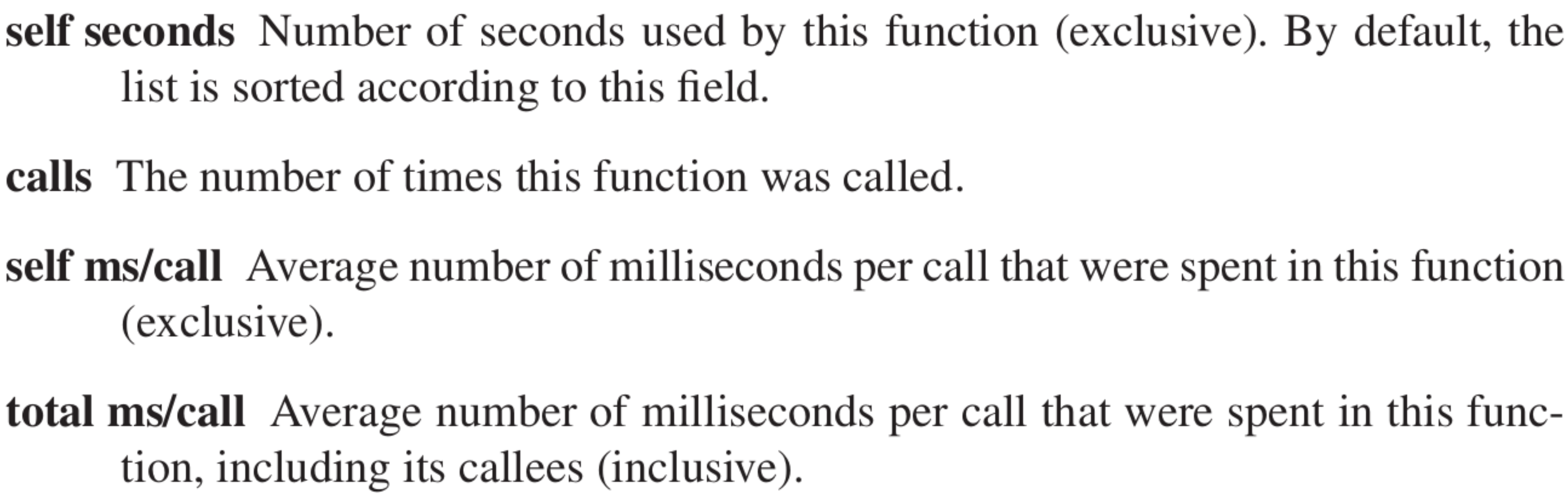
% time Percentage of overall program runtime used *exclusively* by this function, i.e., not counting any of its callees.

cumulative seconds Cumulative sum of exclusive runtimes of all functions up to and including this one.

2.1 Scalar Profiling



```
  %     cumulative    self                 self      total
 time     seconds    seconds      calls  ms/call   ms/call   name
 70.45       5.14       5.14  26074562      0.00      0.00   intersect
 26.01       7.03       1.90   4000000      0.00      0.00   shade
  3.72       7.30       0.27       100      2.71     73.03   calc_tile
```



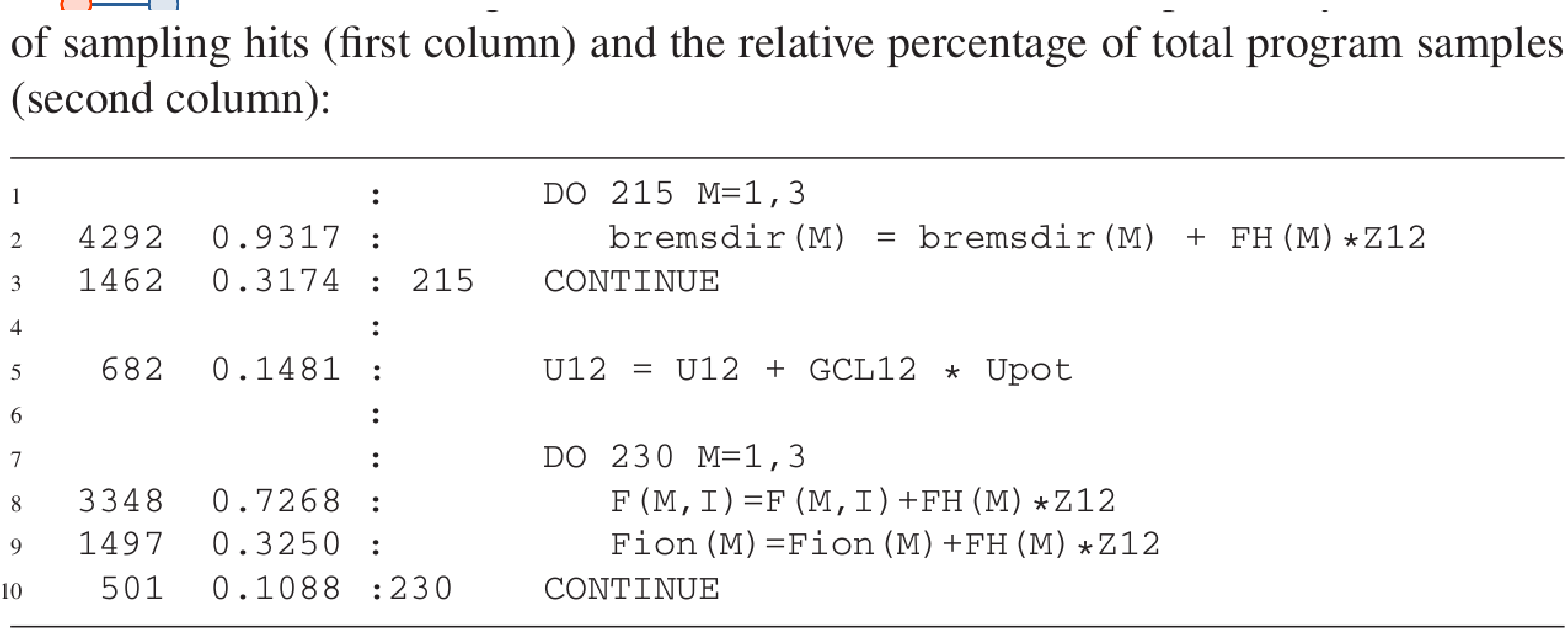
self seconds Number of seconds used by this function (exclusive). By default, the list is sorted according to this field.

calls The number of times this function was called.

self ms/call Average number of milliseconds per call that were spent in this function (exclusive).

total ms/call Average number of milliseconds per call that were spent in this function, including its callees (inclusive).

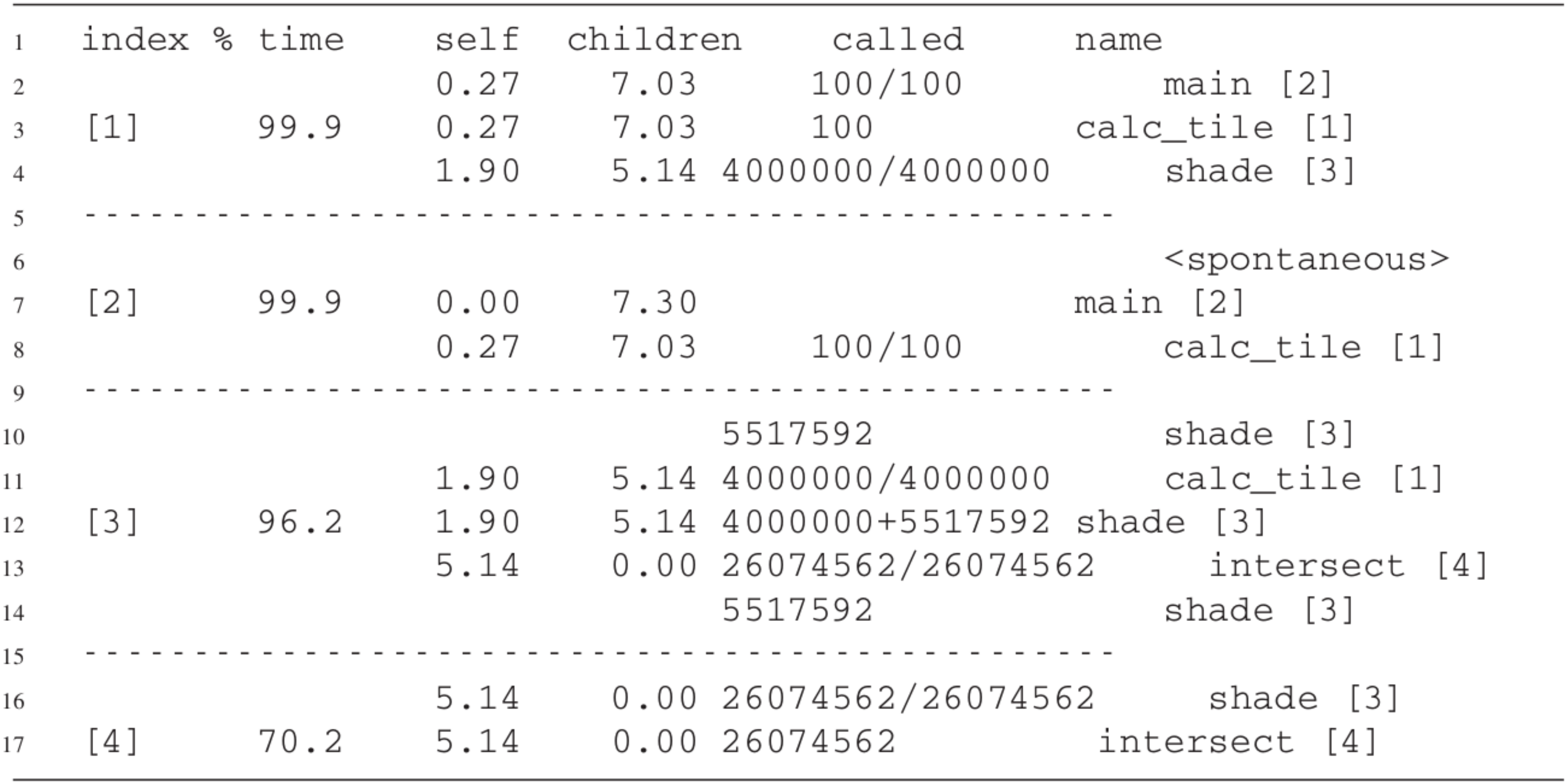
2.1 Scalar Profiling



of sampling hits (first column) and the relative percentage of total program samples (second column):

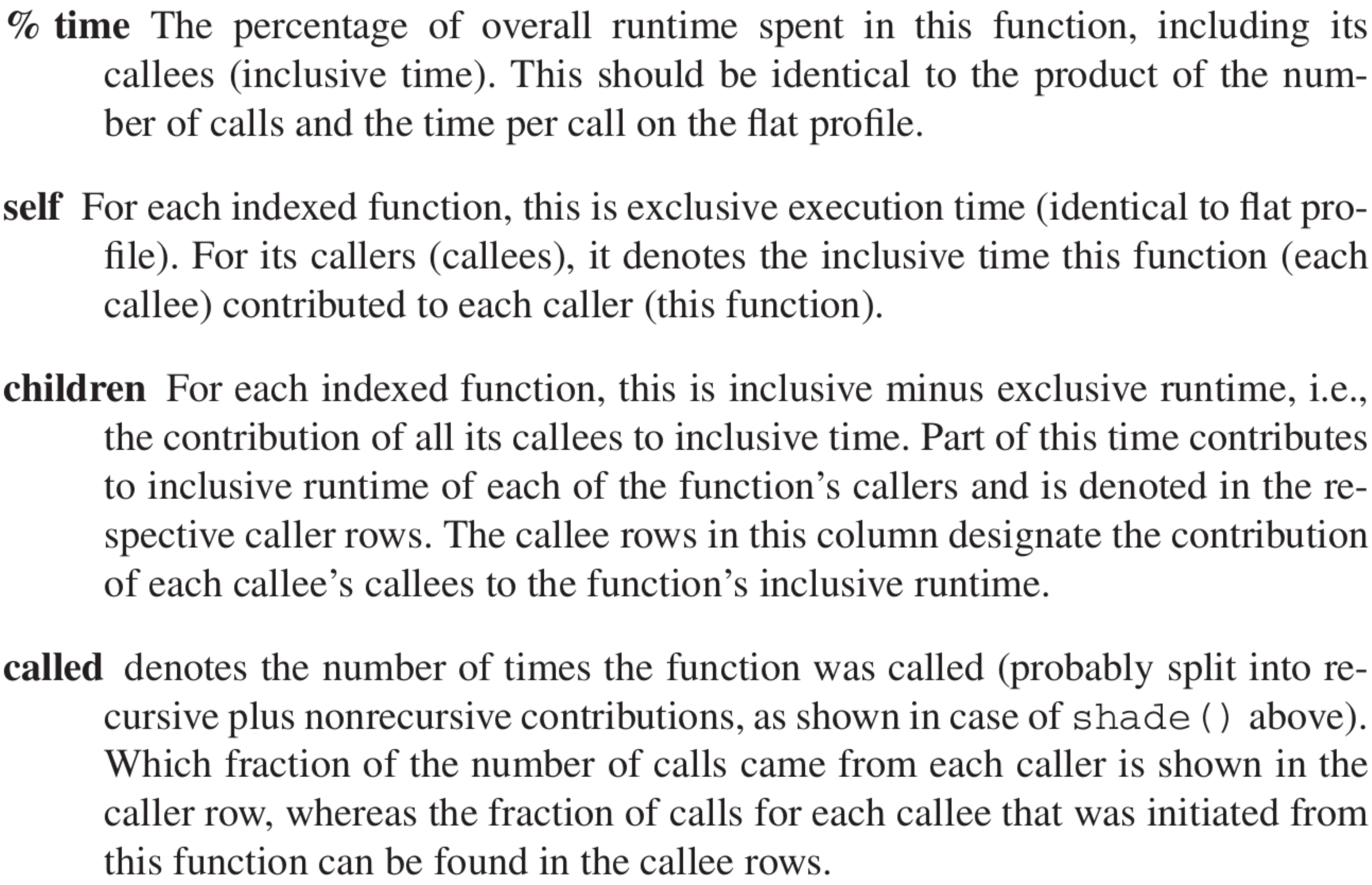
```
                      :        DO 215 M=1,3
   4292   0.9317 :           bremsdir(M) = bremsdir(M) + FH(M)*Z12
   1462   0.3174 :  215    CONTINUE
                      :
    682   0.1481 :         U12 = U12 + GCL12 * Upot
                      :
                      :        DO 230 M=1,3
   3348   0.7268 :           F(M,I)=F(M,I)+FH(M)*Z12
   1497   0.3250 :           Fion(M)=Fion(M)+FH(M)*Z12
    501   0.1088 :230      CONTINUE
```

2.1 Scalar Profiling



```
index % time    self  children    called     name
                0.27    7.03     100/100         main [2]
[1]     99.9    0.27    7.03     100         calc_tile [1]
                1.90    5.14 4000000/4000000     shade [3]
- - - - - - - - - - - - - - - - - - - - - - - -
                                                 <spontaneous>
[2]     99.9    0.00    7.30                 main [2]
                0.27    7.03     100/100         calc_tile [1]
- - - - - - - - - - - - - - - - - - - - - - - -
                             5517592             shade [3]
                1.90    5.14 4000000/4000000     calc_tile [1]
[3]     96.2    1.90    5.14 4000000+5517592 shade [3]
                5.14    0.00 26074562/26074562     intersect [4]
                             5517592             shade [3]
- - - - - - - - - - - - - - - - - - - - - - - -
                5.14    0.00 26074562/26074562     shade [3]
[4]     70.2    5.14    0.00 26074562         intersect [4]
```

2.1 Scalar Profiling



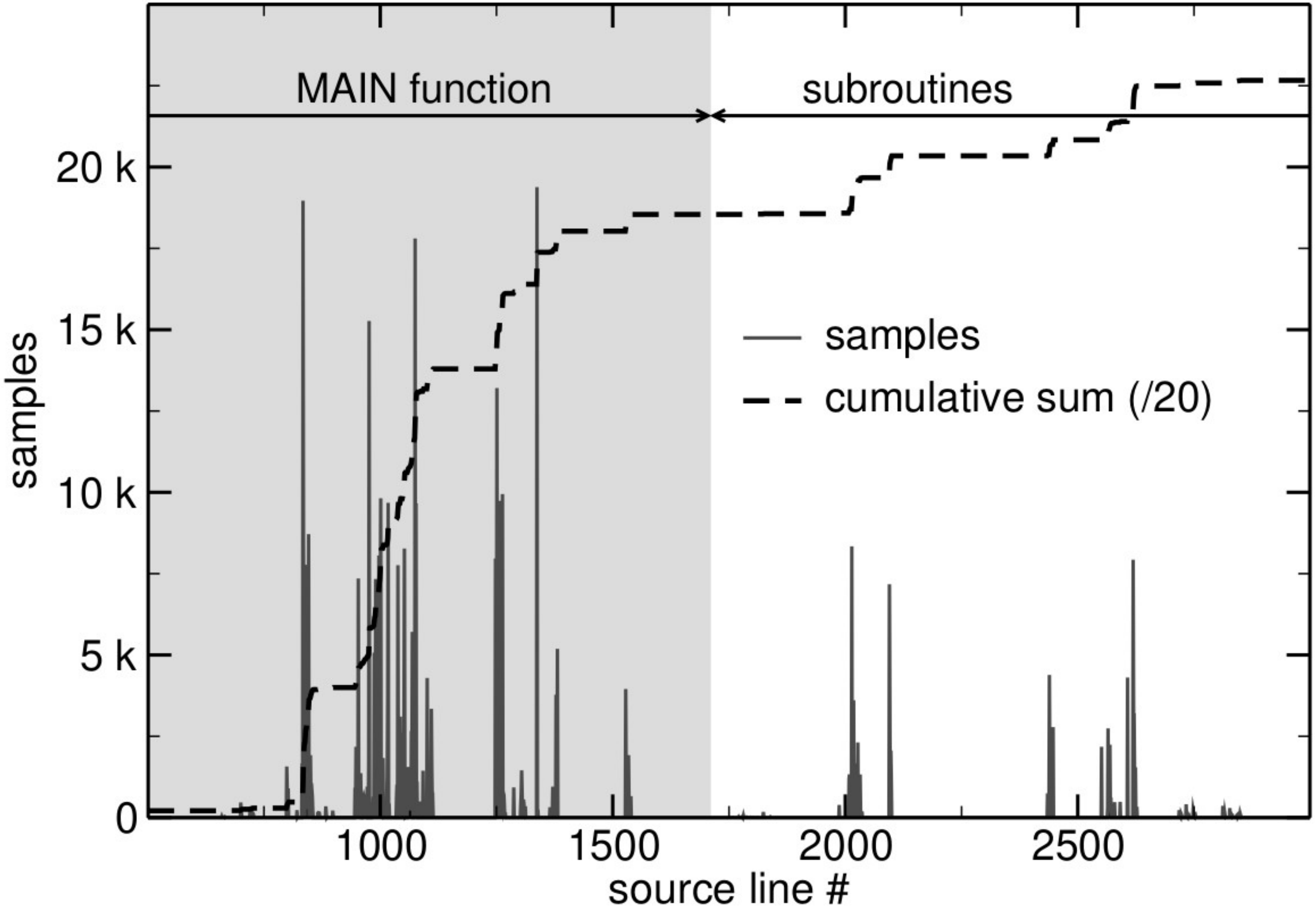
% time The percentage of overall runtime spent in this function, including its callees (inclusive time). This should be identical to the product of the number of calls and the time per call on the flat profile.

self For each indexed function, this is exclusive execution time (identical to flat profile). For its callers (callees), it denotes the inclusive time this function (each callee) contributed to each caller (this function).

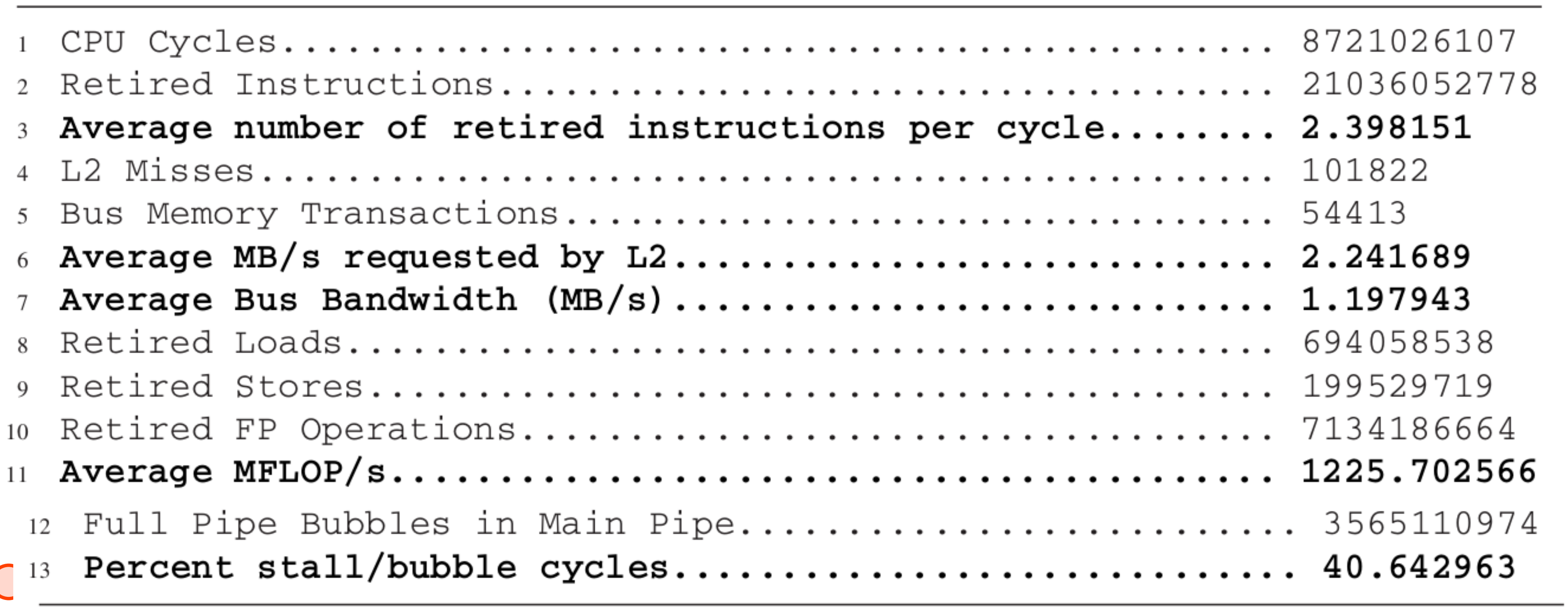
children For each indexed function, this is inclusive minus exclusive runtime, i.e., the contribution of all its callees to inclusive time. Part of this time contributes to inclusive runtime of each of the function's callers and is denoted in the respective caller rows. The callee rows in this column designate the contribution of each callee's callees to the function's inclusive runtime.

called denotes the number of times the function was called (probably split into recursive plus nonrecursive contributions, as shown in case of `shade()` above). Which fraction of the number of calls came from each caller is shown in the caller row, whereas the fraction of calls for each callee that was initiated from this function can be found in the callee rows.

Sampling histogram (solid) with number of samples vs. source code line number.

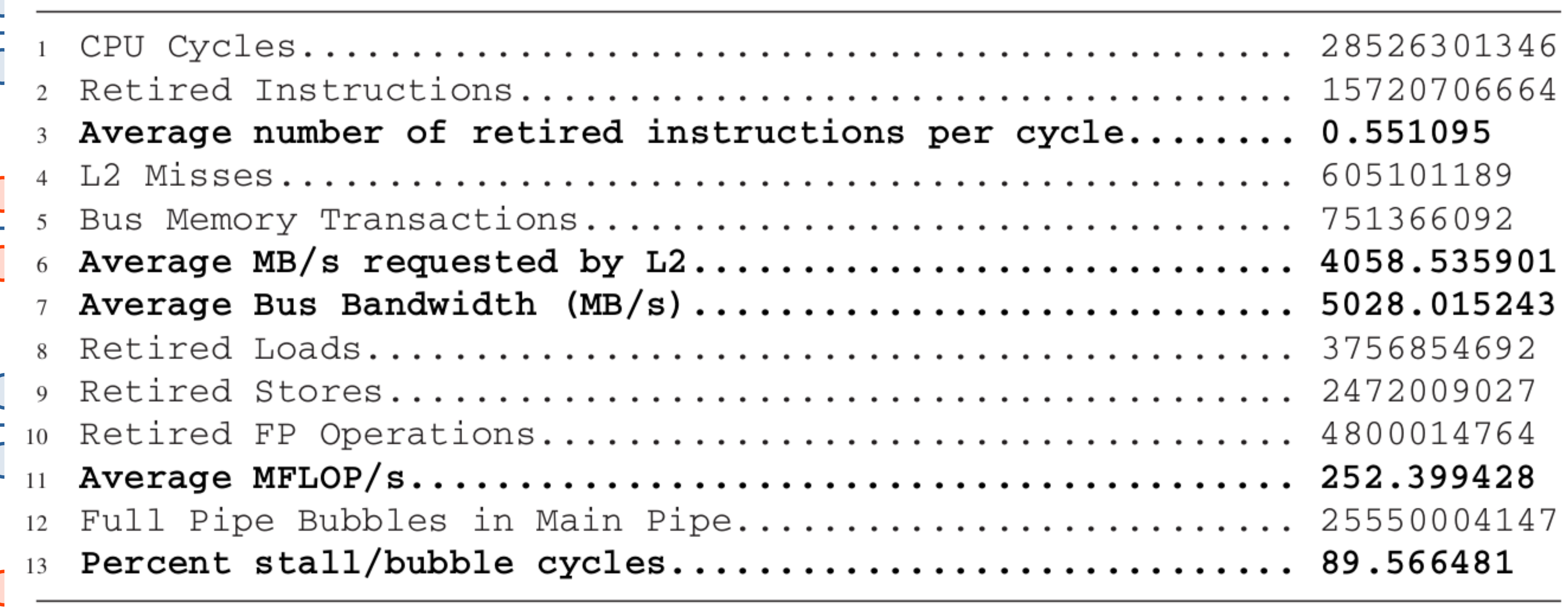


2.1.2 Hardware performance counters



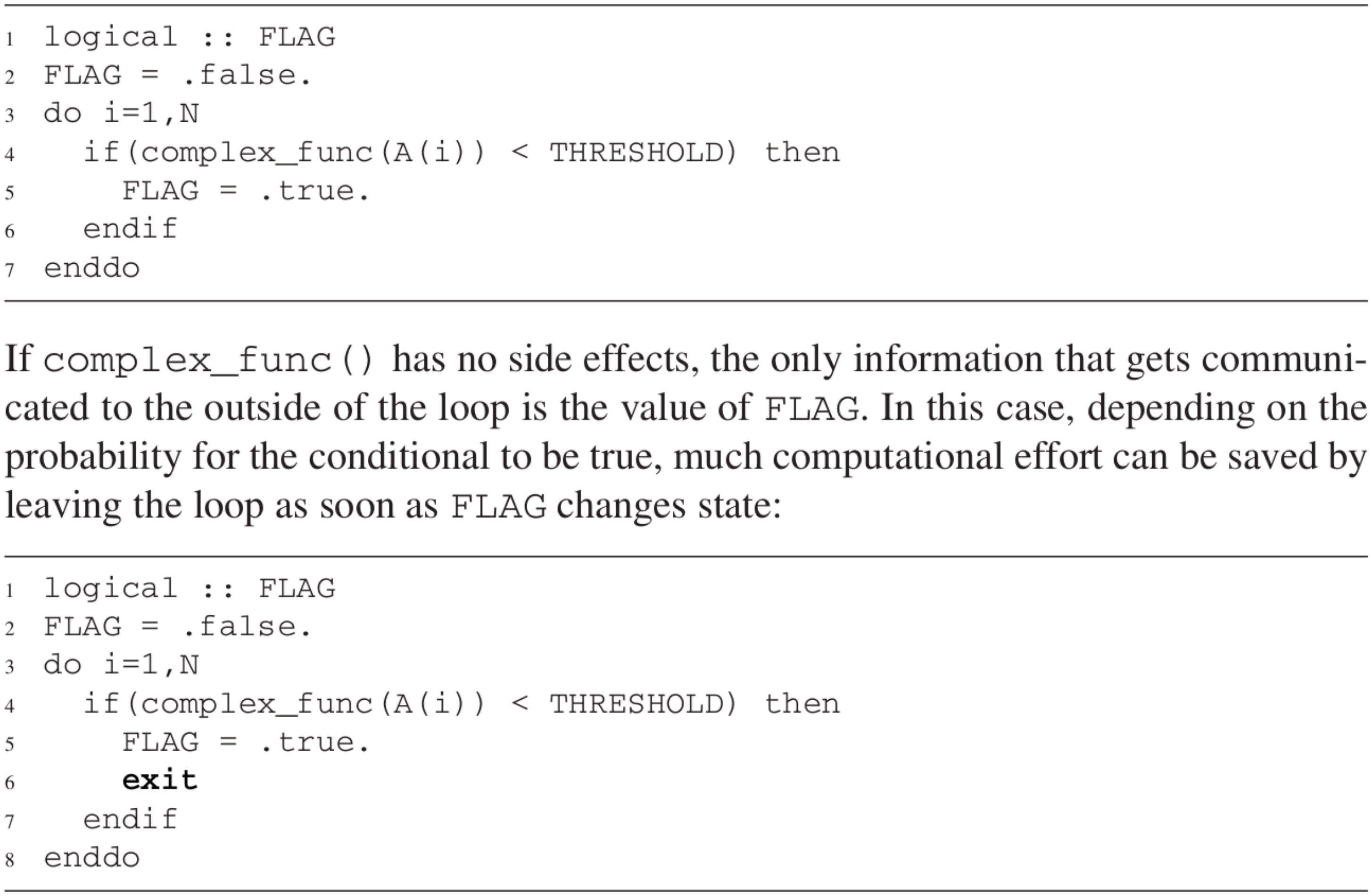
```
CPU Cycles.......................................... 8721026107
Retired Instructions................................ 21036052778
Average number of retired instructions per cycle........ 2.398151
L2 Misses........................................... 101822
Bus Memory Transactions............................. 54413
Average MB/s requested by L2........................ 2.241689
Average Bus Bandwidth (MB/s)........................ 1.197943
Retired Loads....................................... 694058538
Retired Stores...................................... 199529719
Retired FP Operations............................... 7134186664
Average MFLOP/s..................................... 1225.702566
Full Pipe Bubbles in Main Pipe...................... 3565110974
Percent stall/bubble cycles......................... 40.642963
```

2.1.2 Hardware performance counters



```
CPU Cycles................................................ 28526301346
Retired Instructions...................................... 15720706664
Average number of retired instructions per cycle.......... 0.551095
L2 Misses................................................. 605101189
Bus Memory Transactions................................... 751366092
Average MB/s requested by L2.............................. 4058.535901
Average Bus Bandwidth (MB/s).............................. 5028.015243
Retired Loads............................................. 3756854692
Retired Stores............................................ 2472009027
Retired FP Operations..................................... 4800014764
Average MFLOP/s........................................... 252.399428
Full Pipe Bubbles in Main Pipe............................ 25550004147
Percent stall/bubble cycles............................... 89.566481
```

2.2 Common sense optimizations: Do less work!

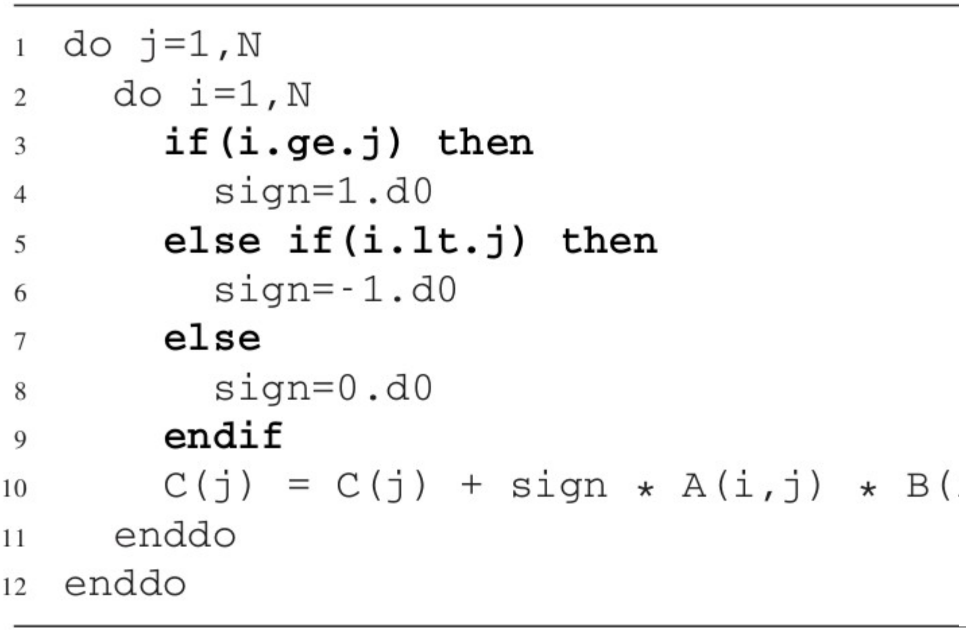


```
logical :: FLAG
FLAG = .false.
do i=1,N
  if(complex_func(A(i)) < THRESHOLD) then
    FLAG = .true.
  endif
enddo
```

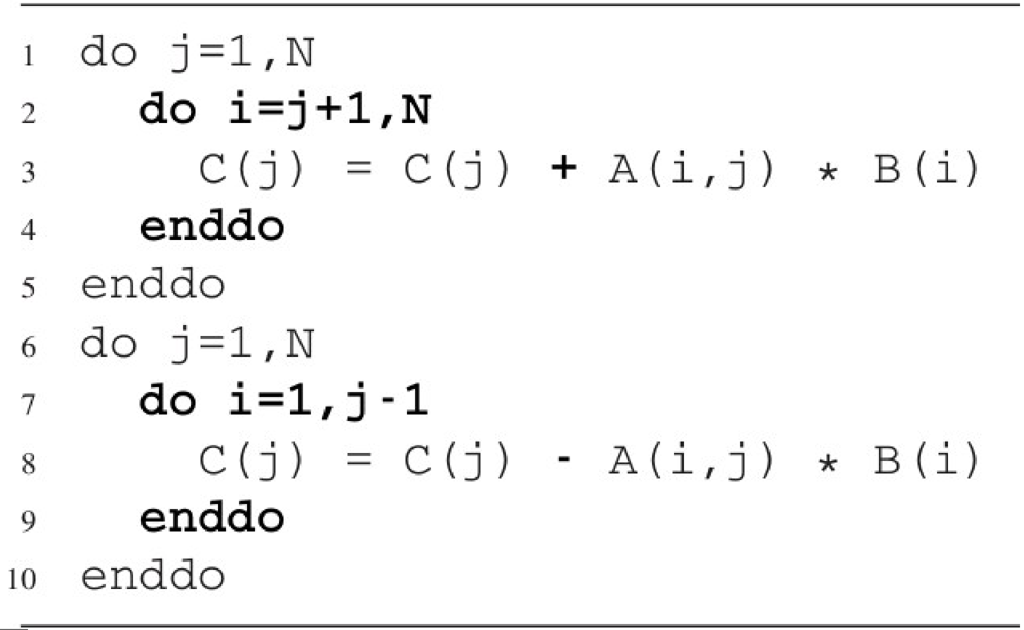
If complex_func() has no side effects, the only information that gets communicated to the outside of the loop is the value of FLAG. In this case, depending on the probability for the conditional to be true, much computational effort can be saved by leaving the loop as soon as FLAG changes state:

```
logical :: FLAG
FLAG = .false.
do i=1,N
  if(complex_func(A(i)) < THRESHOLD) then
    FLAG = .true.
    exit
  endif
enddo
```

2.3.2 Avoiding branches

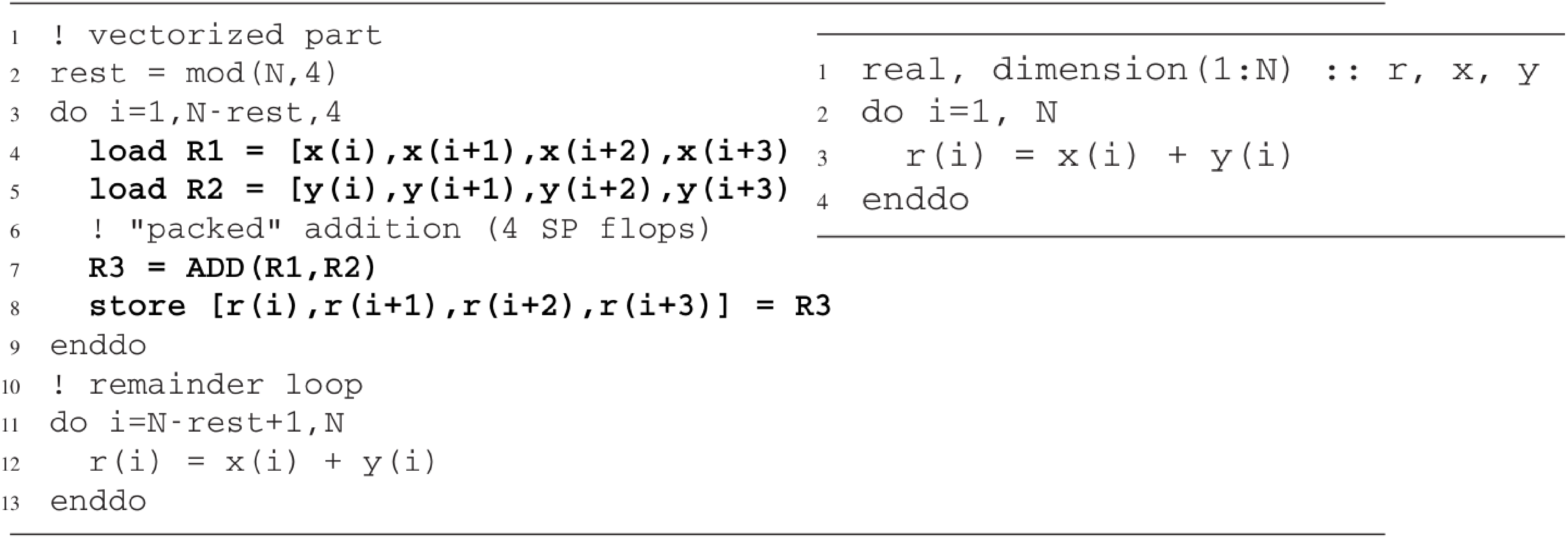


```
do j=1,N
  do i=1,N
    if(i.ge.j) then
      sign=1.d0
    else if(i.lt.j) then
      sign=-1.d0
    else
      sign=0.d0
    endif
    C(j) = C(j) + sign * A(i,j) * B(
  enddo
enddo
```



```
do j=1,N
  do i=j+1,N
    C(j) = C(j) + A(i,j) * B(i)
  enddo
enddo
do j=1,N
  do i=1,j-1
    C(j) = C(j) - A(i,j) * B(i)
  enddo
enddo
```

2.3.3 Using SIMD instruction sets



```
! vectorized part
rest = mod(N,4)
do i=1,N-rest,4
   load R1 = [x(i),x(i+1),x(i+2),x(i+3)]
   load R2 = [y(i),y(i+1),y(i+2),y(i+3)]
   ! "packed" addition (4 SP flops)
   R3 = ADD(R1,R2)
   store [r(i),r(i+1),r(i+2),r(i+3)] = R3
enddo
! remainder loop
do i=N-rest+1,N
   r(i) = x(i) + y(i)
enddo
```

```
real, dimension(1:N) :: r, x, y
do i=1, N
   r(i) = x(i) + y(i)
enddo
```