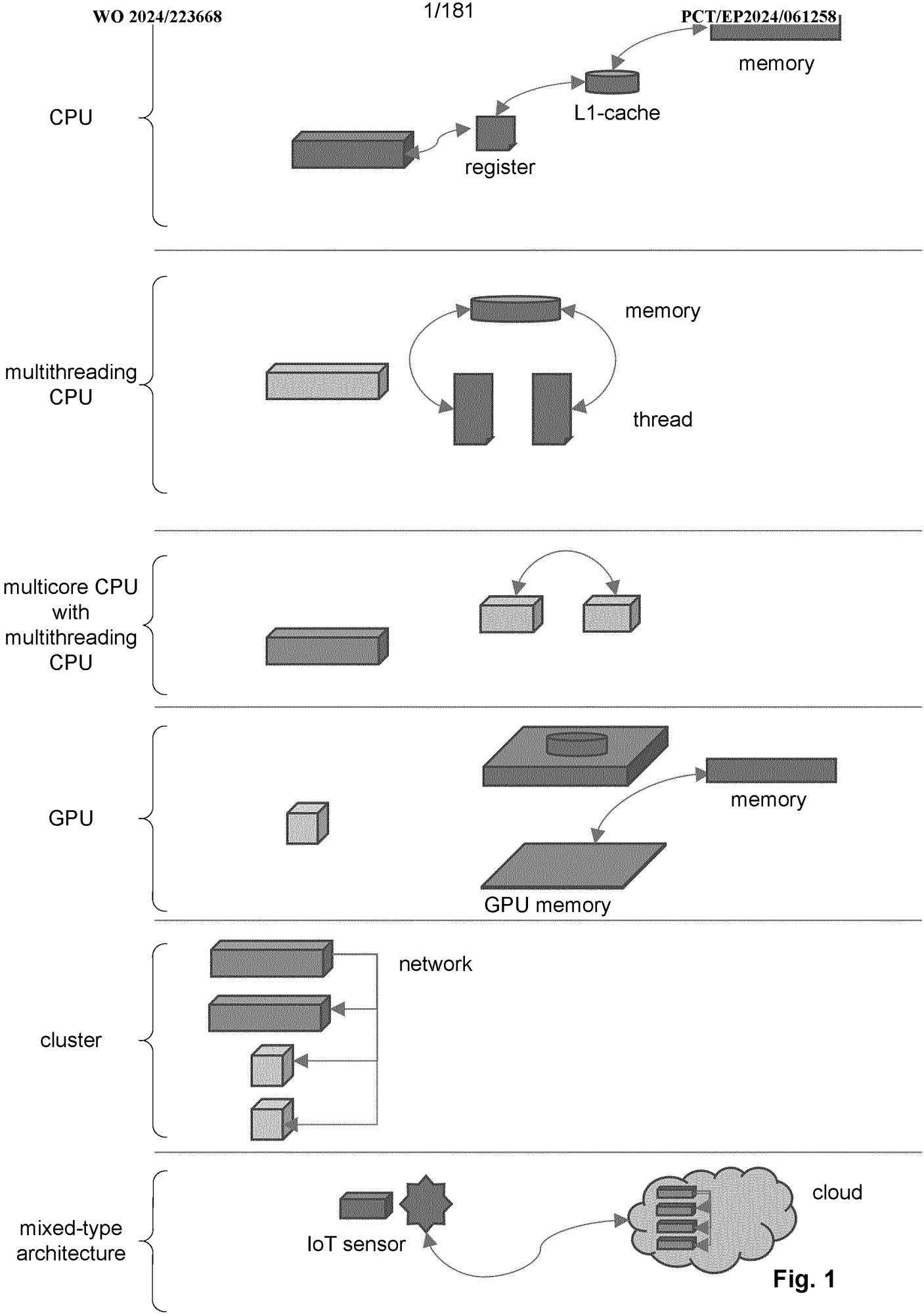

CPU
memory
L1-cache
register
multithreading CPU
memory
thread
multicore CPU with multithreading CPU
GPU
memory
GPU memory
cluster
network
mixed-type architecture
IoT sensor
cloud

Fig. 1

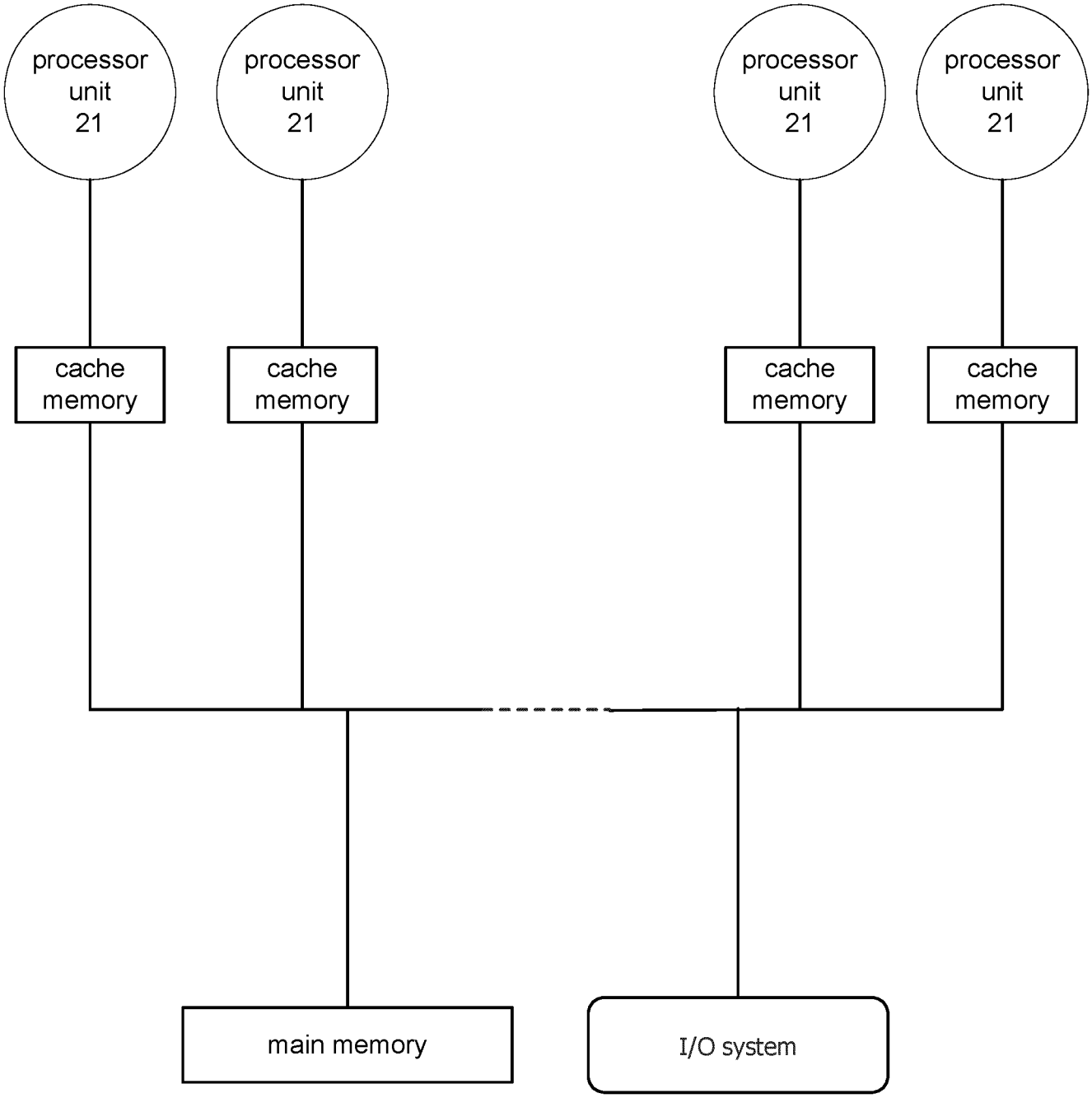

processor unit 21
processor unit 21
processor unit 21
processor unit 21
cache memory
cache memory
cache memory
cache memory
main memory
I/O system

Fig. 2

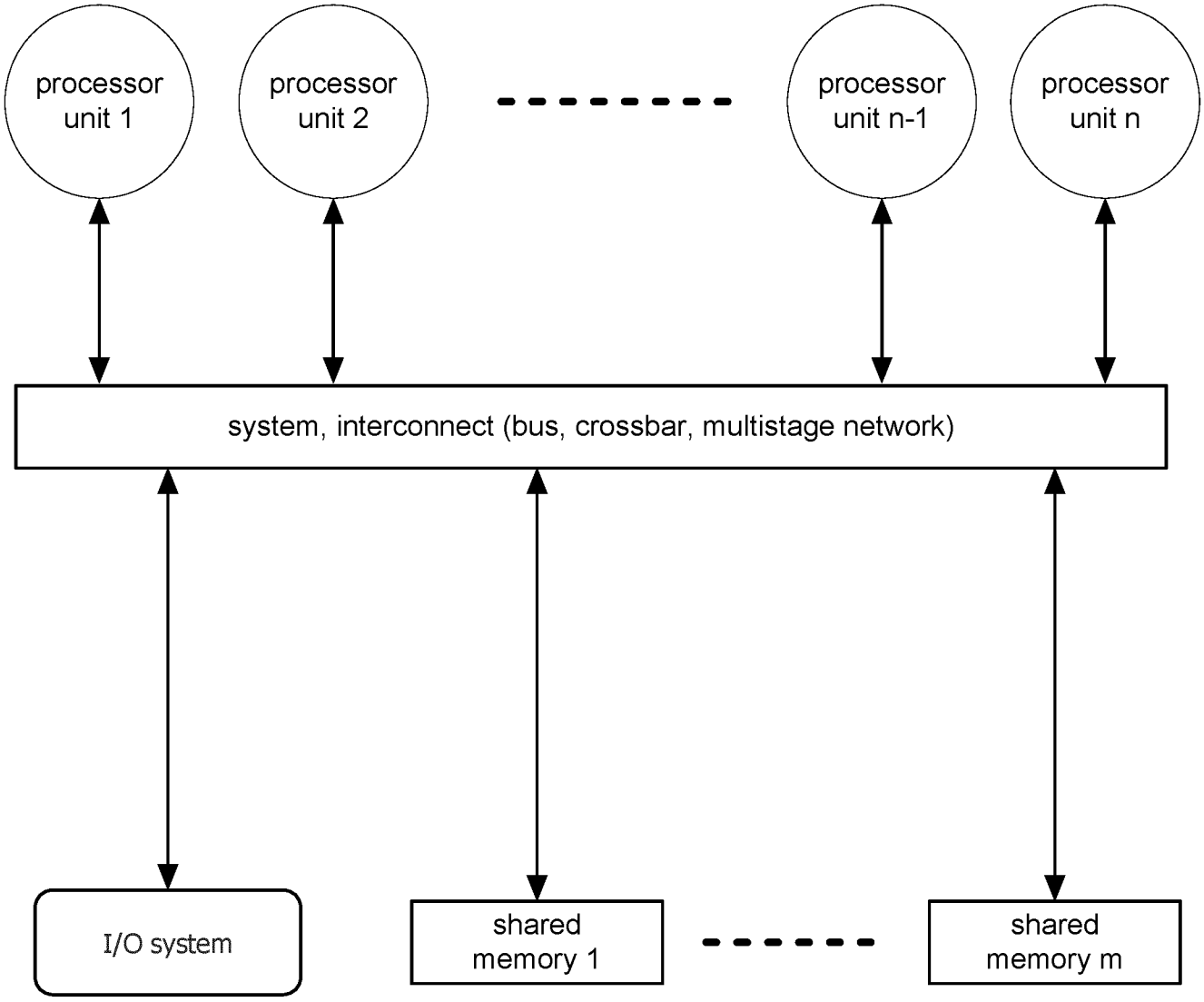

processor unit 1
processor unit 2
processor unit n-1
processor unit n
system, interconnect (bus, crossbar, multistage network)
I/O system
shared memory 1
shared memory m

Fig. 3

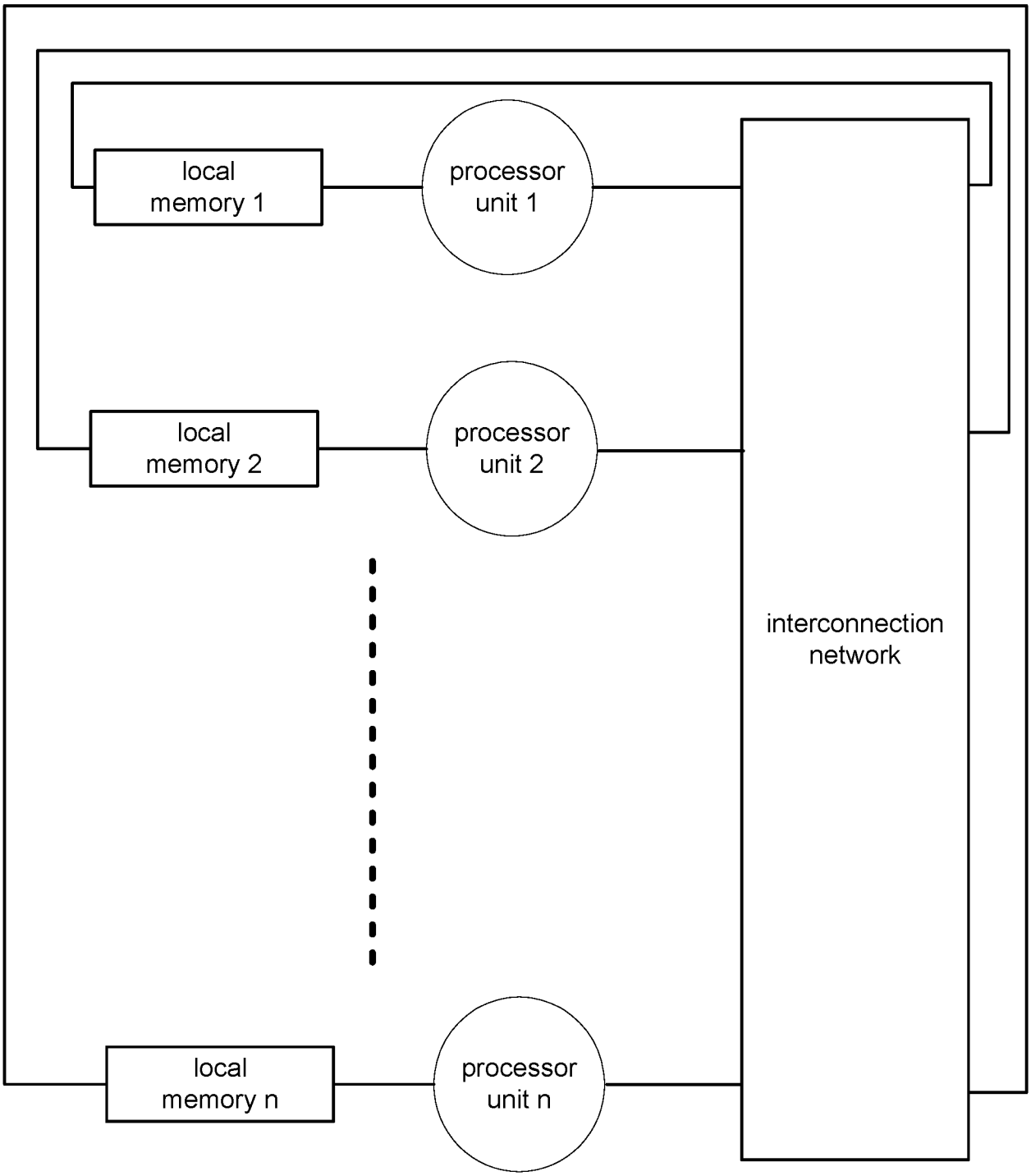

local
memory 1
processor
unit 1
local
memory 2
processor
unit 2
interconnection
network
local
memory n
processor
unit n

Fig. 4

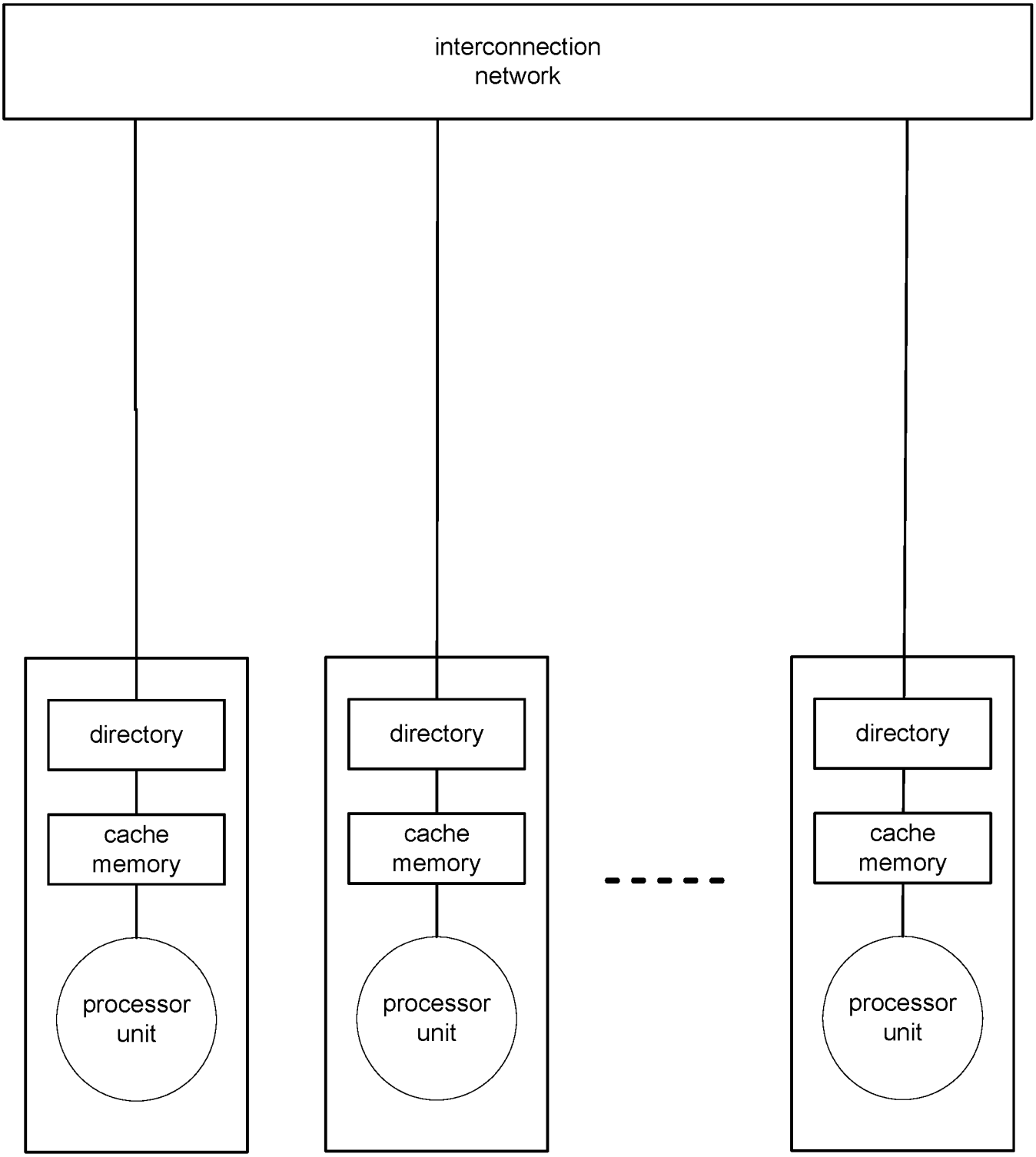

interconnection network
directory
cache memory
processor unit
directory
cache memory
processor unit
directory
cache memory
processor unit

Fig. 5

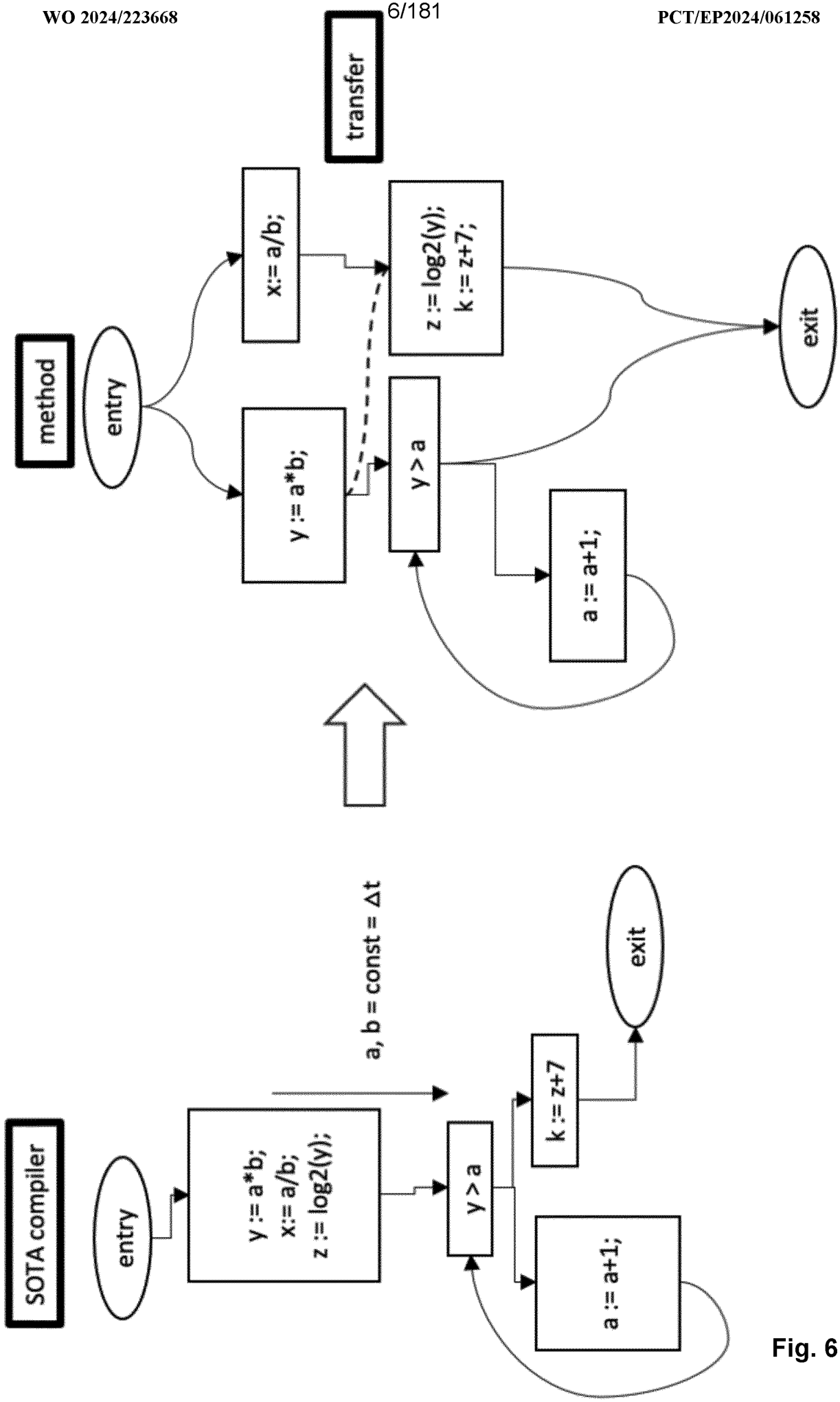

SOTA compiler
entry
y := a*b;
x:= a/b;
z := log2(y);
y > a
a := a+1;
k := z+7
exit
a, b = const = Δt
transfer
method
entry
y := a*b;
x:= a/b;
z := log2(y);
k := z+7;
y > a
a := a+1;
exit

Fig. 6

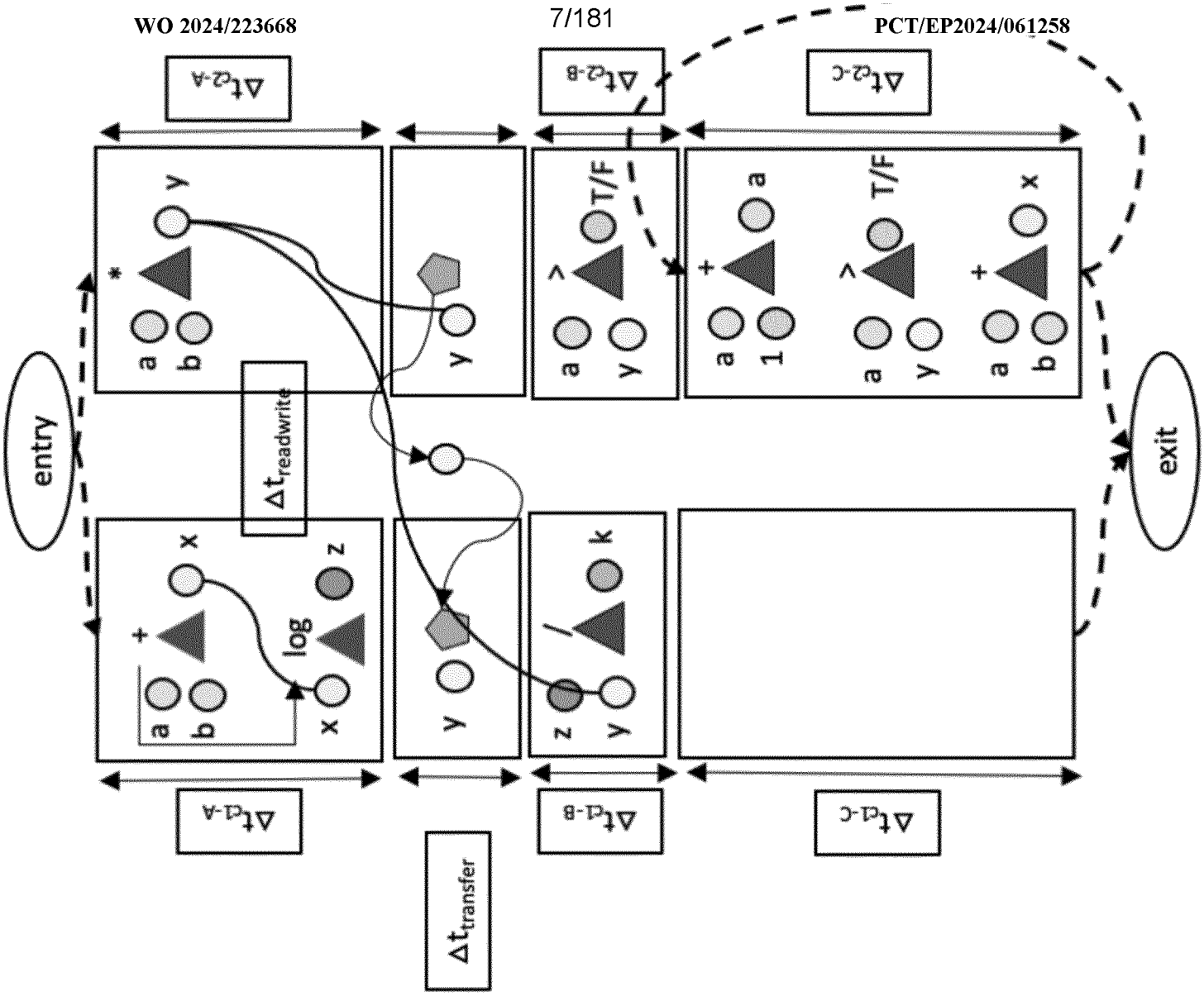

entry
exit
Δt_readwrite
Δt_transfer
Δt_c2-A
Δt_c2-B
Δt_c2-C
Δt_d-A
Δt_d-B
Δt_d-C
x := a+b;
y := a*b;
z := log2(x);
y > a
k := z/y;
a := a+1;
x := a+b;

Fig. 7

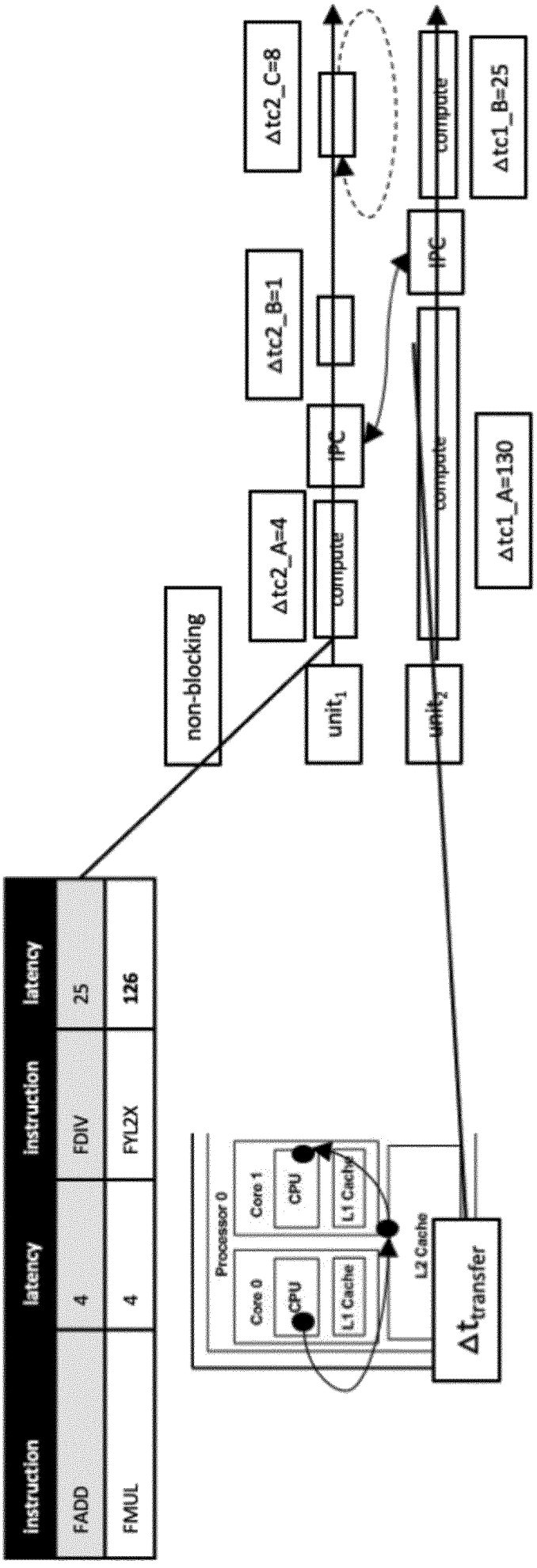

Δtc2_C=8
Δtc2_B=1
Δtc2_A=4
Δtc1_B=25
Δtc1_A=130
compute
IPC
non-blocking
unit1
unit2
Processor 0
Core 0
Core 1
CPU
L1 Cache
L2 Cache
Δttransfer
Instruction
latency
FADD
FMUL
FDIV
FYL2X
25
126

Fig. 8

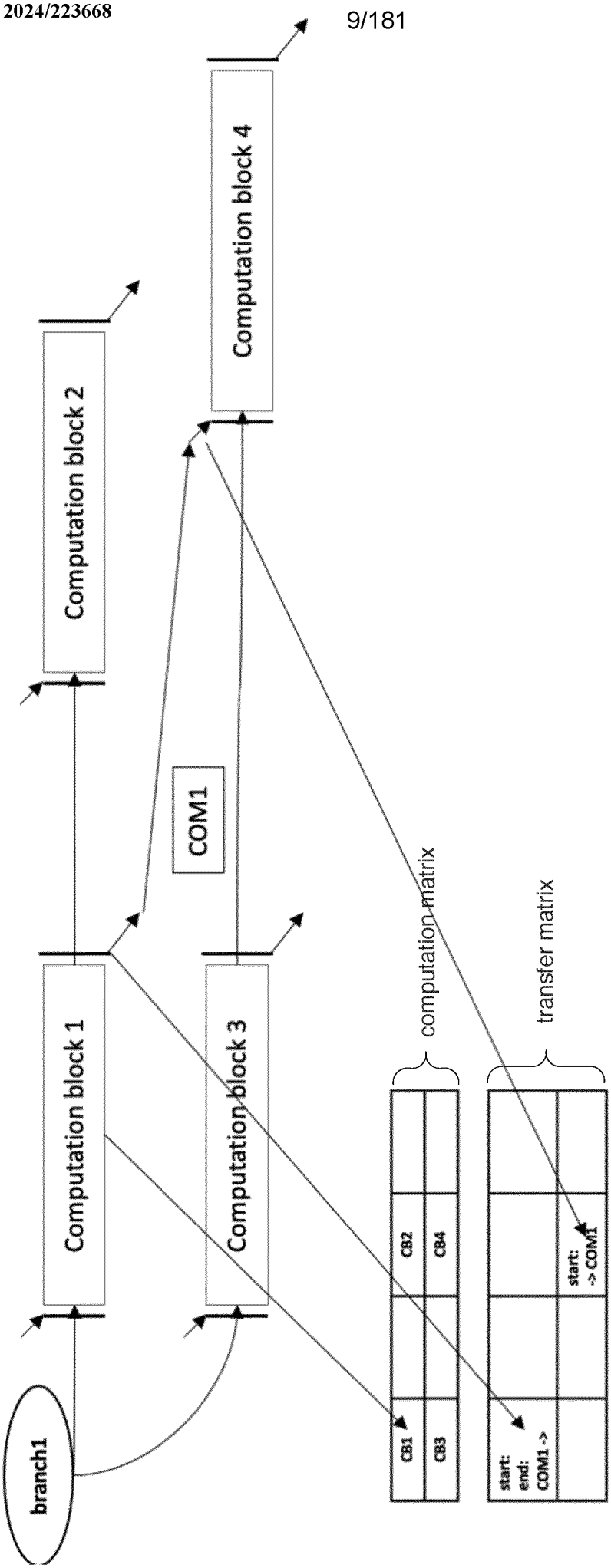

Computation block 4
Computation block 2
COM1
Computation block 1
Computation block 3
computation matrix
transfer matrix
CB2
CB4
CB1
CB3
start:
-> COM1
start:
end:
COM1 ->
branch1

Fig. 9

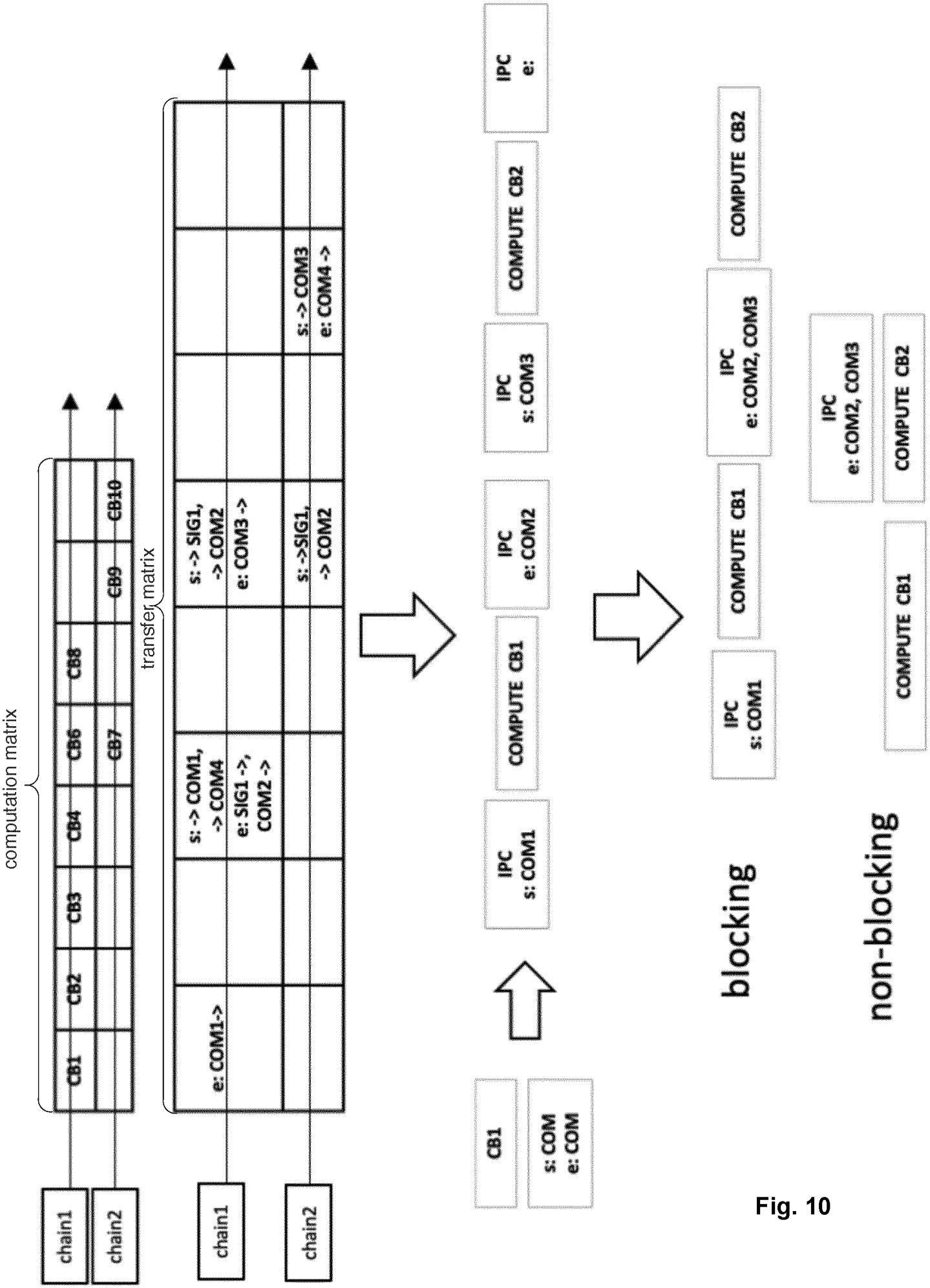
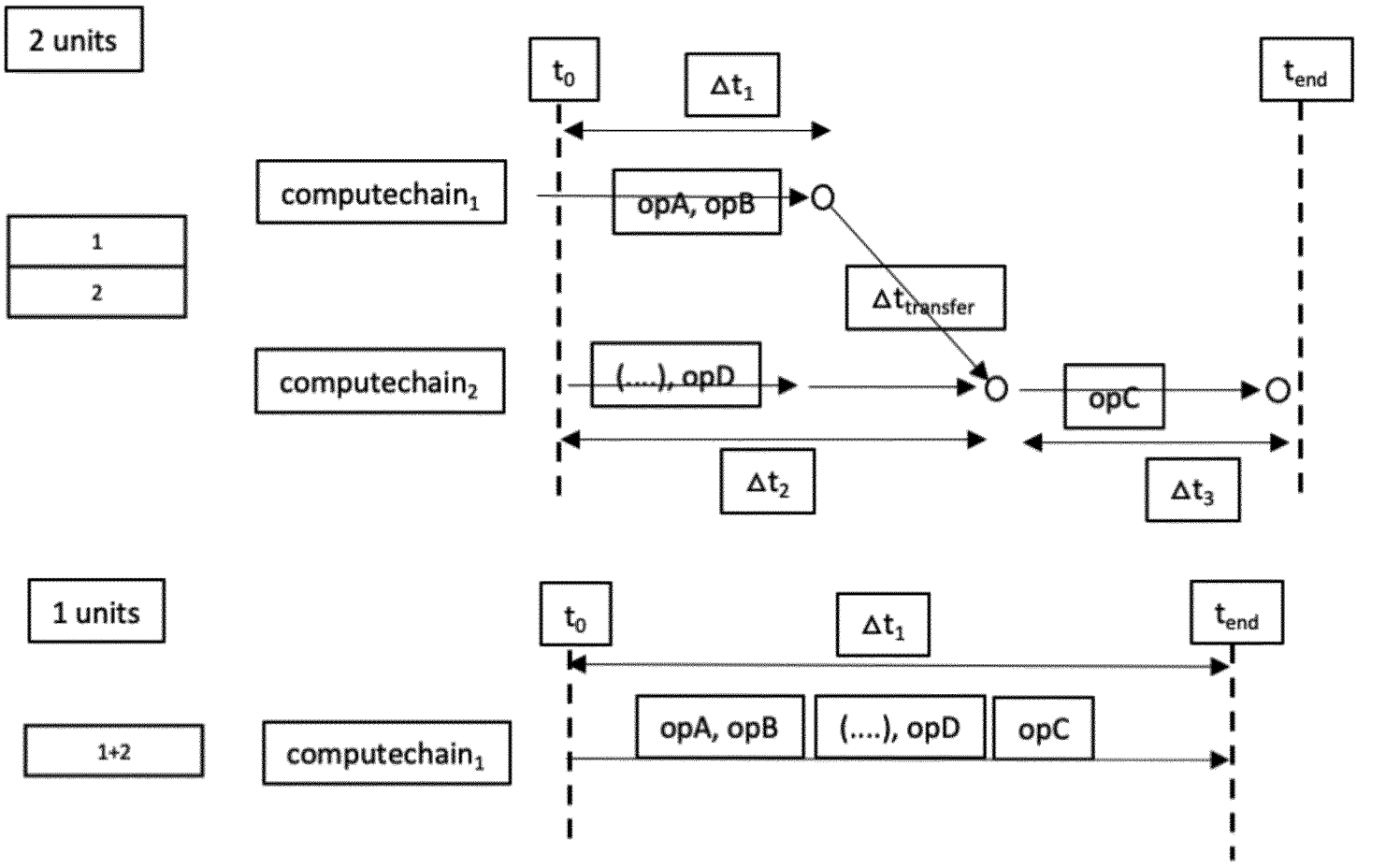

computation matrix
transfer matrix
CB1
CB2
CB3
CB4
CB6
CB8
CB7
CB9
CB10
chain1
chain2
e: COM1->
s: -> COM1, -> COM4
e: SIG1 ->, COM2 ->
s: -> SIG1, -> COM2
e: COM3 ->
s: ->SIG1, -> COM2
s: -> COM3
e: COM4 ->
chain1
chain2
CB1
s: COM
e: COM
IPC
s: COM1
COMPUTE CB1
IPC
e: COM2
IPC
s: COM3
COMPUTE CB2
IPC
e:
blocking
IPC
s: COM1
COMPUTE CB1
IPC
e: COM2, COM3
COMPUTE CB2
non-blocking
IPC
e: COM2, COM3
COMPUTE CB2
COMPUTE CB1

Fig. 10



computation Δt

communication $\Delta t_{transfer}$

$$t_{end1} = \max(\Delta t_1 + \Delta t_{transfer}, \Delta t_2) + \Delta t_3$$

$$t_{end2} = \Delta t_1$$

minimal runtime t: $\min(t_{end1}, t_{end2})$

Fig. 11

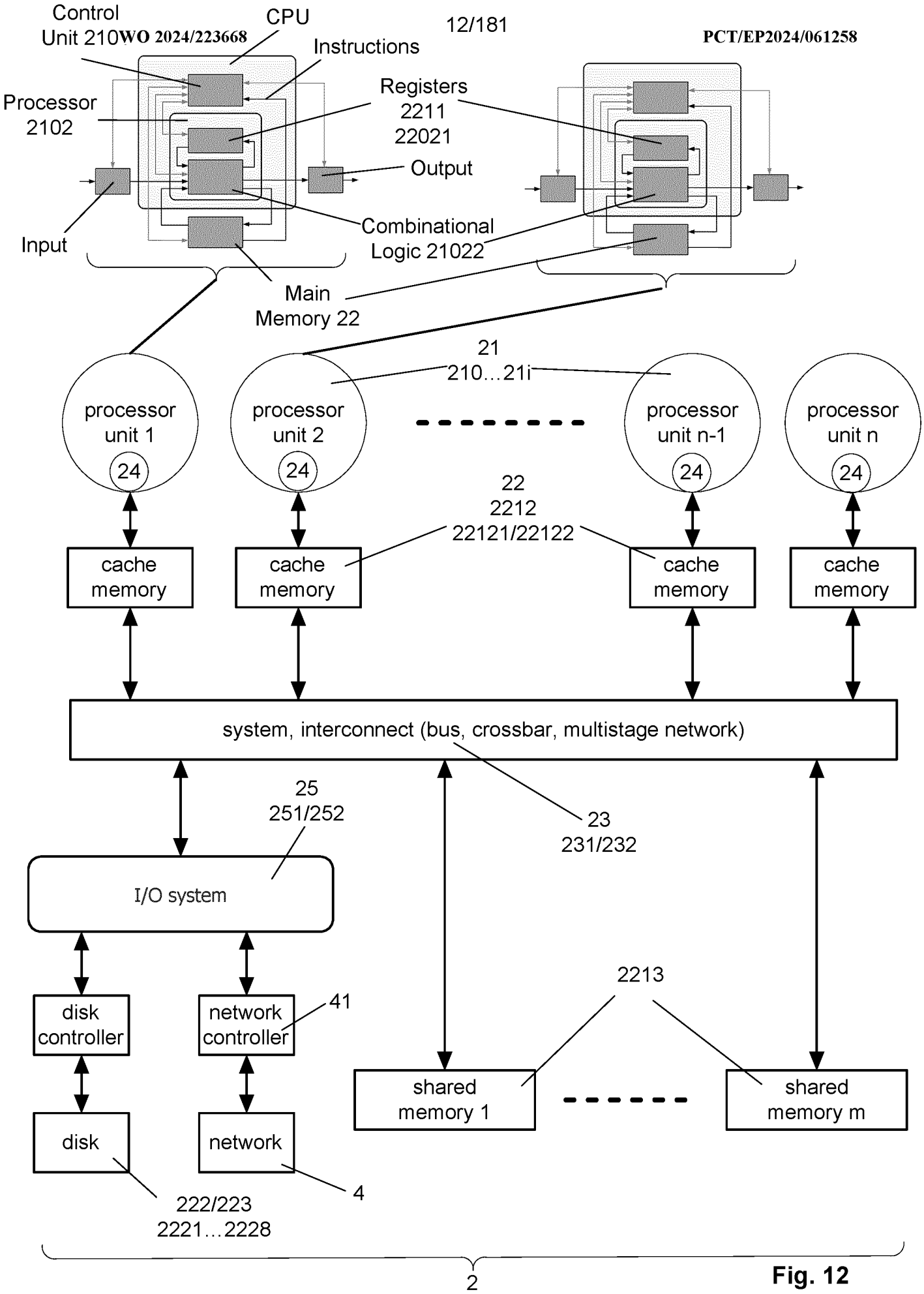

Control Unit 210
CPU
Instructions
Processor 2102
Registers 2211 22021
Output
Input
Combinational Logic 21022
Main Memory 22
21
210…21i
processor unit 1
processor unit 2
processor unit n-1
processor unit n
24
22
2212
22121/22122
cache memory
system, interconnect (bus, crossbar, multistage network)
25
251/252
23
231/232
I/O system
2213
disk controller
network controller
41
shared memory 1
shared memory m
disk
network
4
222/223
2221…2228
2

Fig. 12

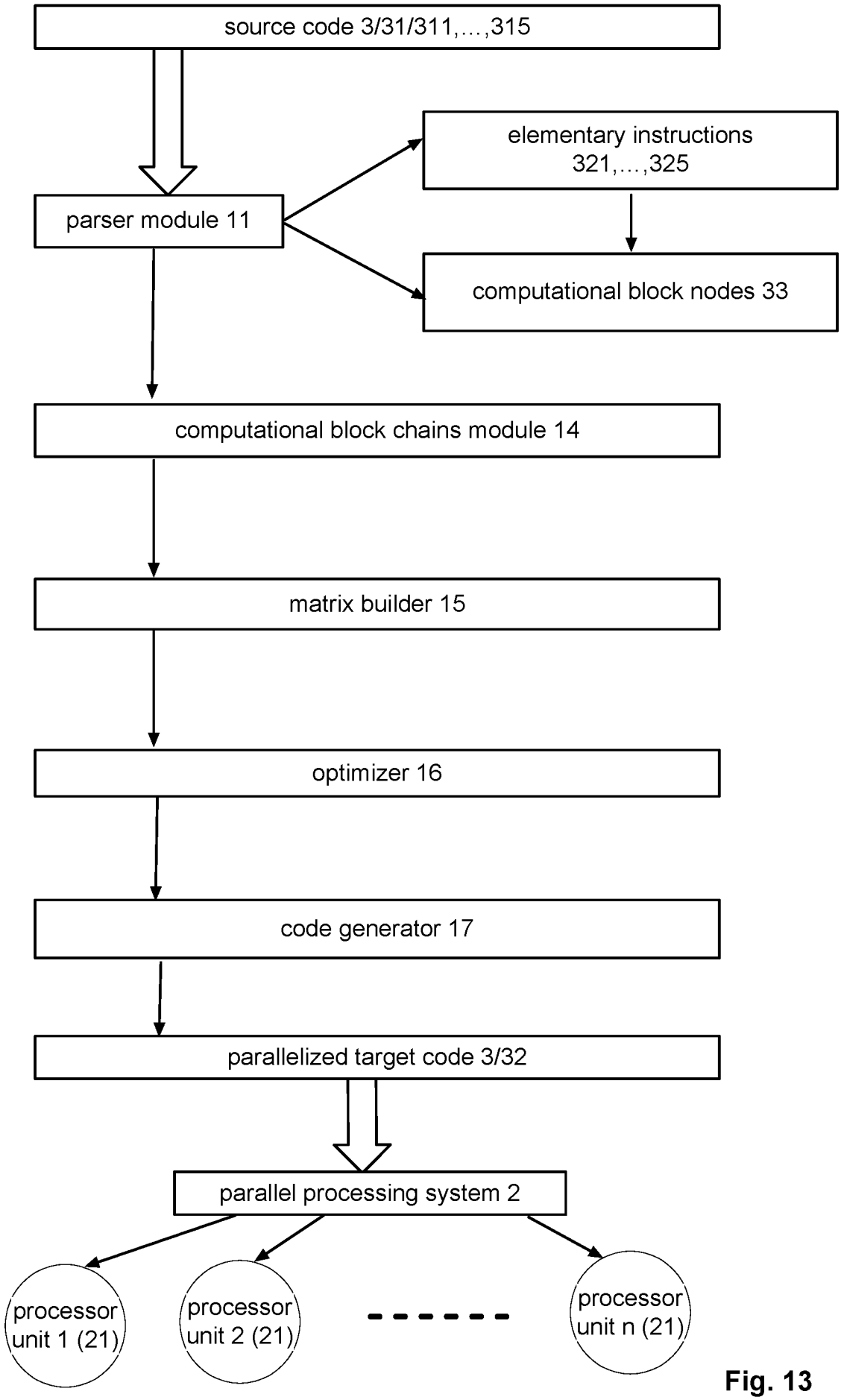

source code 3/31/311,…,315
elementary instructions 321,…,325
parser module 11
computational block nodes 33
computational block chains module 14
matrix builder 15
optimizer 16
code generator 17
parallelized target code 3/32
parallel processing system 2
processor unit 1 (21)
processor unit 2 (21)
processor unit n (21)

Fig. 13

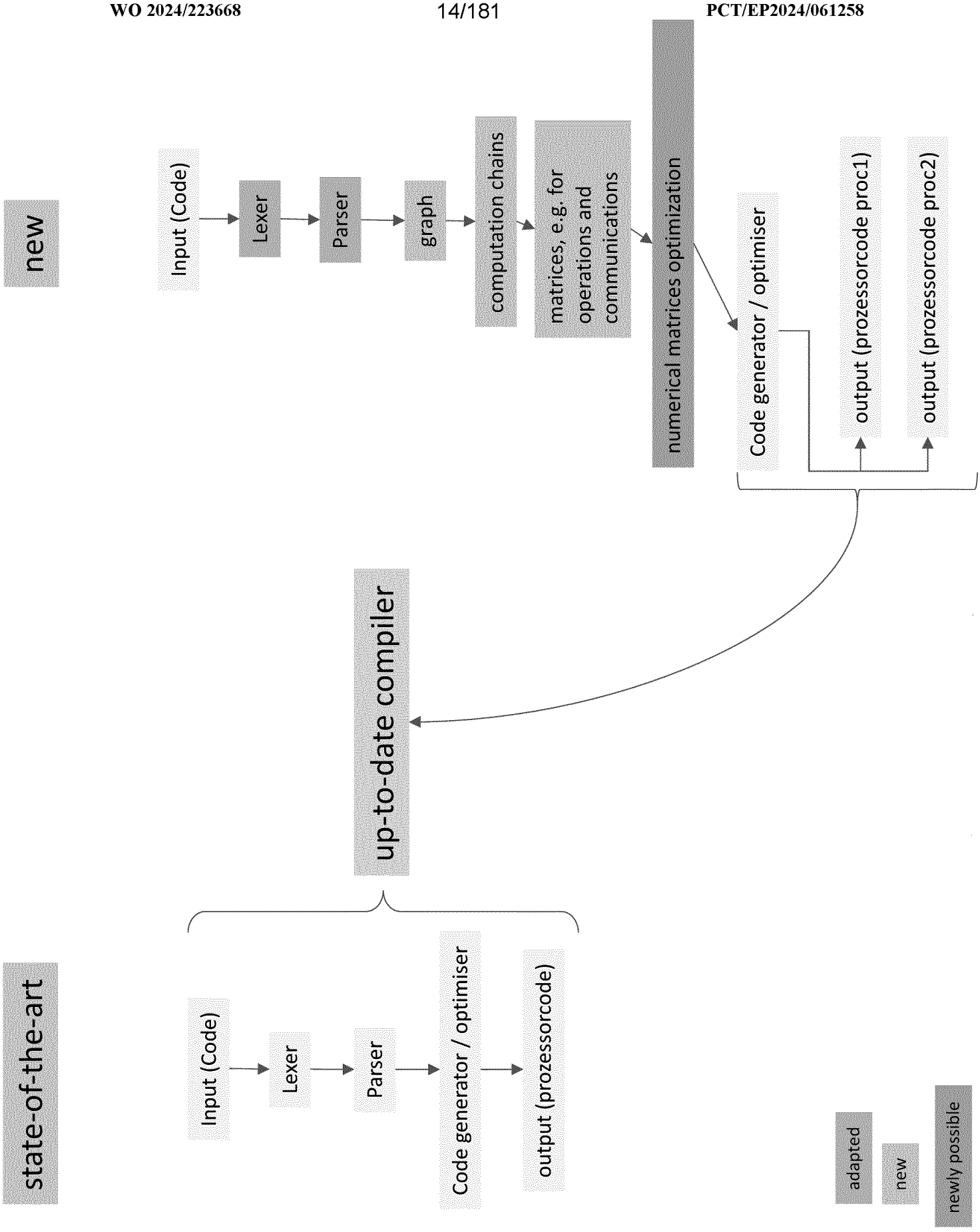

new
Input (Code)
Lexer
Parser
graph
computation chains
matrices, e.g. for operations and communications
numerical matrices optimization
Code generator / optimiser
output (prozessorcode proc1)
output (prozessorcode proc2)
up-to-date compiler
state-of-the-art
Input (Code)
Lexer
Parser
Code generator / optimiser
output (prozessorcode)
adapted
new
newly possible

Fig. 14

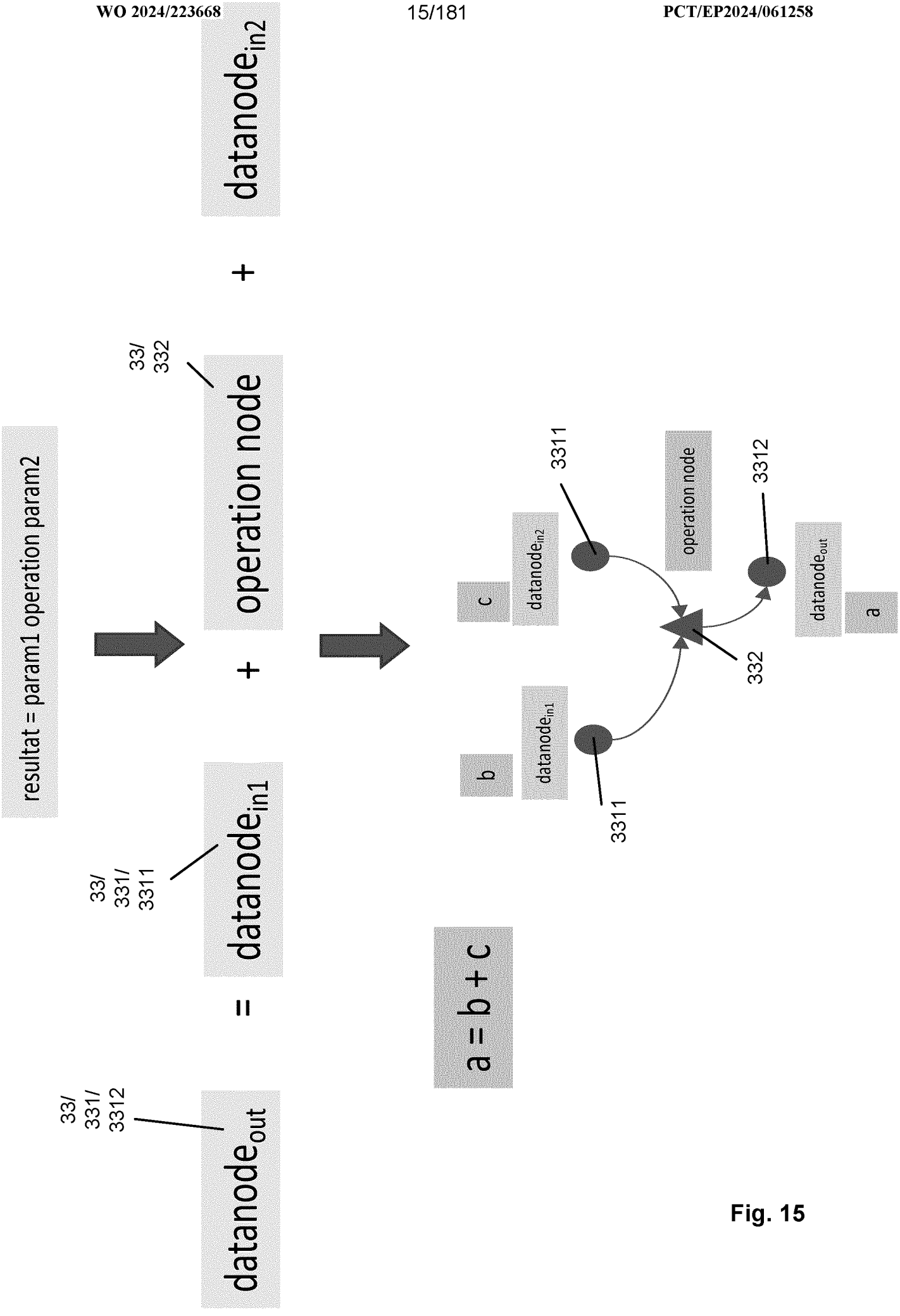
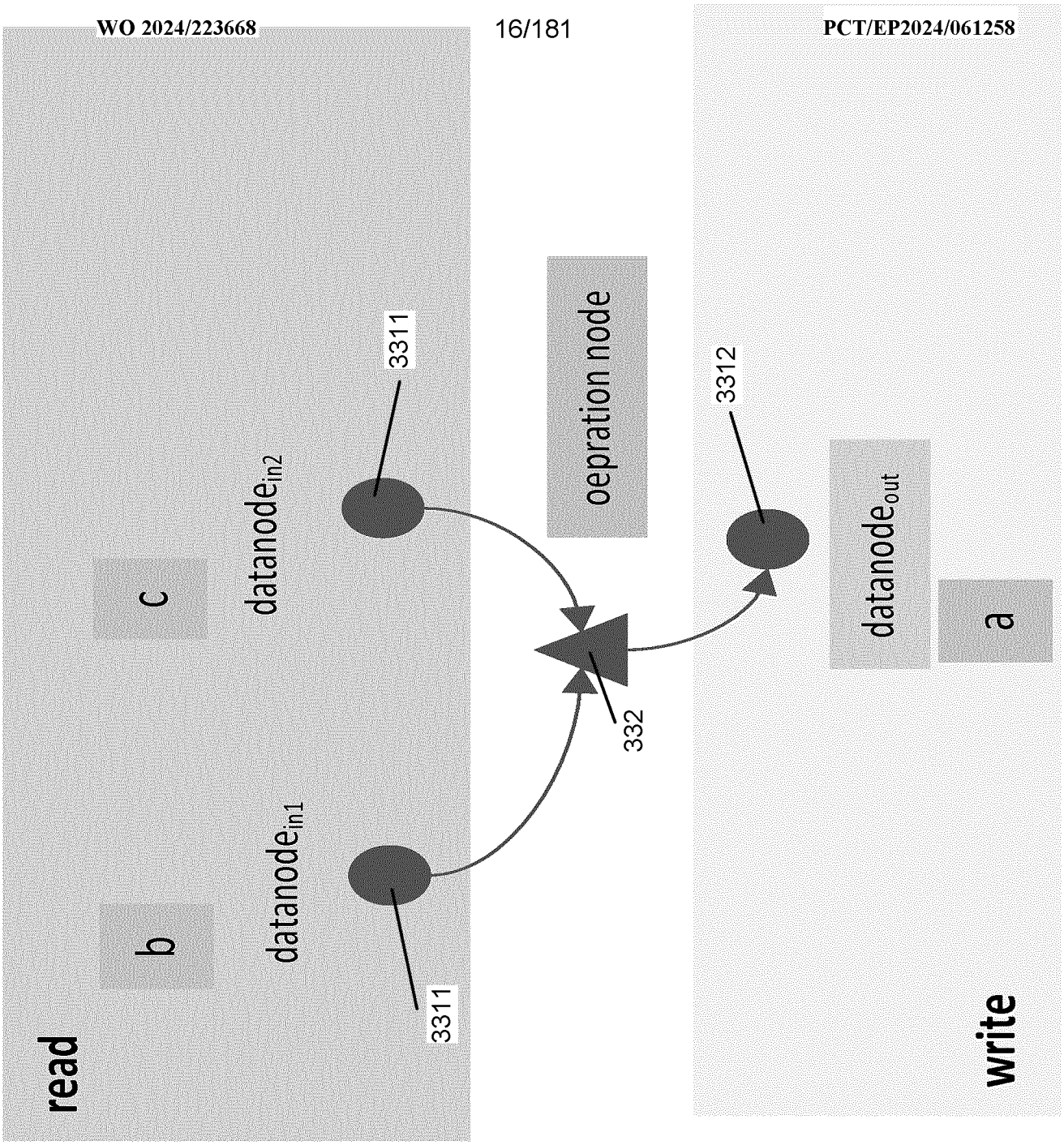
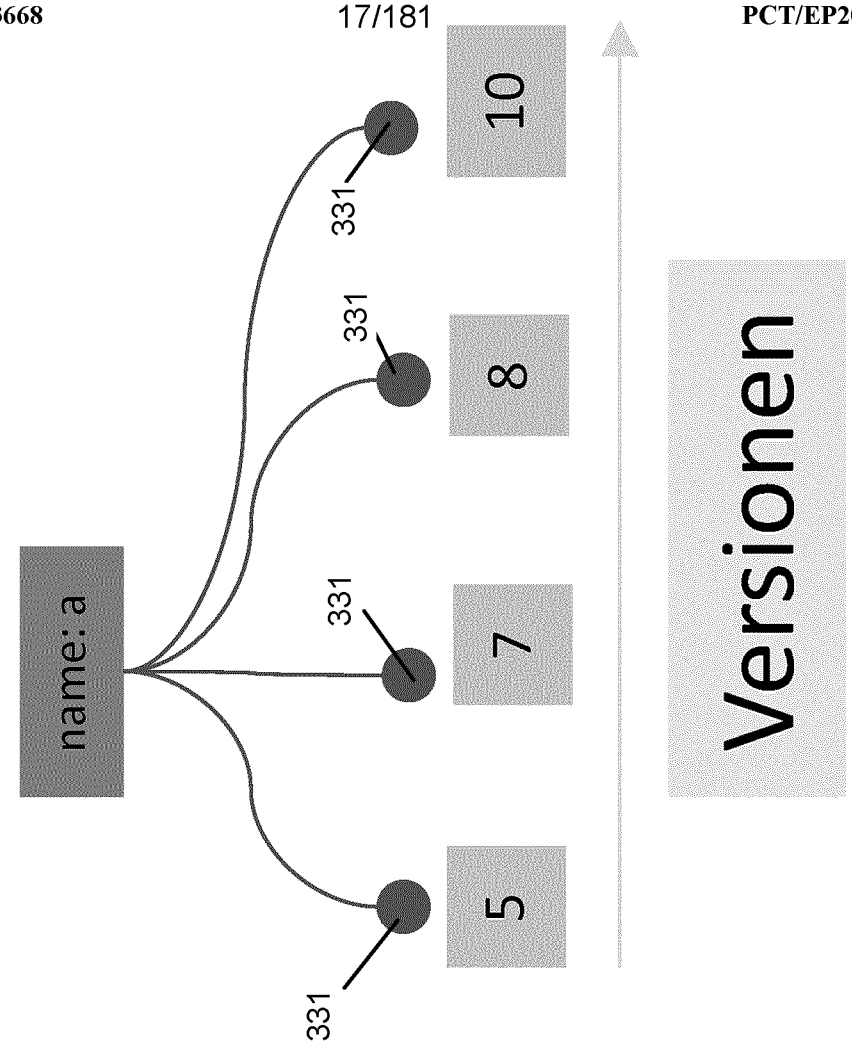

resultat = param1 operation param2
=
datanode$_{out}$
33/ 331/ 3312
datanode$_{in1}$
33/ 331/ 3311
+
operation node
33/ 332
+
datanode$_{in2}$
a = b + c
b
datanode$_{in1}$
3311
c
datanode$_{in2}$
3311
operation node
332
3312
datanode$_{out}$
a

Fig. 15


read
b
c
datanode$_{in1}$
datanode$_{in2}$
3311
3311
oepration node
332
3312
write
a
datanode$_{out}$

a = b + c

Fig. 16



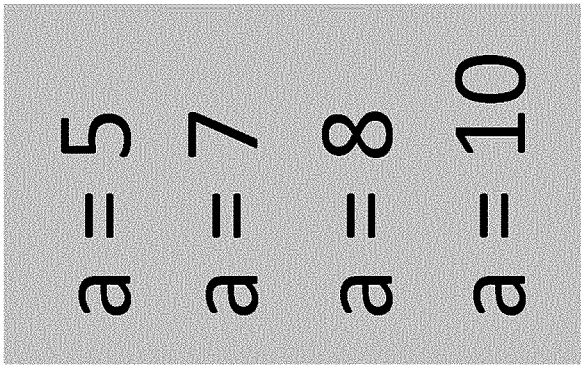


Fig. 17

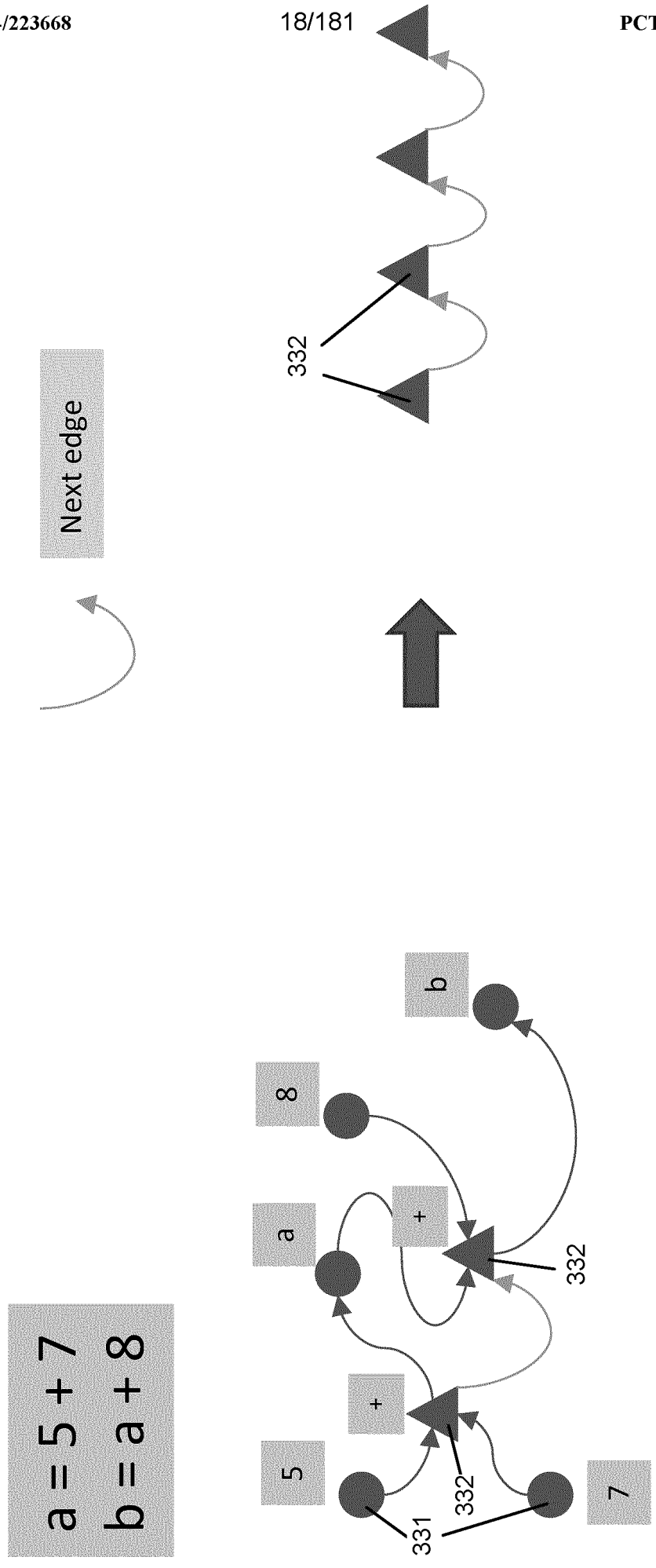


Fig. 18

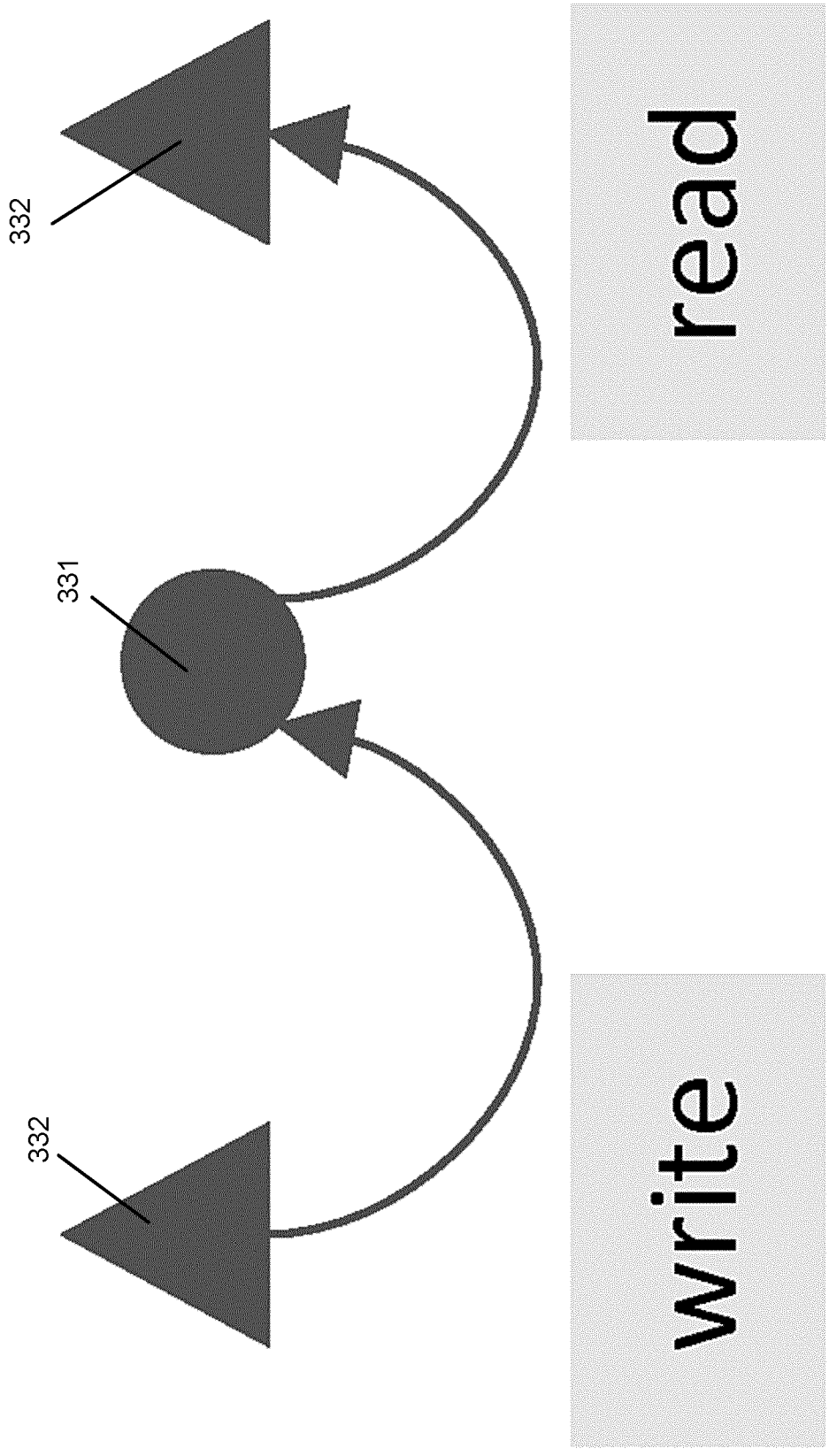
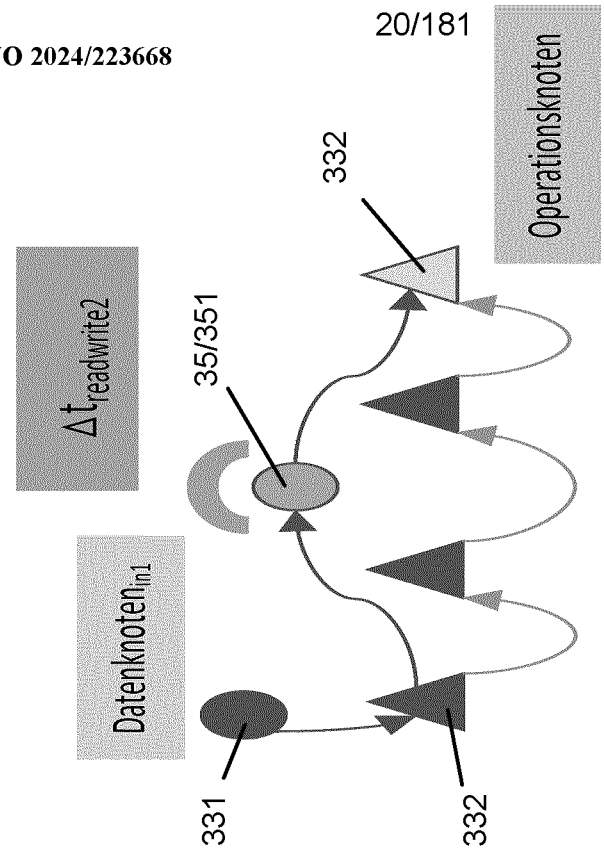


Fig. 19



$\Delta t_{readwrite}$ as a indicator what can be done with this data node

- $\Delta t_{readwrite}$ sehr klein: same cache_line or nearby
- $\Delta t_{readwrite}$ „mittel gross“: on same #cpu unit, use prefetch
- $\Delta t_{readwrite}$ „genug gross“: possible to transfer over network

Fig. 20

0-dependency
a = 8 + 7
8
+
7
a

1-dependency
b = 8
a = b + 7
8
=
b
+
7
a

2-dependency
c = 5 + 7
b = 7
a = b + c
7
=
b
+
c
+
7
5
331
332
351
331
a
Δt readwrite

Fig. 21

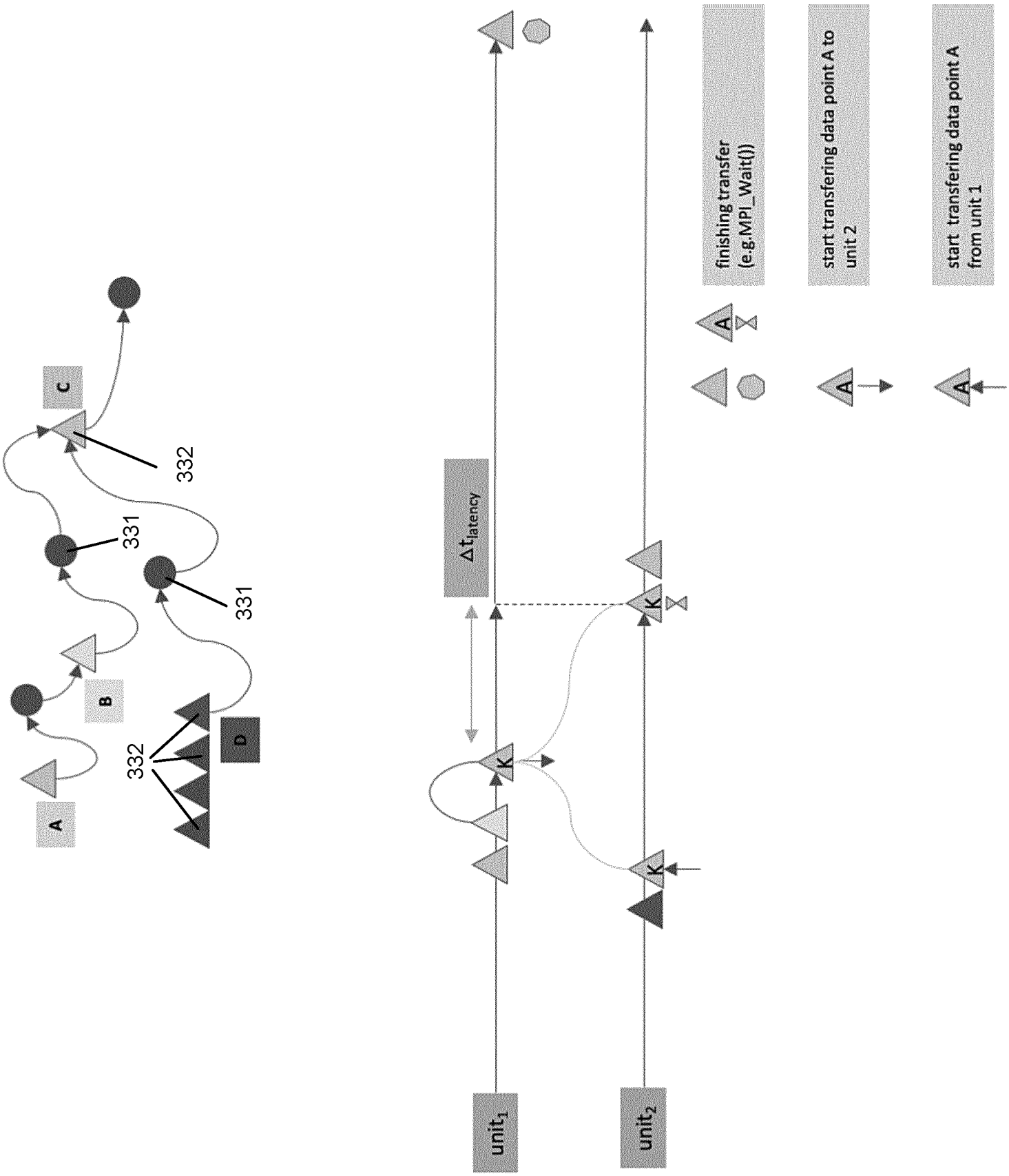
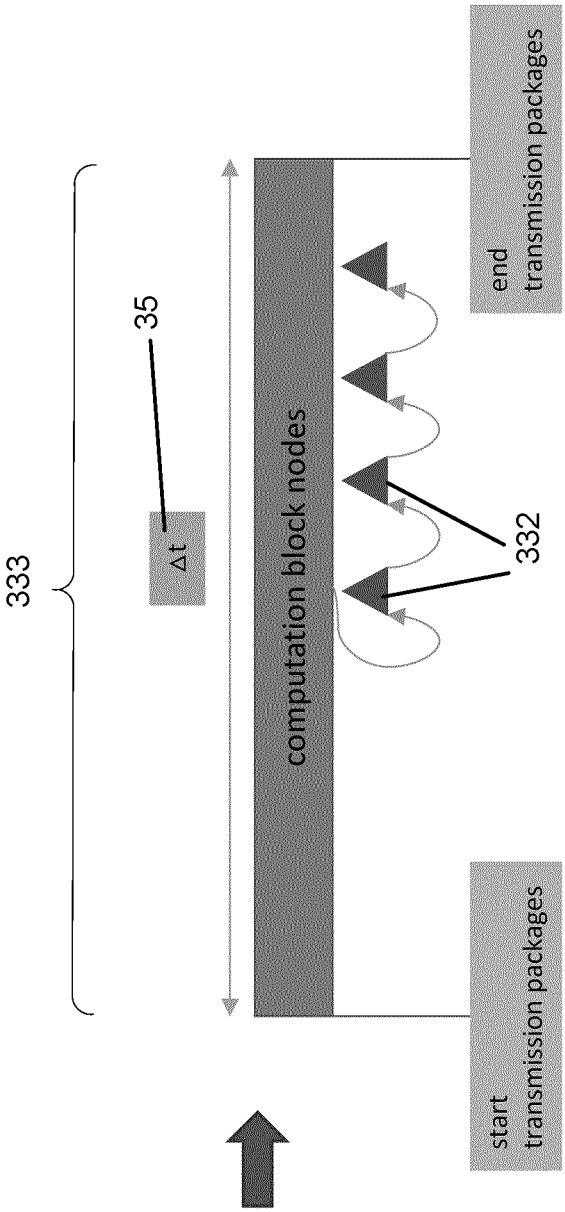

A
B
C
D
331
331
332
332
unit1
unit2
K
K
K
Δtlatency
finishing transfer (e.g.MPI_Wait())
start transfering data point A to unit 2
start transfering data point A from unit 1
A
A
A

Fig. 22


333
35
Δt
computation block nodes
332
end
transmission packages
start
transmission packages

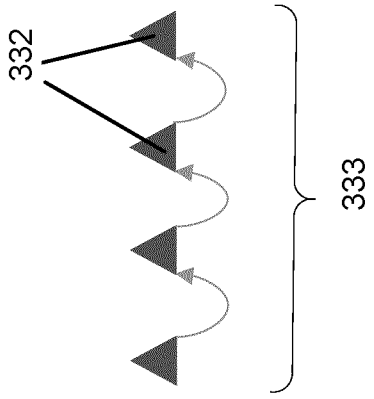

332
333

Fig. 23

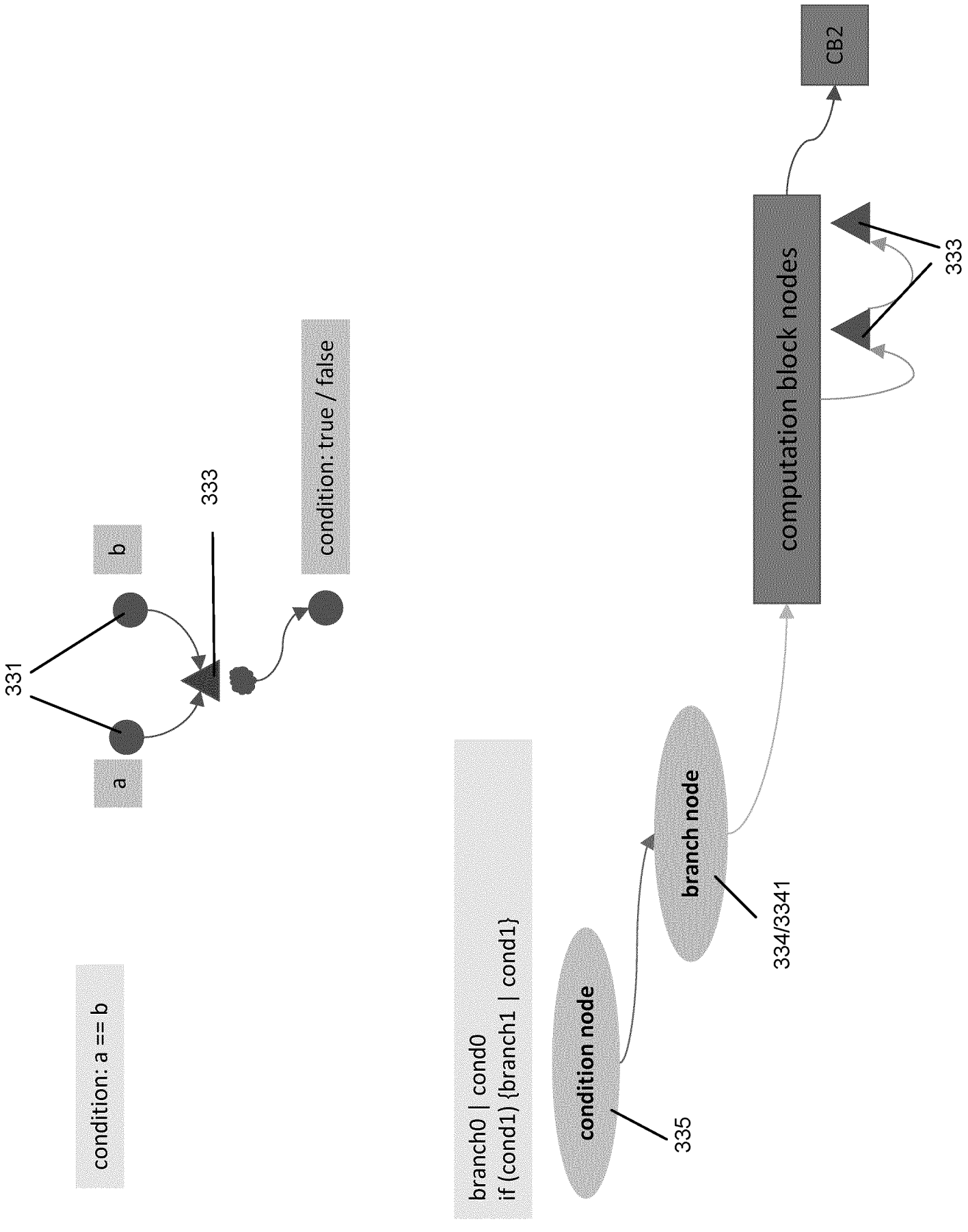

condition: a == b
a
b
331
333
condition: true / false
branch0 | cond0
if (cond1) {branch1 | cond1}
condition node
335
branch node
334/3341
computation block nodes
CB2
333

Fig. 24

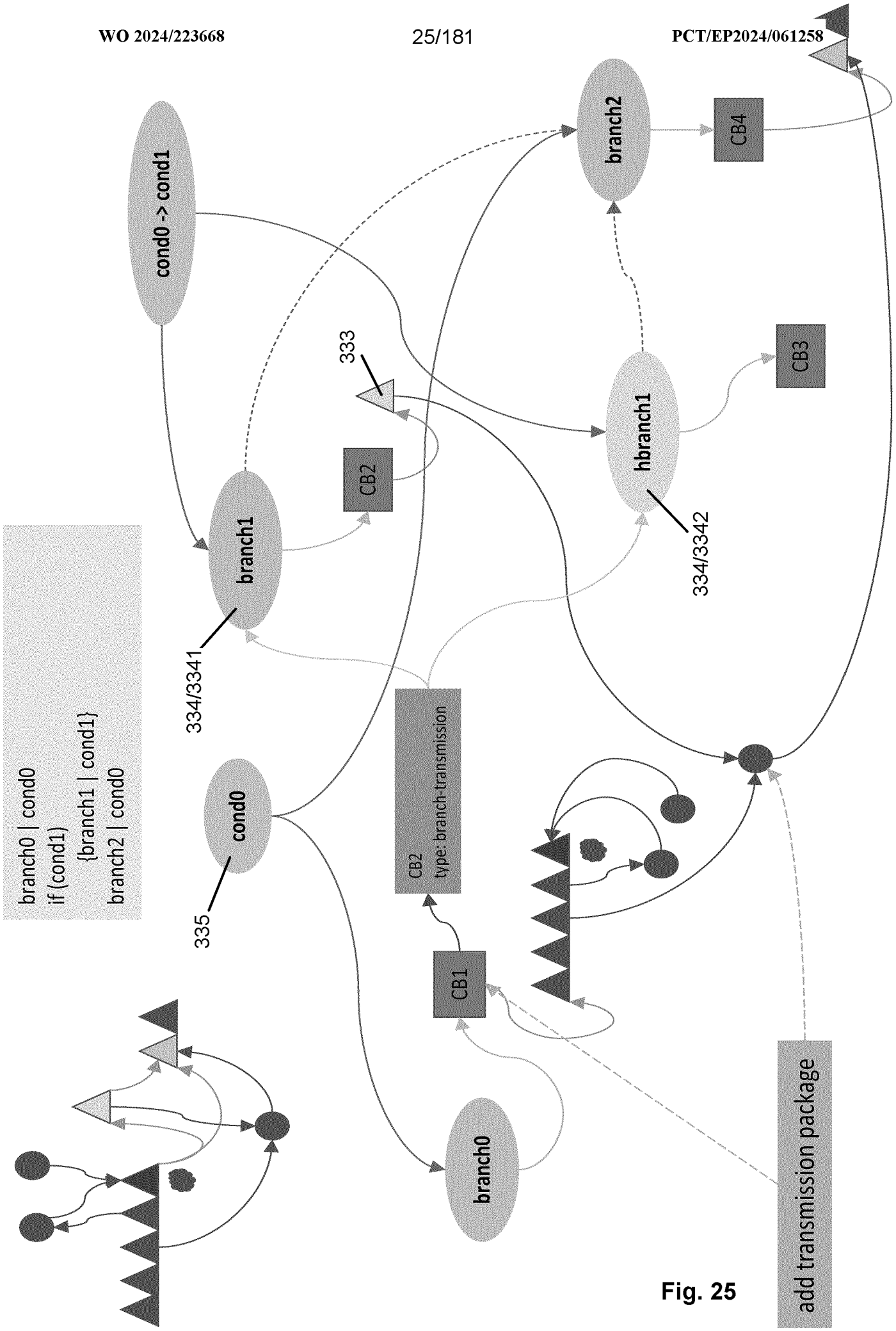

branch0 | cond0
if (cond1)
{branch1 | cond1}
branch2 | cond0
cond0 -> cond1
branch2
CB4
333
CB2
hbranch1
CB3
branch1
334/3342
334/3341
CB2
type: branch-transmission
cond0
335
CB1
branch0
add transmission package

Fig. 25

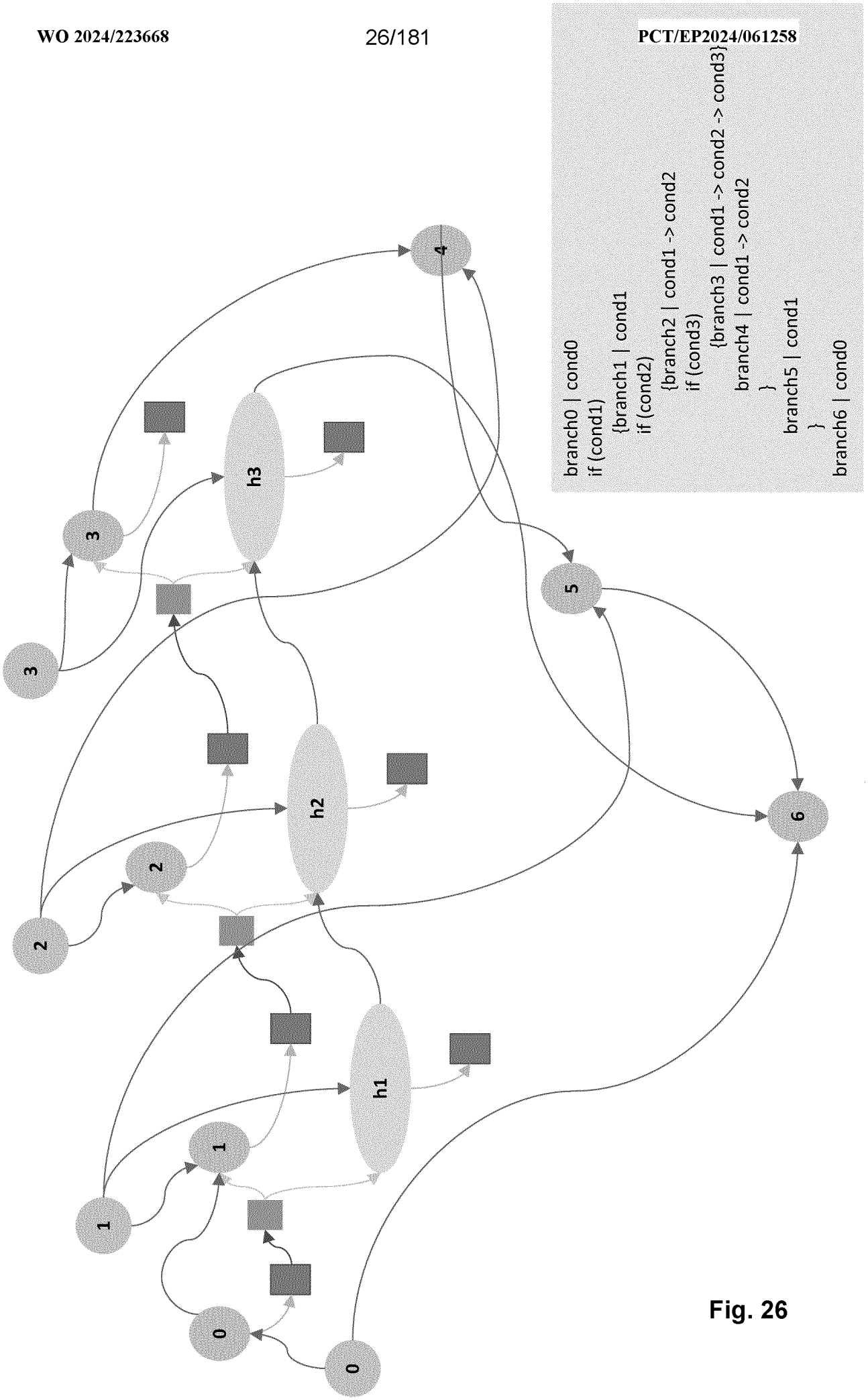

branch0 | cond0
if (cond1)
{branch1 | cond1
if (cond2)
{branch2 | cond1 -> cond2
if (cond3)
{branch3 | cond1 -> cond2 -> cond3}
branch4 | cond1 -> cond2
}
branch5 | cond1
}
branch6 | cond0
h1
h2
h3
0
1
2
3
4
5
6

Fig. 26

branch0 | cond0
for (cond1; increment variable, increment size)
{branch1 | cond1}
branch2 | cond0
334/3341
branch2
compare for condition
increment a variable
335
cond0
cond0 -> cond1
CB5
333
CB4
type: loopbranch-transmission
CB3
334/3343
branch1
332
increment size
increment variable
334
CB1
branch0
add transmission package

Fig. 27

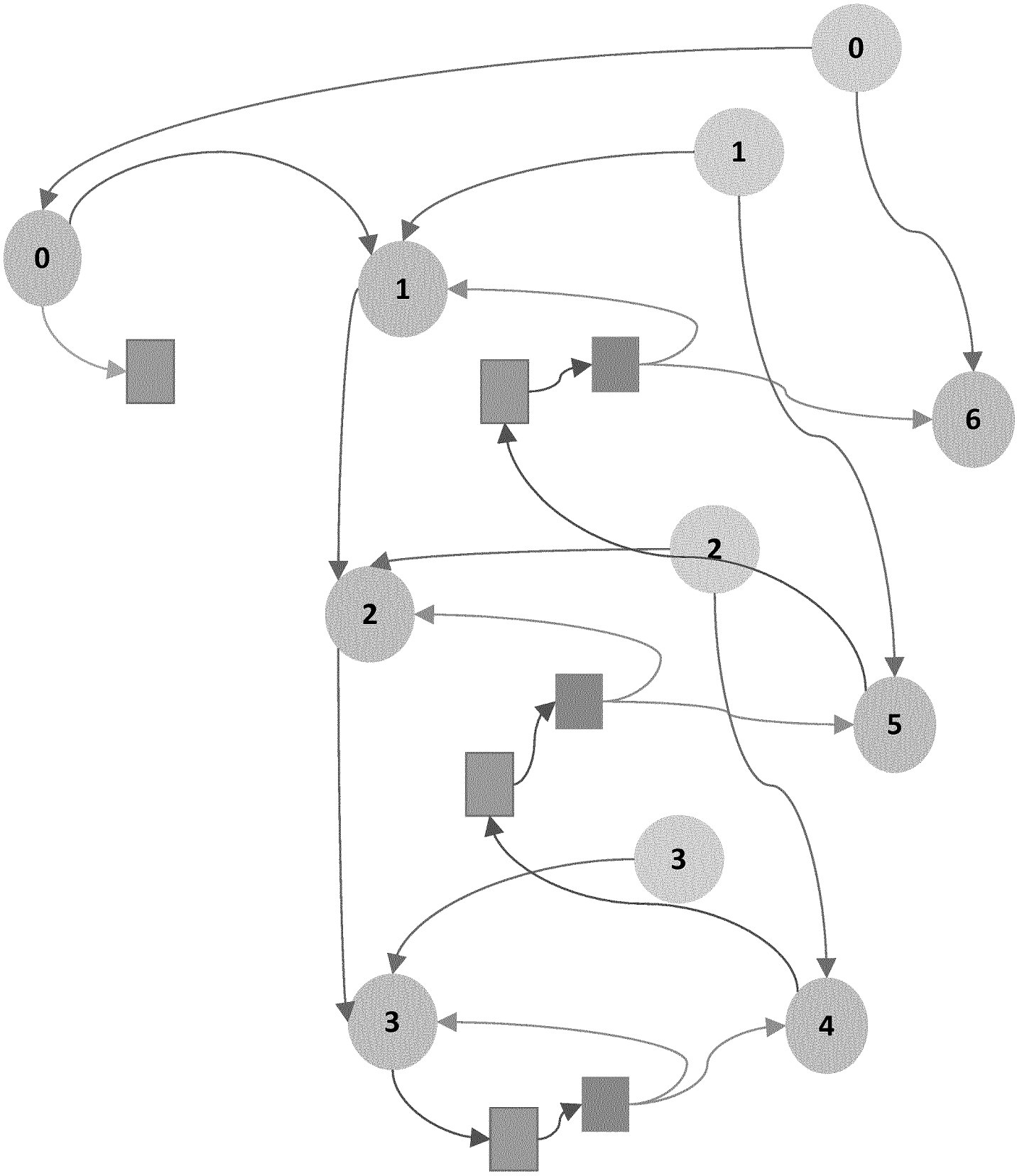


Fig. 28

```
branch0 | cond0
if (cond1)
      {branch1 | cond1
      if (cond2)
            {branch2 | cond1 -> cond2}
      branch3 | cond1
      }
branch4 | cond0
```

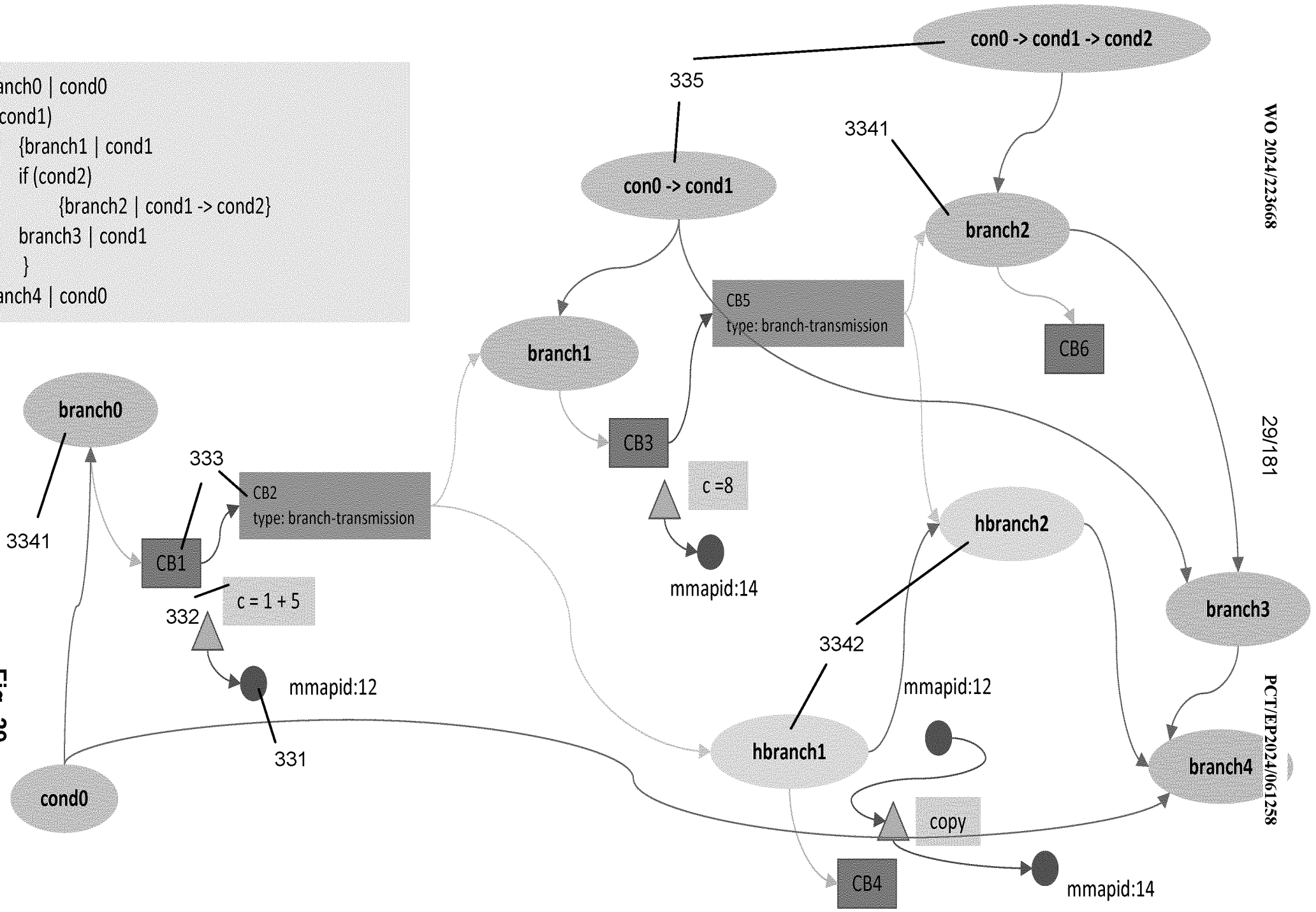


Fig. 29

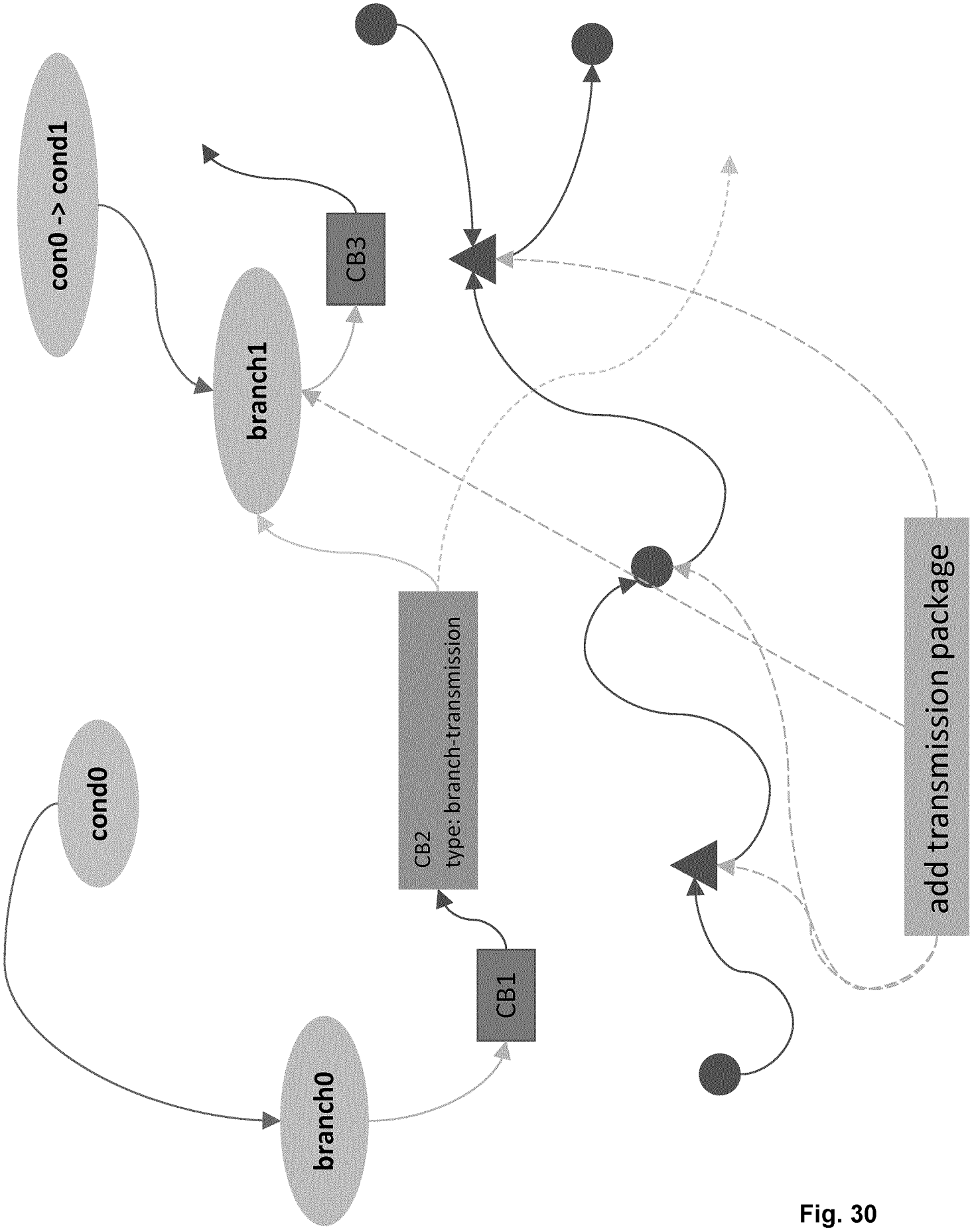

con0 -> cond1
CB3
branch1
CB2
type: branch-transmission
add transmission package
cond0
CB1
branch0

Fig. 30

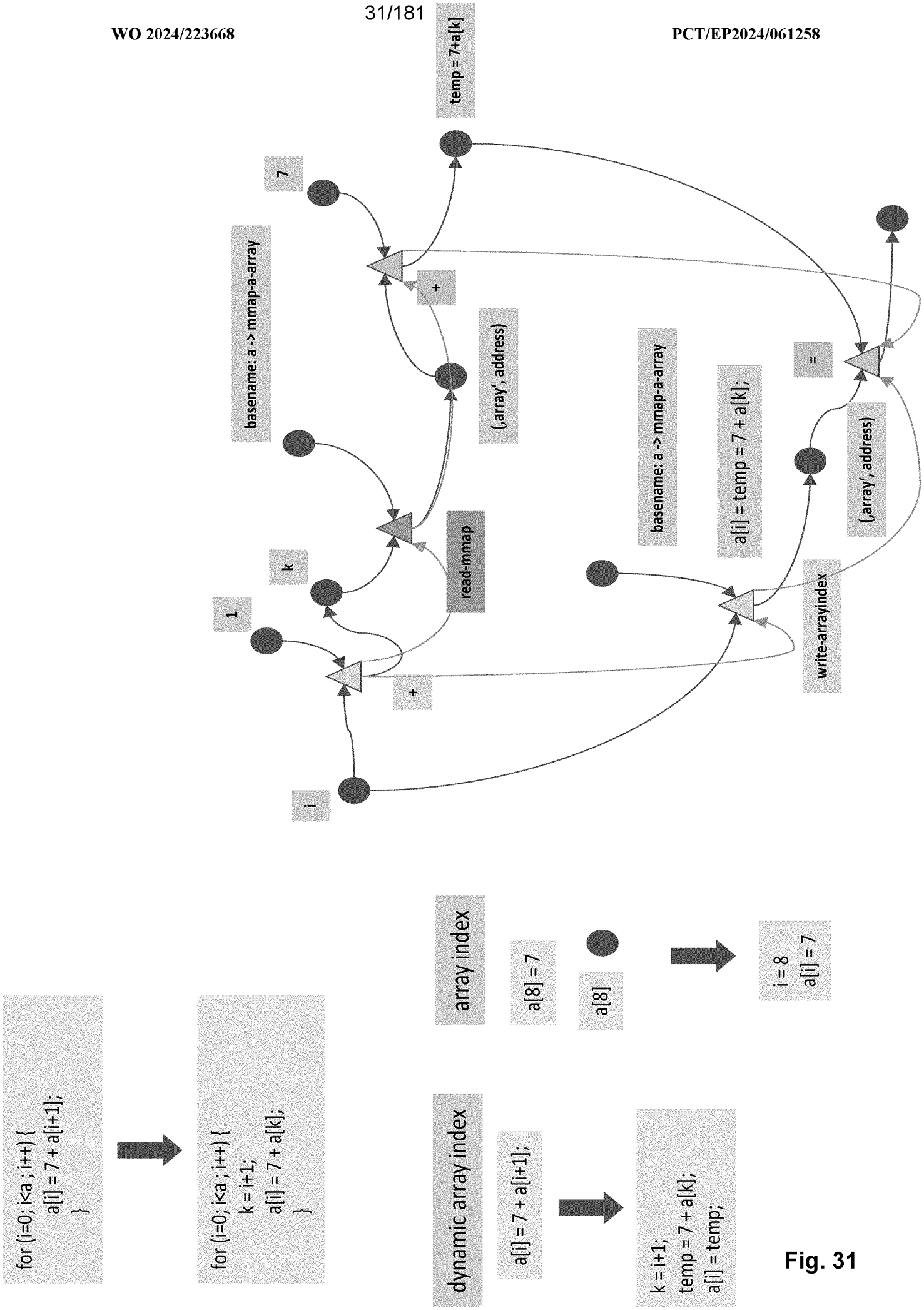
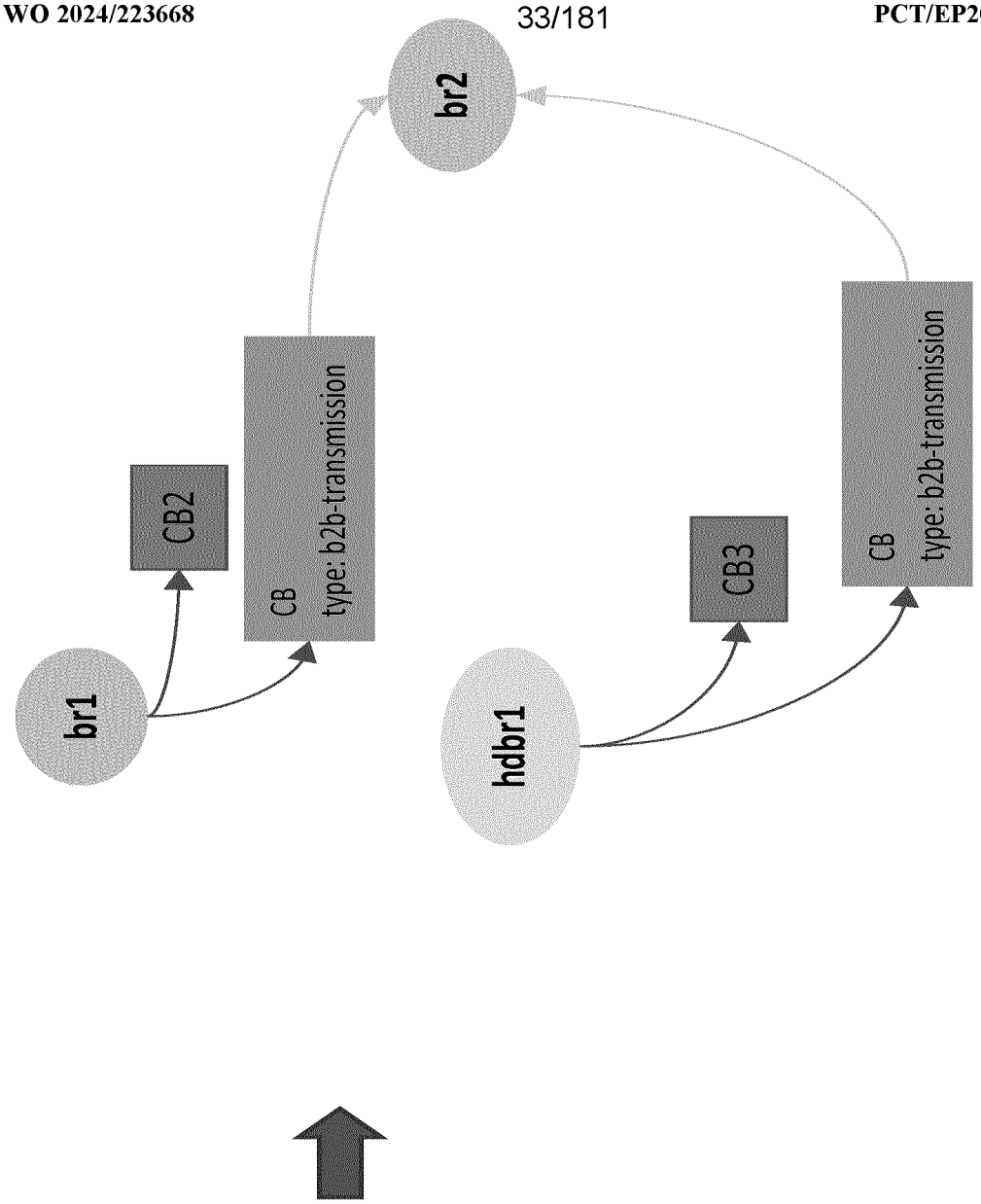

for (i=0; i<a ; i++) {
a[i] = 7 + a[i+1];
}
for (i=0; i<a ; i++) {
k = i+1;
a[i] = 7 + a[k];
}
dynamic array index
a[i] = 7 + a[i+1];
k = i+1;
temp = 7 + a[k];
a[i] = temp;
array index
a[8] = 7
a[8]
i = 8
a[i] = 7
temp = 7+a[k]
7
basename: a -> mmap-a-array
+
(‚array', address)
k
read-mmap
1
+
i
basename: a -> mmap-a-array
a[i] = temp = 7 + a[k];
=
(‚array', address)
write-arrayindex

Fig. 31

computation block node
CB1
Next edge computation block nodes
CB3
CB2
type: fusion-transmission
CB1
CB2
fusion
add transmission package
CB3
CB2
CB2
type: split-transmission
CB1
add transmission package
split

Fig. 32


br2
CB2
CB
type: b2b-transmission
br1
CB3
CB
type: b2b-transmission
hdbr1

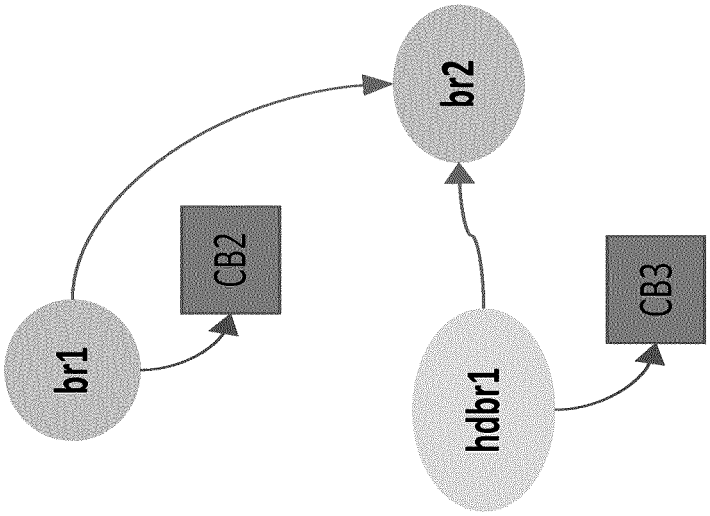
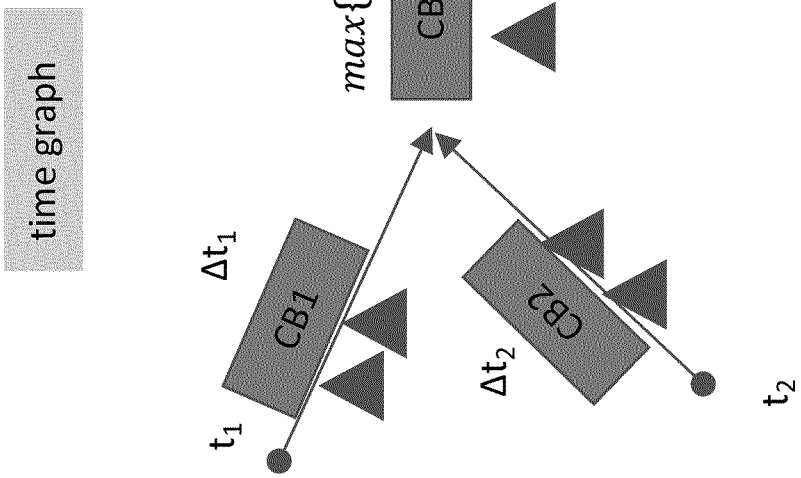

br2
CB2
br1
CB3
hdbr1

Fig. 33


time graph
$max\{t_1 + \Delta t_1, t_2 + \Delta t_2\}$
CB3
Δt_1
CB1
t_1
Δt_2
CB2
t_2

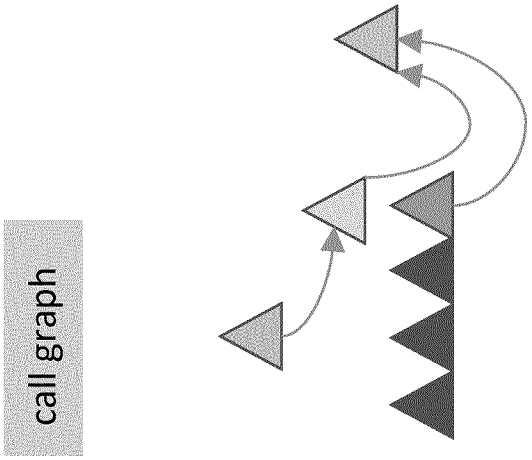

call graph

Fig. 34

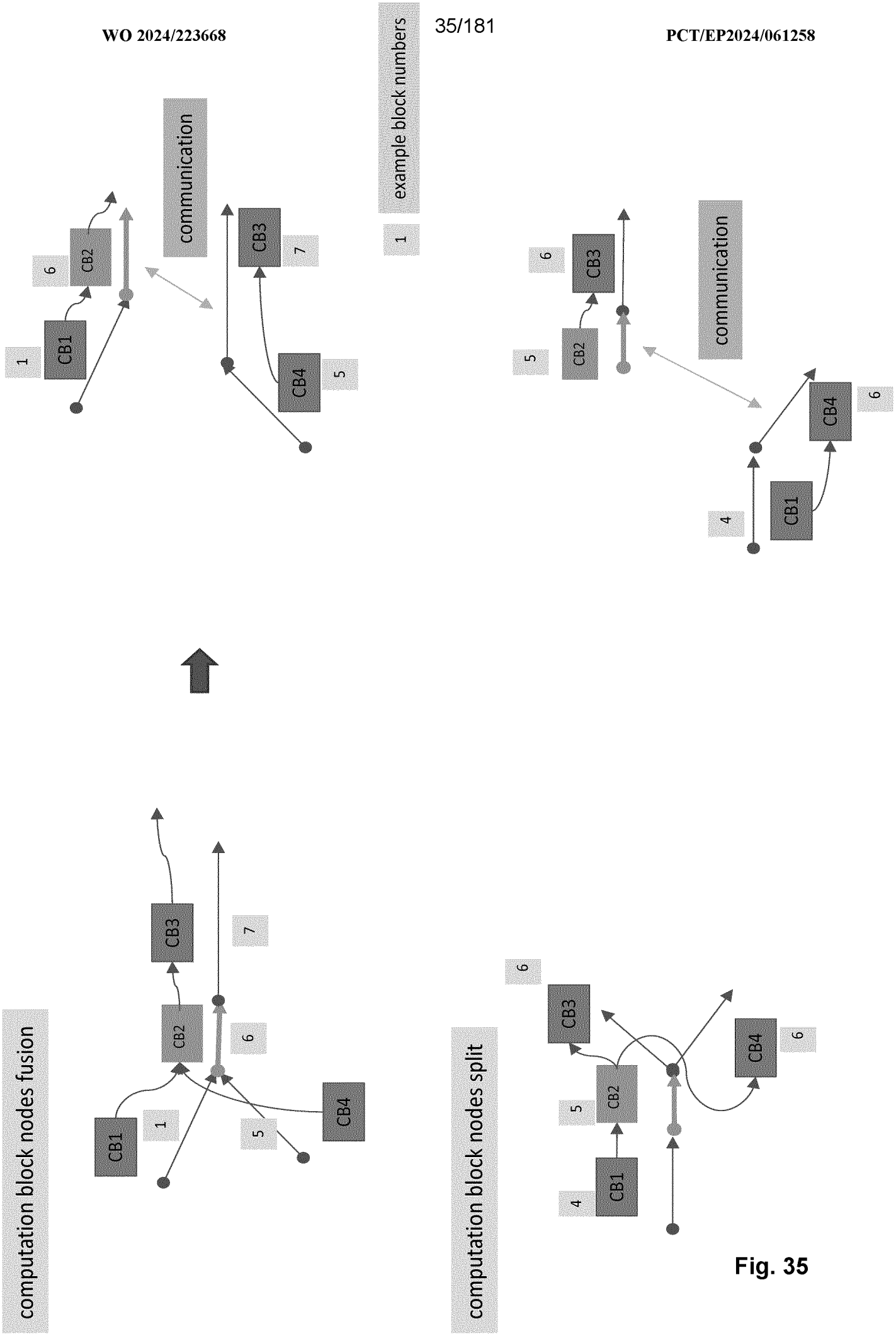

example block numbers
1
communication
CB1
CB2
CB3
CB4
1
6
7
5
communication
CB1
CB2
CB3
CB4
4
5
6
6
computation block nodes fusion
CB1
CB2
CB3
CB4
1
6
7
5
computation block nodes split
CB1
CB2
CB3
CB4
4
5
6
6

Fig. 35

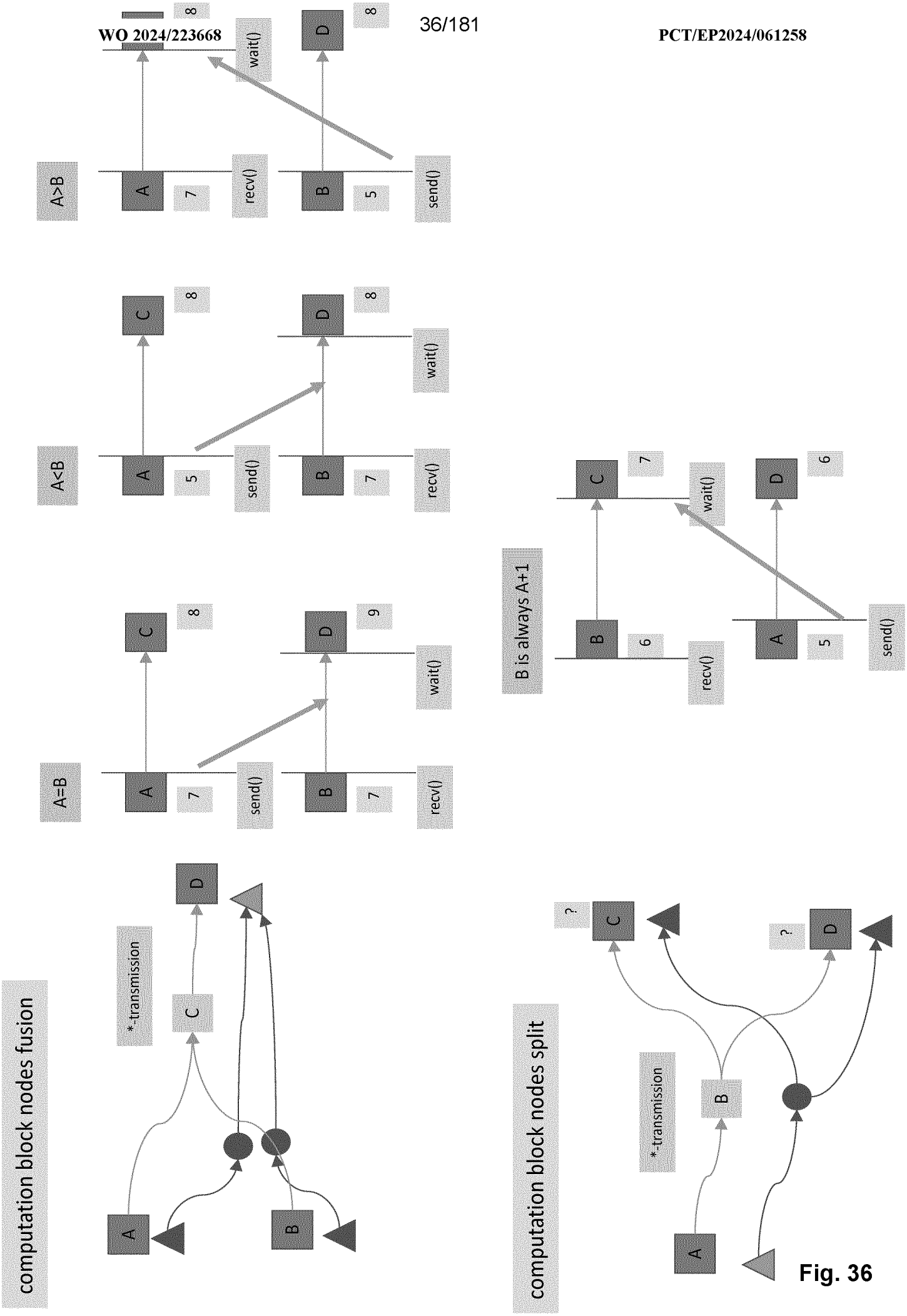
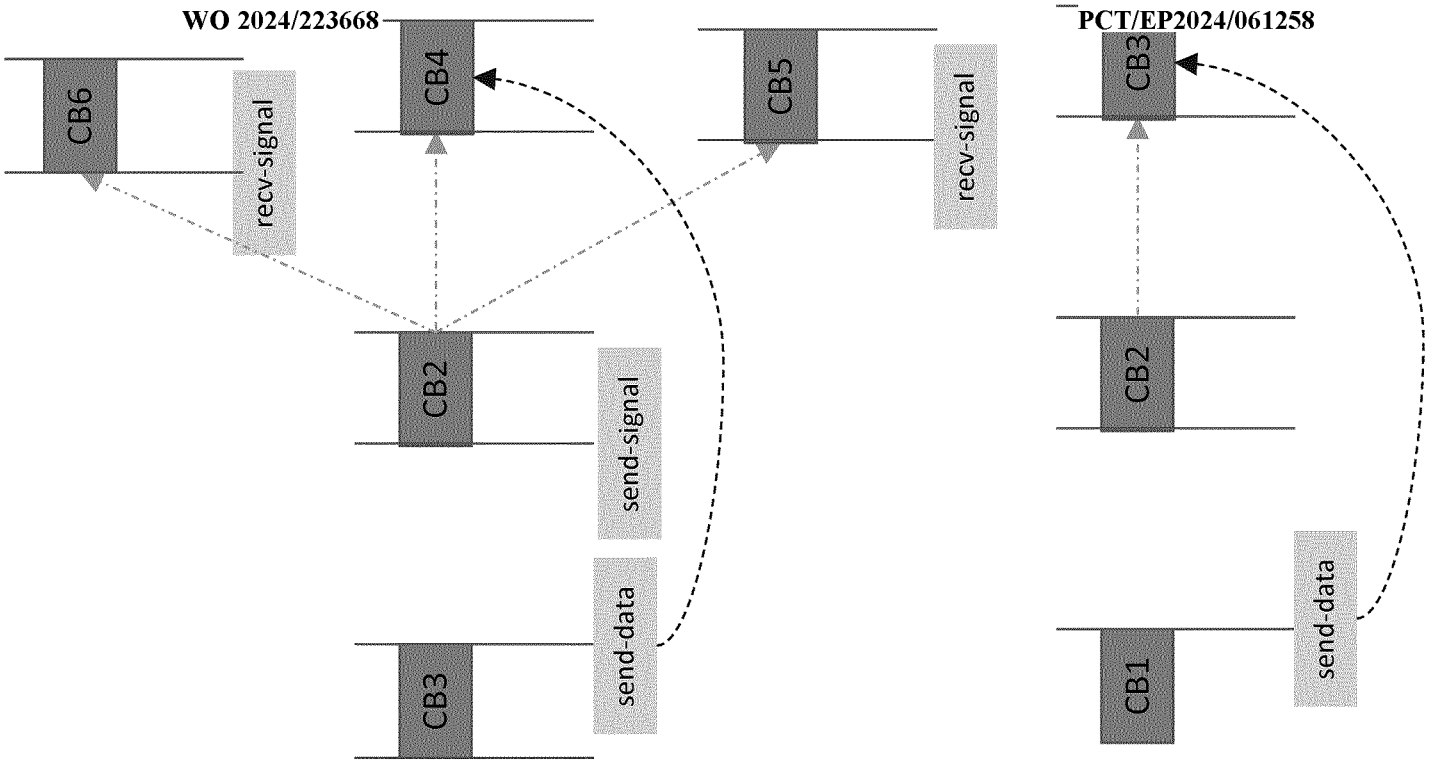

A>B
A
7
recv()
B
5
send()
wait()
8
D
8
A<B
A
5
send()
B
7
recv()
C
8
D
8
wait()
A=B
A
7
send()
B
7
recv()
C
8
D
9
wait()
B is always A+1
B
6
recv()
C
7
wait()
A
5
send()
D
6
computation block nodes fusion
A
B
C
*-transmission
D
computation block nodes split
A
B
*-transmission
C
?
D
?

Fig. 36


CB6
recv-signal
CB4
CB5
recv-signal
CB2
send-signal
send-data
CB3
CB3
CB2
CB1
send-data

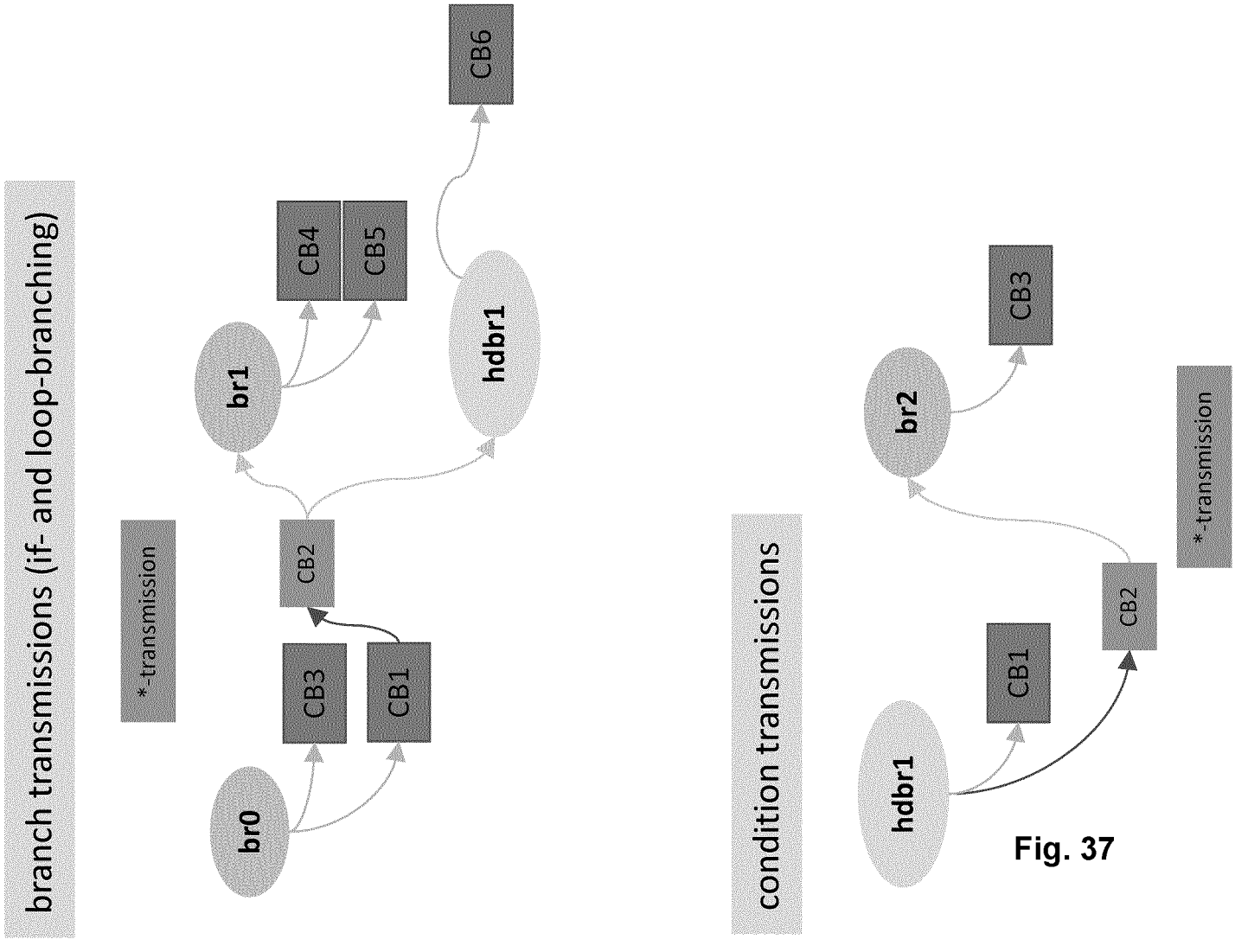

branch transmissions (if- and loop-branching)
*-transmission
br0
CB3
CB1
CB2
br1
CB4
CB5
hdbr1
CB6
condition transmissions
hdbr1
CB1
CB2
br2
CB3
*-transmission

Fig. 37

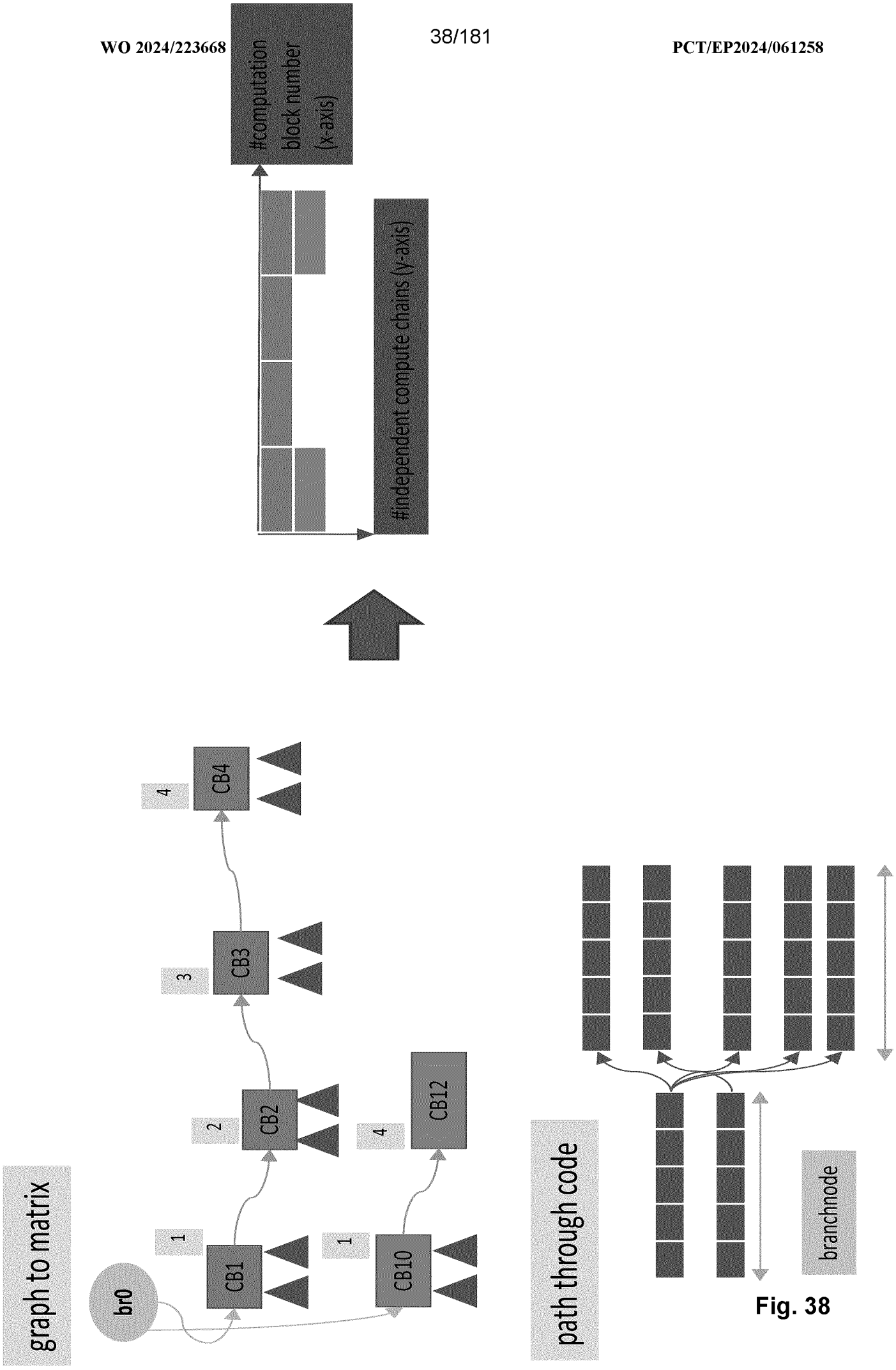

#computation block number (x-axis)
#independent compute chains (y-axis)
graph to matrix
br0
CB1
CB2
CB3
CB4
CB10
CB12
1
2
3
4
1
4
path through code
branchnode

Fig. 38

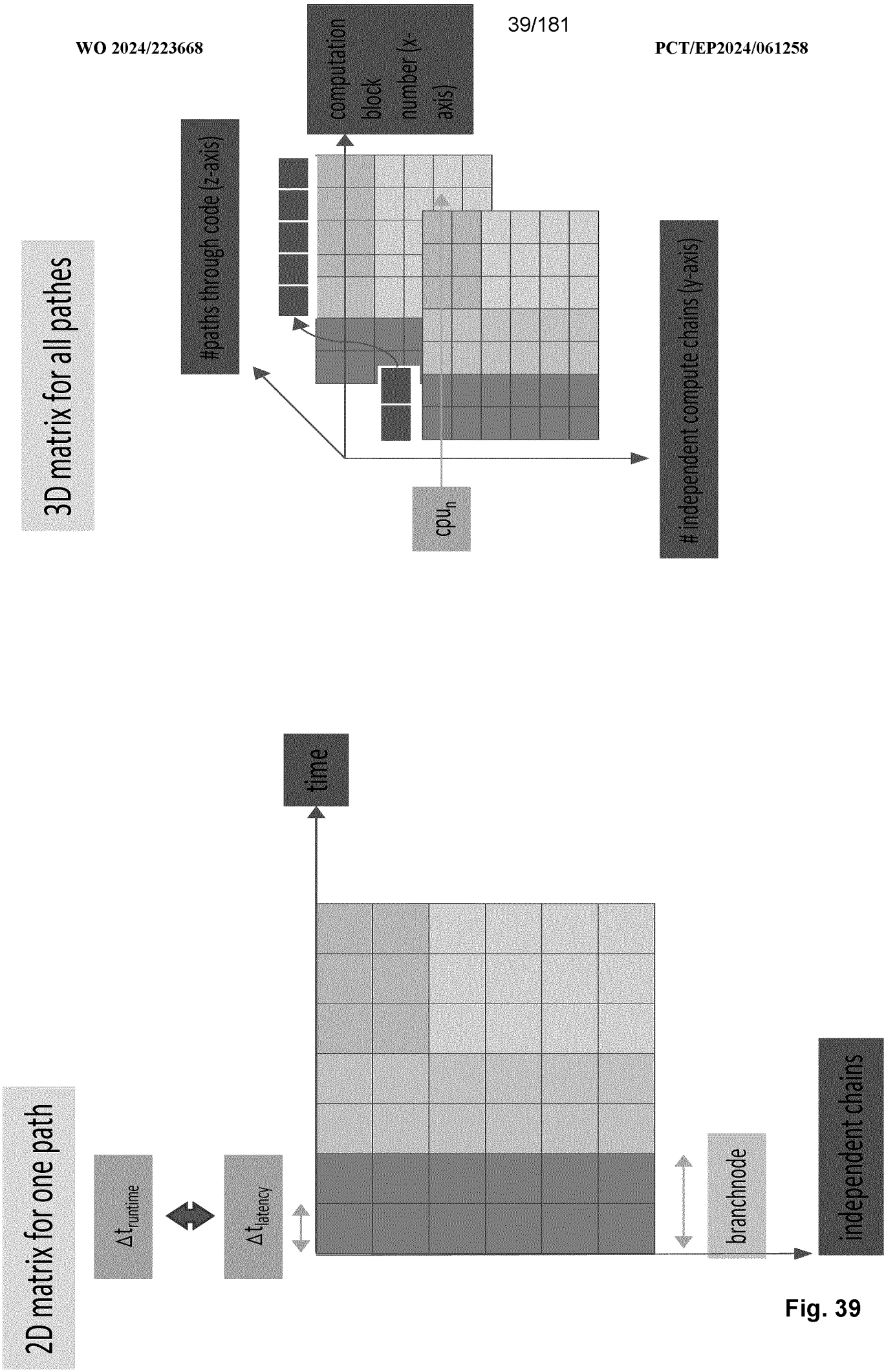

3D matrix for all pathes
#paths through code (z-axis)
computation block number (x-axis)
independent compute chains (y-axis)
cpu_n
2D matrix for one path
Δt_runtime
Δt_latency
time
branchnode
independent chains

Fig. 39

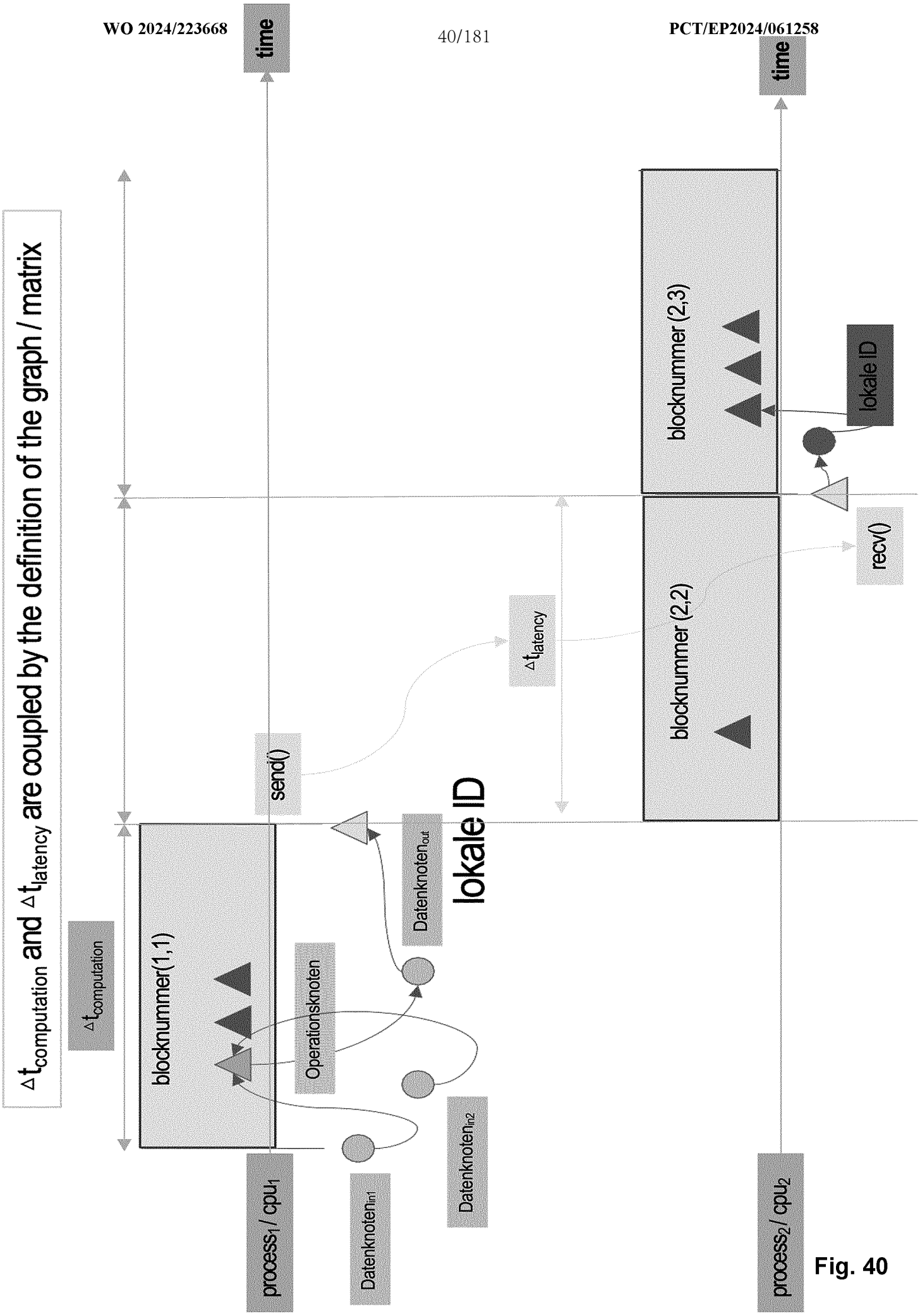

$\Delta t_{computation}$ and $\Delta t_{latency}$ are coupled by the definition of the graph / matrix
time
time
$\Delta t_{computation}$
blocknummer(1,1)
Operationsknoten
Datenknoten$_{out}$
send()
$\Delta t_{latency}$
blocknummer (2,2)
blocknummer (2,3)
recv()
lokale ID
lokale ID
Datenknoten$_{in1}$
Datenknoten$_{in2}$
process$_1$ / cpu$_1$
process$_2$ / cpu$_2$

Fig. 40

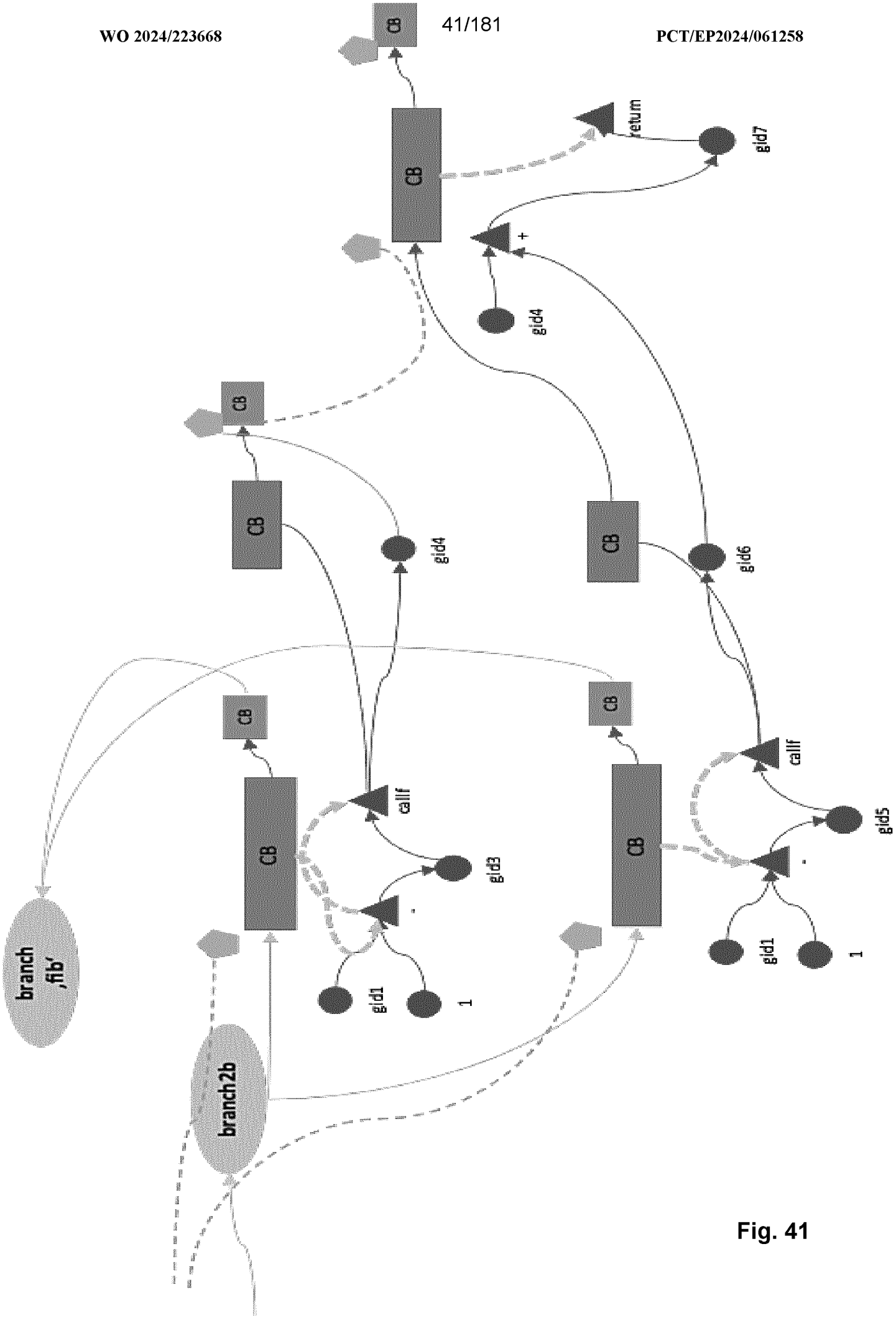

CB
CB
return
gId7
+
gId4
CB
CB
gId4
CB
gId6
CB
CB
callf
callf
CB
CB
gId3
gId5
.
.
gId1
1
gId1
1
branch
‚fib'
branch2b

Fig. 41

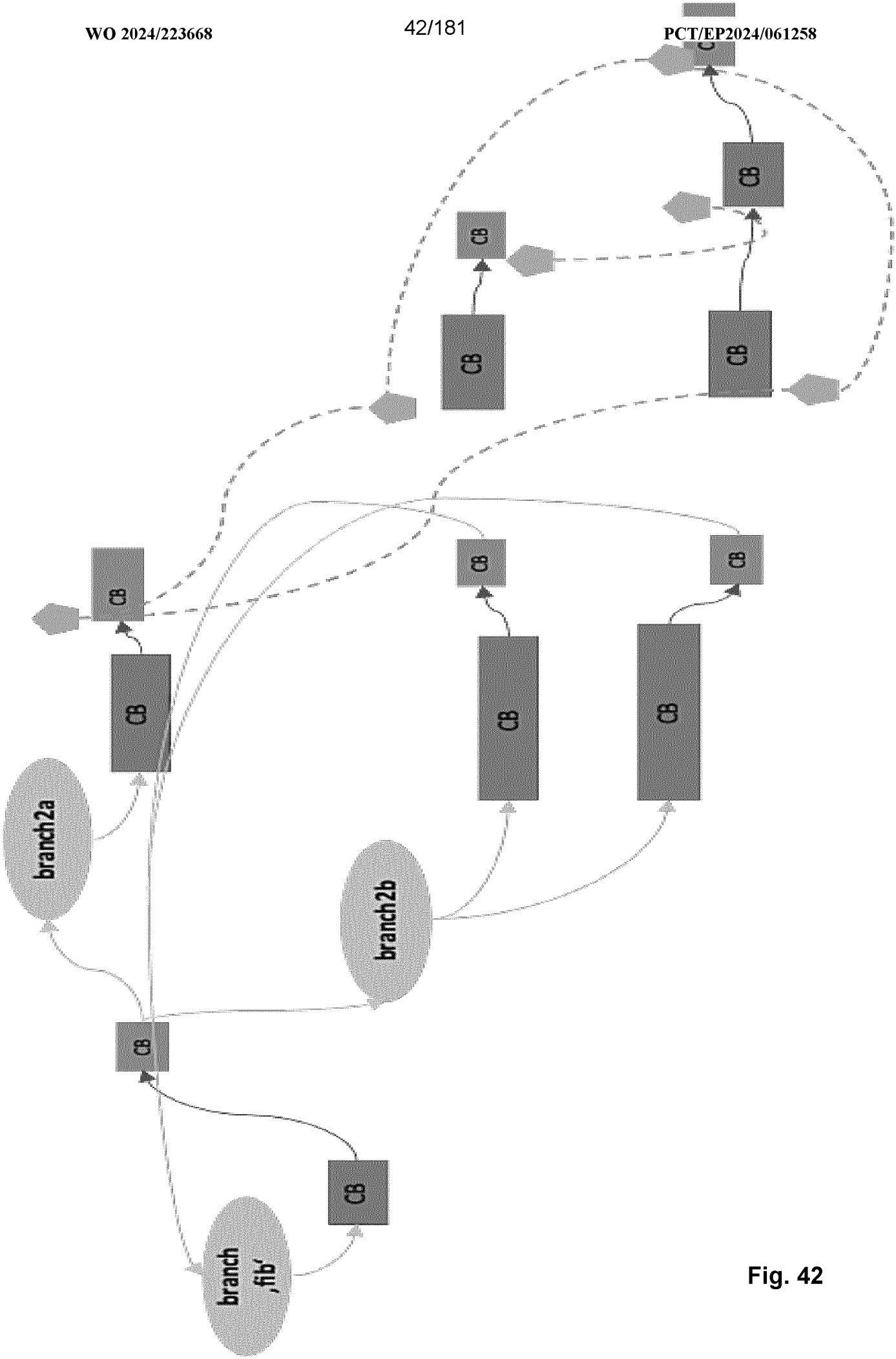


Fig. 42

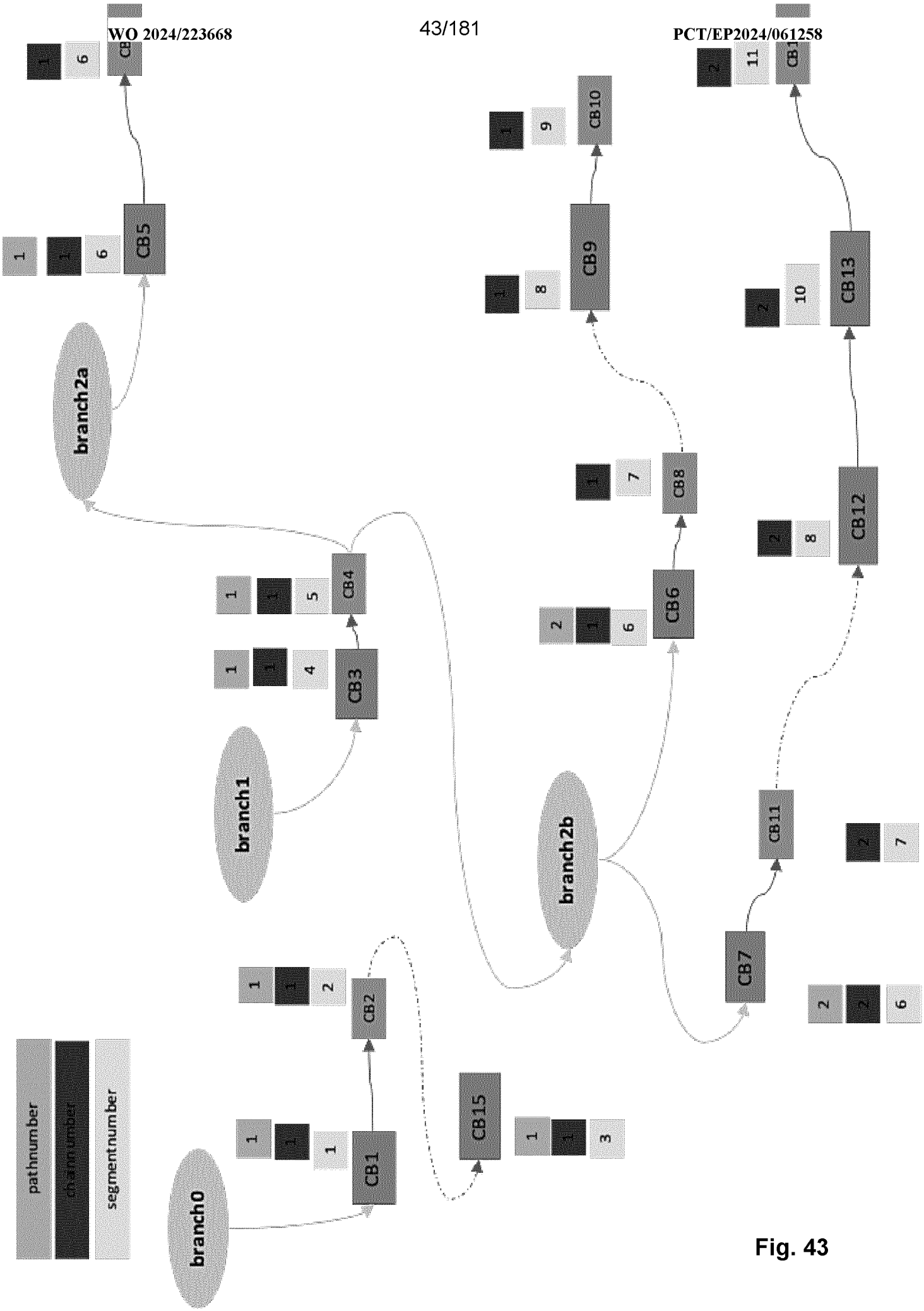

pathnumber
chainnumber
segmentnumber
branch0
branch1
branch2a
branch2b
CB1
CB2
CB3
CB4
CB5
CB6
CB7
CB8
CB9
CB10
CB11
CB12
CB13
CB15

Fig. 43

cbid	segm,chain,path	x	y	z
CB1	1,1,1	1	1	1,2
CB2	2,1,1	2	1	1,2
CB15	3,1,1	3	1	1,2
CB3	4,1,1	4	1	1,2
CB4	5,1,1	5	1	1
CB6	6,1,2	6	1	2
CB8	7,1,2	7	1	2
CB9	8,1,2	8	1	2
CB10	9,1,2	9	1	2
CB7	6,2,2	6	2	2
CB11	7,2,2,	7	2	2
CB12	8,2,2	8	2	2
CB13	10,2,2	10	2	2
CB14	11,2,2	11	2	2

Fig. 44

Transfer-Matrix

2	1	2	3	4	5	6	7	8	9	10	11
1	e: COM1->		s: ->COM10	e: SIG1-> COM2-> s: ->COM1 ->COM4 ->COM5		s: ->SIG1 ->COM2 e: COM4->		s: ->COM8	s: COM6->		
2						s: ->SIG1 ->COM2 e: COM5->		s: ->COM9		s: ->COM6	s: COM7->

1	1	2	3	4	5	6	7	8	9	10	11
1	e: COM1->		s: ->COM10	e: SIG1-> COM2-> s: ->COM1 ->COM4 ->COM5		s: ->SIG1 ->COM2 e: COM3->					
2											

Fig. 45

	source cbid		target cbid			direction
	start	end	start	end	variable	
COM1		CB1 - (1,1,1)	CB3 - (4,1,1)		gid8, addr CB15	forward
SIG1		CB3 - (4,1,1)	CB5 - (6,1,1), CB6 - (6,1,2), CB7 - (6,2,2)		gid2	forward
COM2		CB3 - (4,1,1)	CB5 - (6,1,1), CB6 - (6,1,2), CB7 - (6,2,2)		gid1	forward
COM3		CB5 - (6,1,1)	CB15 / CB9 / CB12		gid1	
COM4		CB6 - (6,1,2)	CB3 - (4,1,1)		gid3, addr CB9	backward
COM5		CB7 - (6,2,2)	CB3 - (4,1,1)		gid5, addr CB12	backward
COM6		CB9 - (9,1,2)	CB10 - (10,2,2)		gid4	forward
COM7		CB13 - (10,2,2)	CB15 / CB9 / CB12		gid7	
COM8		LL	CB9 - (8,1,2)		gid4	
COM9		LL	CB12 - (8,2,2)		gid6	
COM10		LL	CB15 - (3,1,1)		gid9	

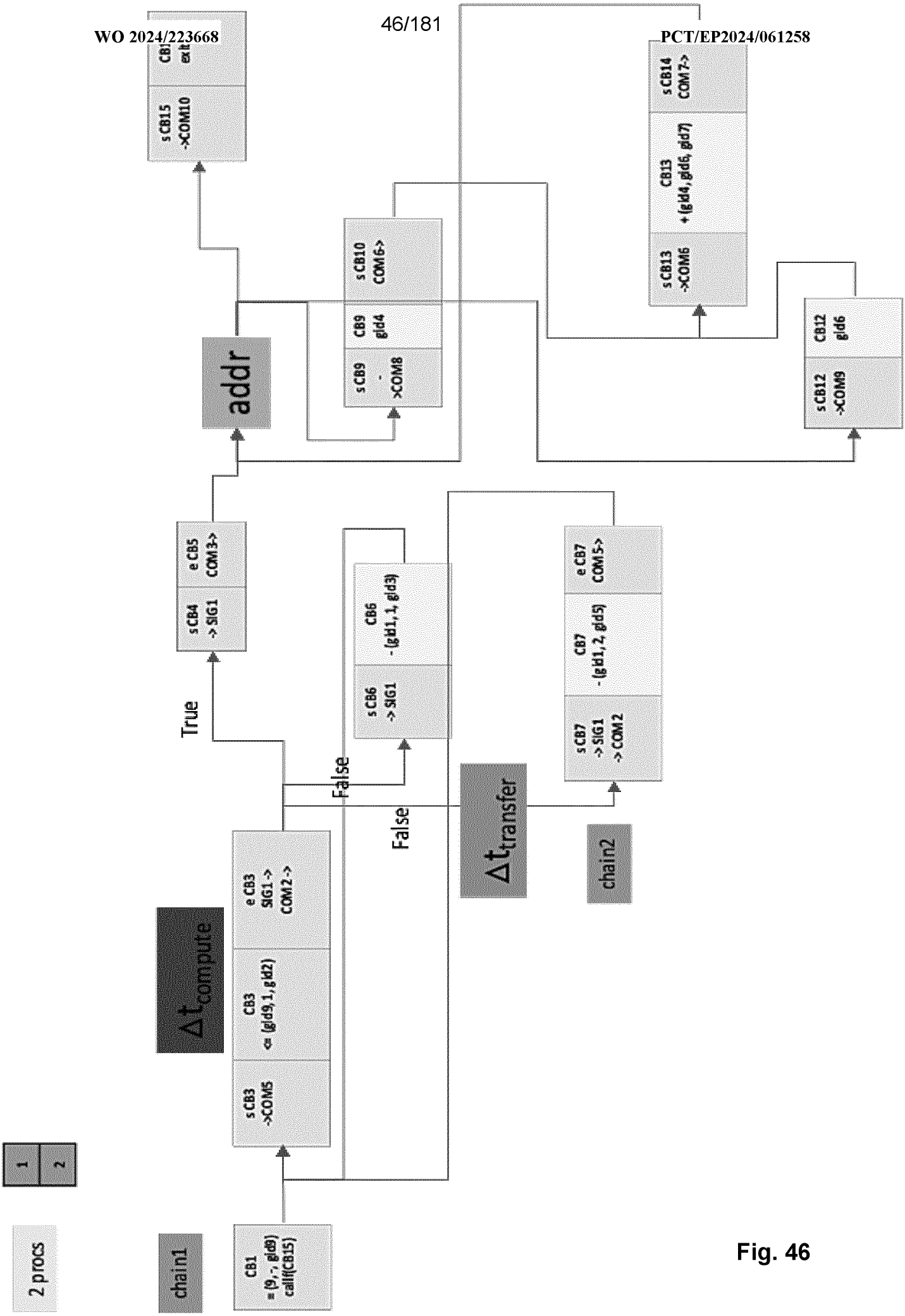

2 procs
1
2
chain1
CB1
= (9, -, gId9)
call(CB15)
chain2
Δt compute
Δt transfer
addr
CB3
SIG1 ->
COM2 ->
CB3
<= (gId9, 1, gId2)
sCB3
->COM5
CB6
- (gId1, 1, gId3)
sCB6
-> SIG1
CB7
- (gId1, 2, gId5)
sCB7
-> SIG1
-> COM2
eCB5
COM3->
sCB4
-> SIG1
sCB10
COM6->
CB9
gId4
sCB9
-
>COM8
CB13
+ (gId4, gId6, gId7)
sCB13
->COM6
sCB14
COM7->
CB12
gId6
sCB12
->COM9
sCB15
->COM10
True
False
False

Fig. 46

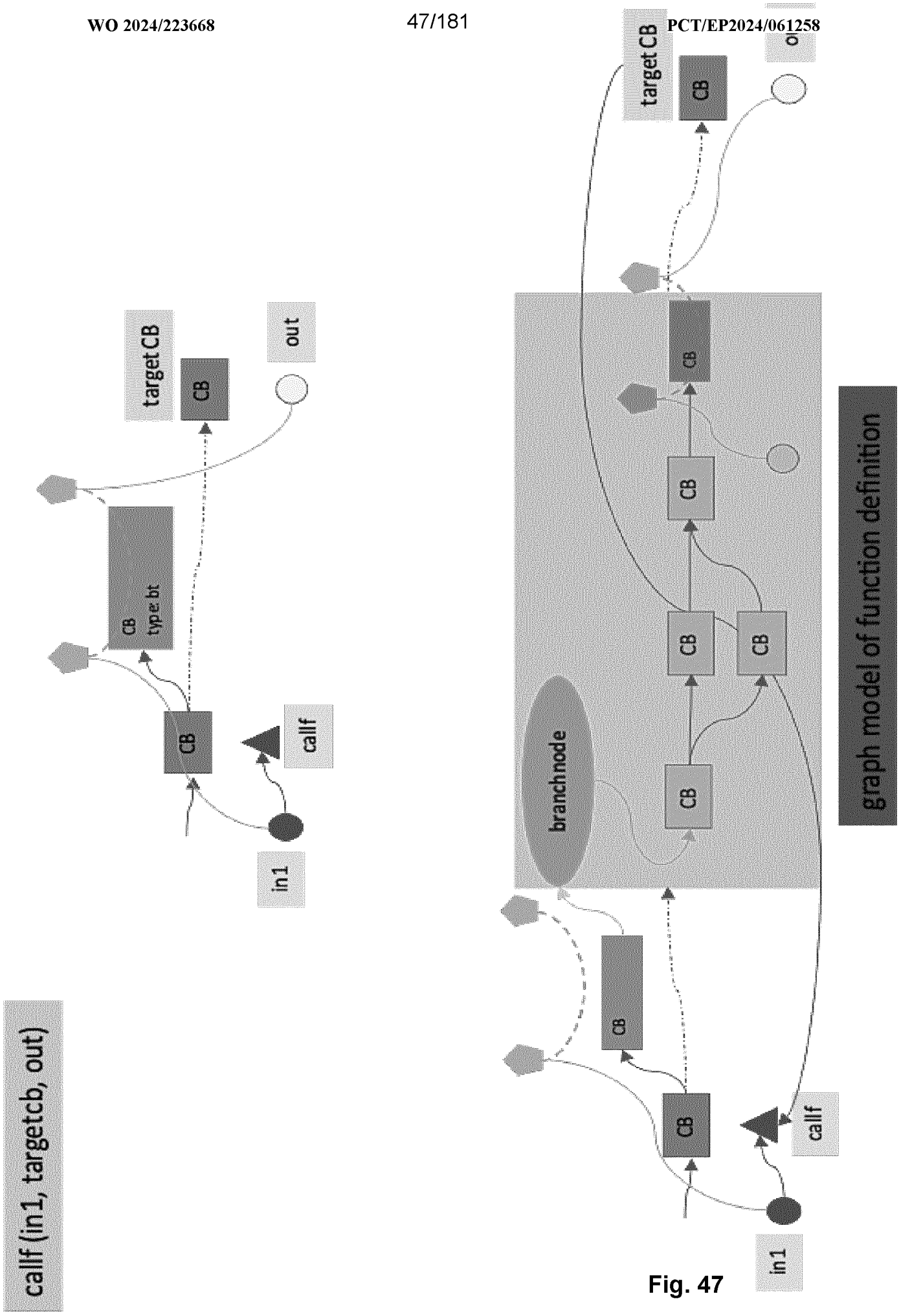

callf (in1, targetcb, out)
target CB
CB
out
CB type bt
CB
callf
in1
target CB
CB
out
CB
CB
CB
CB
branchnode
CB
CB
callf
in1
graph model of function definition

Fig. 47

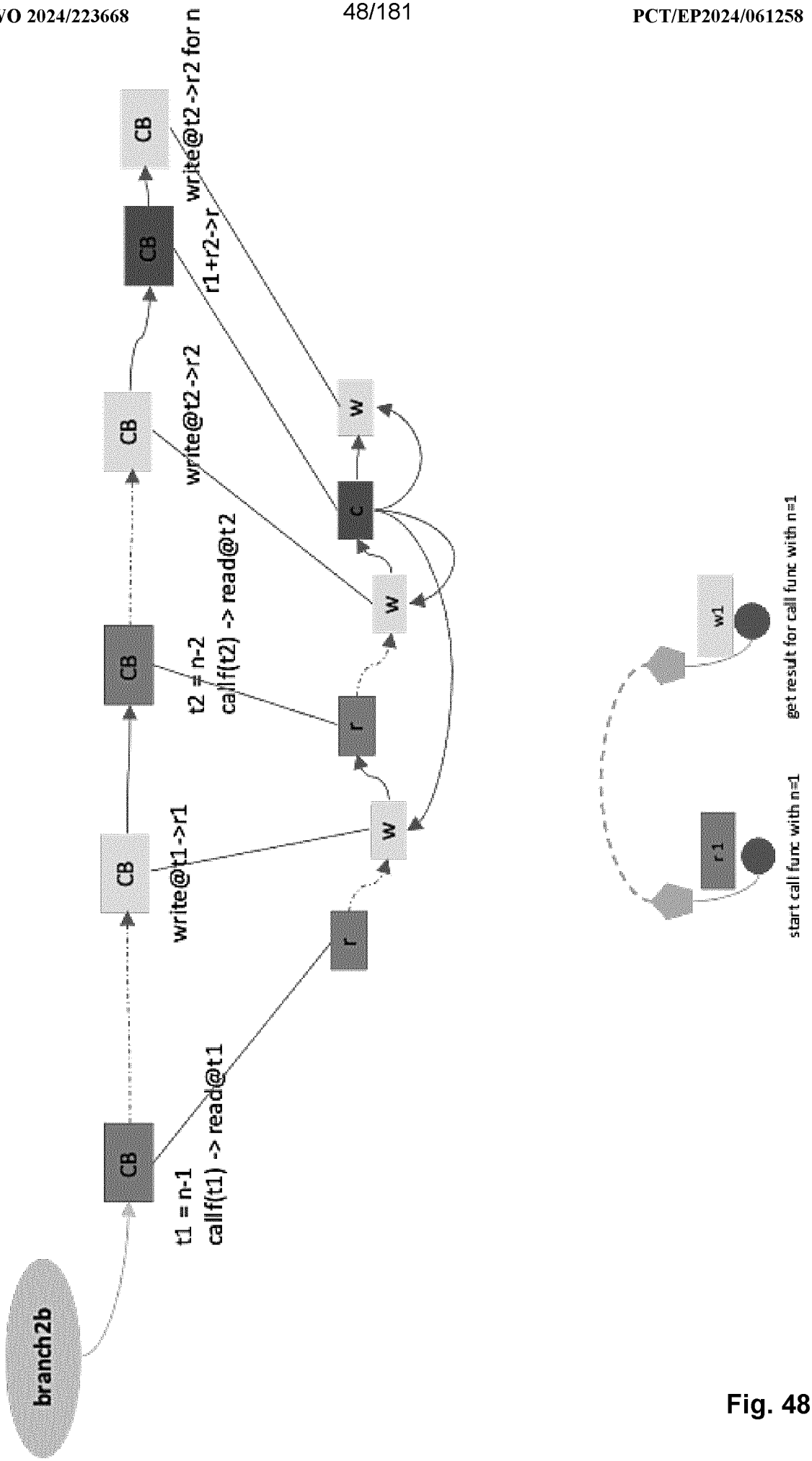

branch2b
CB
CB
CB
CB
CB
CB
t1 = n-1
callf(t1) -> read@t1
write@t1->r1
t2 = n-2
callf(t2) -> read@t2
write@t2->r2
r1+r2->r
write@t2->r2 for n
r
r
w
w
w
w
c
w1
r1
start call func with n=1
get result for call func with n=1

Fig. 48

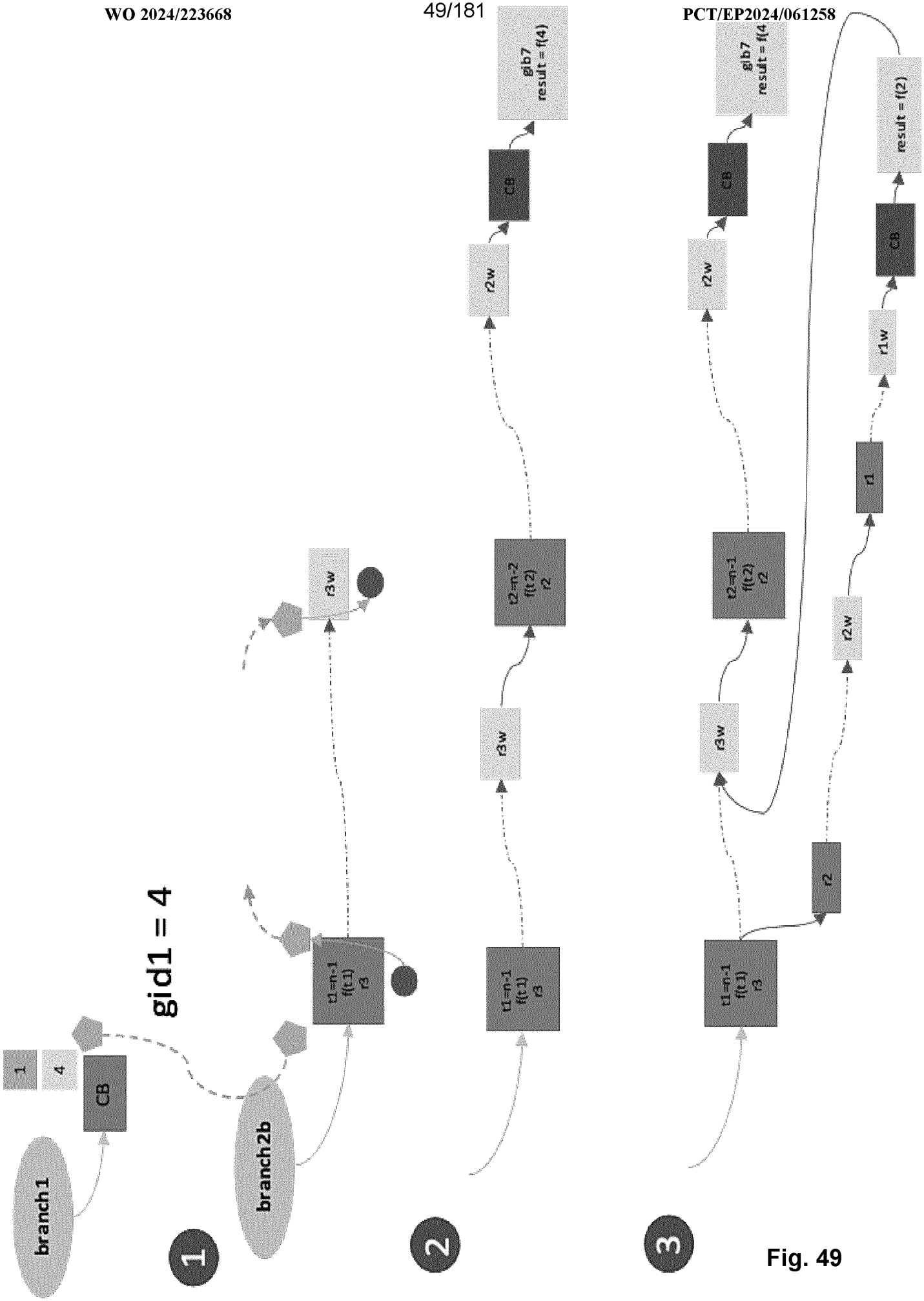

gid1 = 4
branch1
branch2b
1
4
CB
t1=n-1
f(t1)
r3
r3w
t2=n-2
f(t2)
r2
r2w
gib7
result = f(4)
t1=n-1
f(t1)
r3
r3w
t2=n-1
f(t2)
r2
r2w
r2
t1
r1w
CB
result = f(2)
1
2
3

Fig. 49

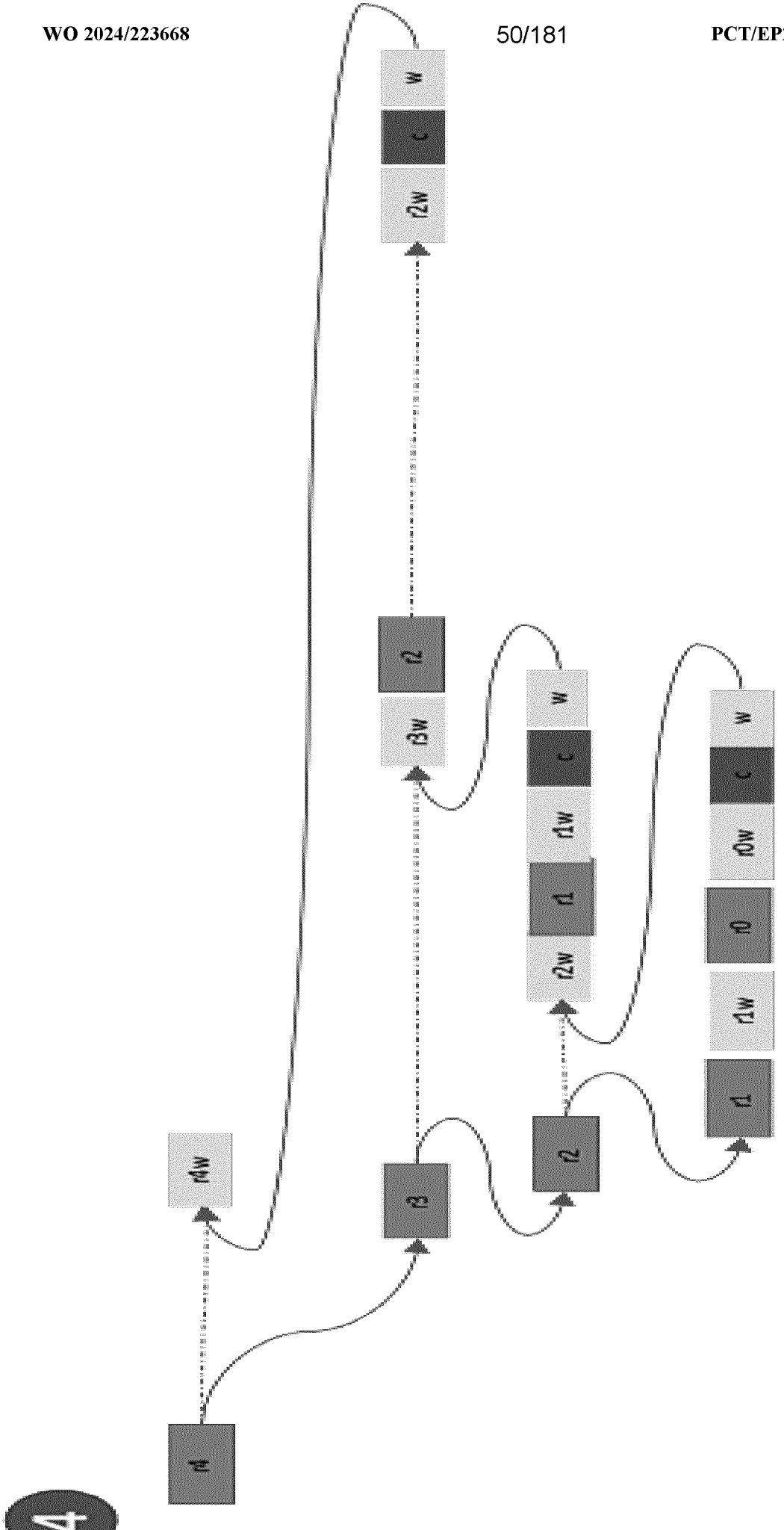


Fig. 50

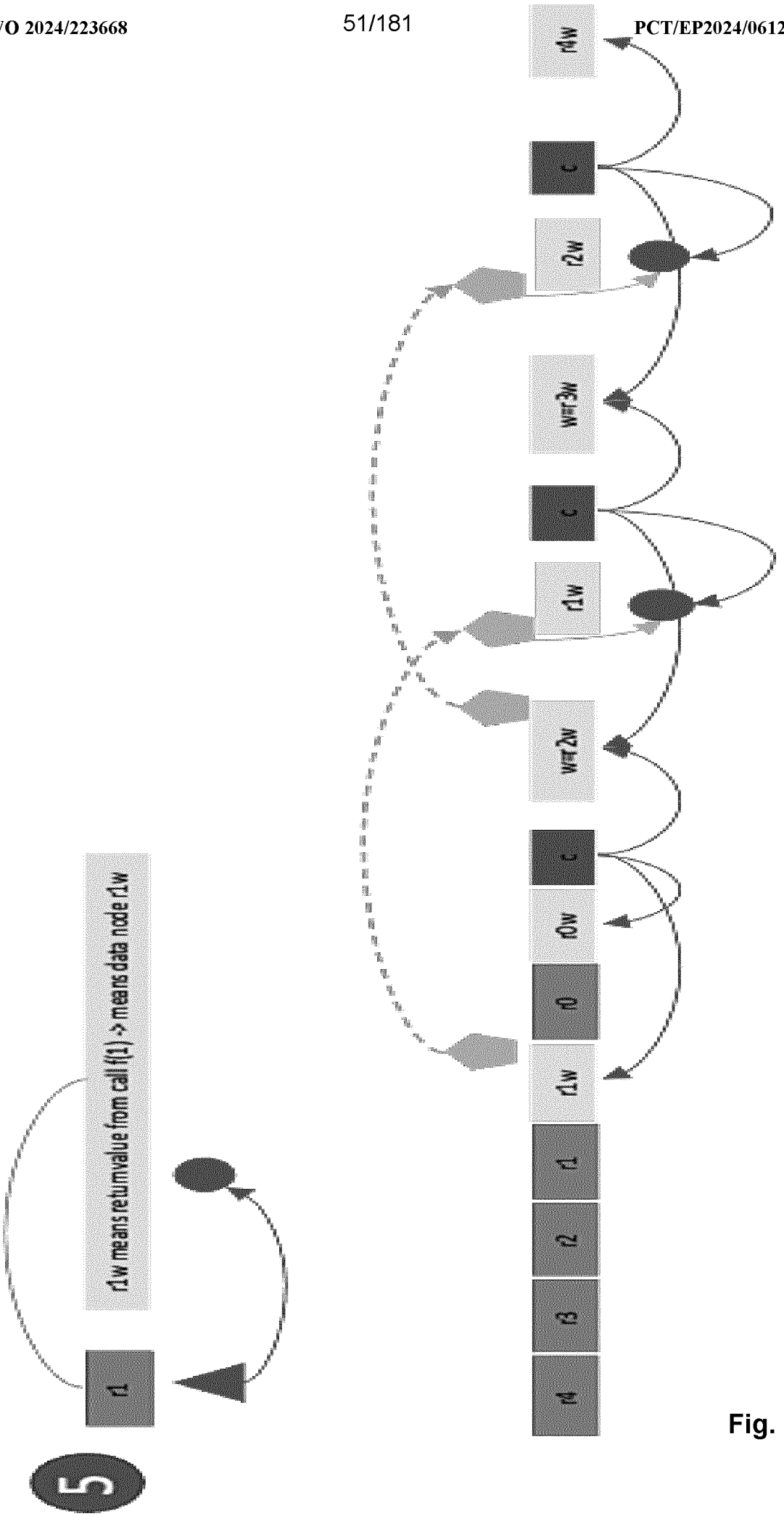

5
r1
r1w means returnvalue from call f(1) -> means data node r1w
r4
r3
r2
r1
r1w
r0
c
r0w
c
w#r2w
r1w
c
w#r3w
r2w
c
r4w

Fig. 51

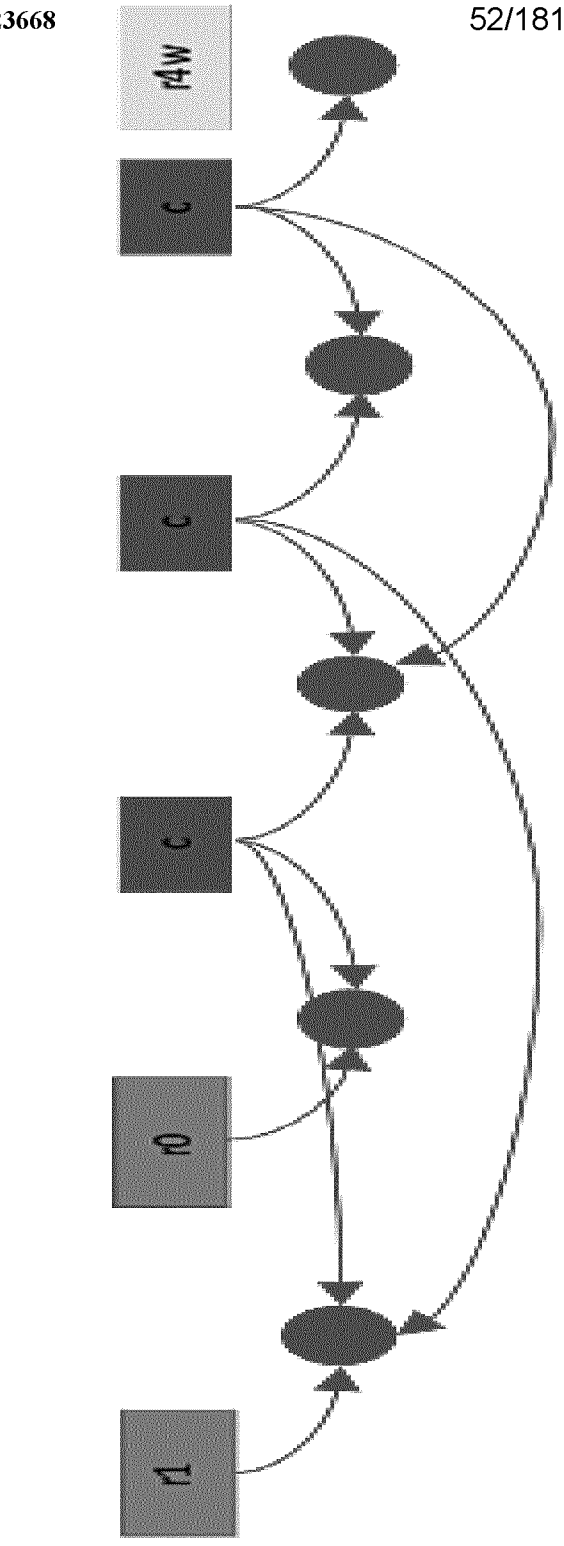


Fig. 52

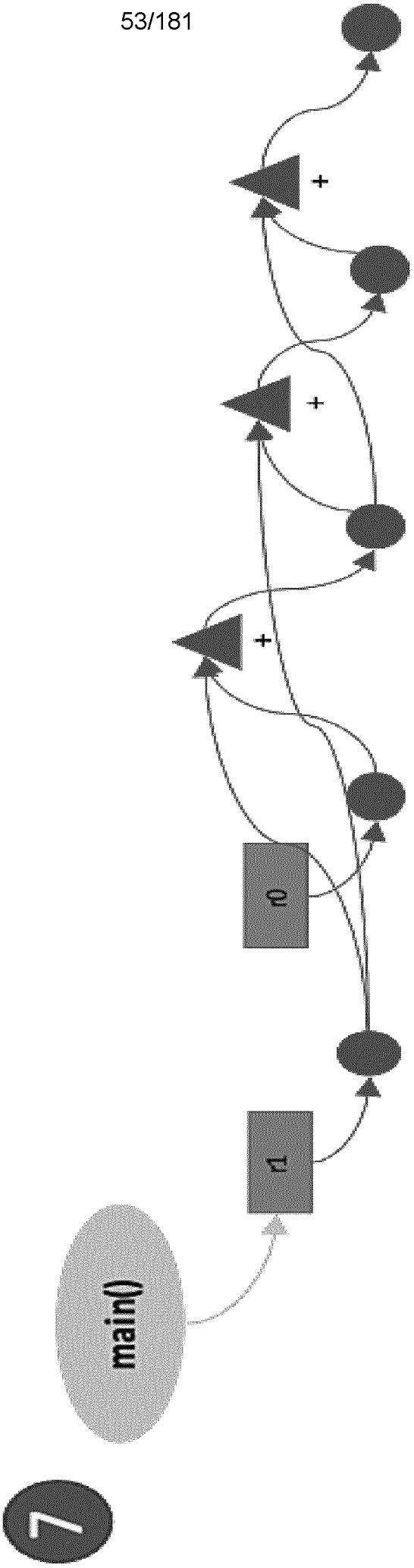


Fig. 53

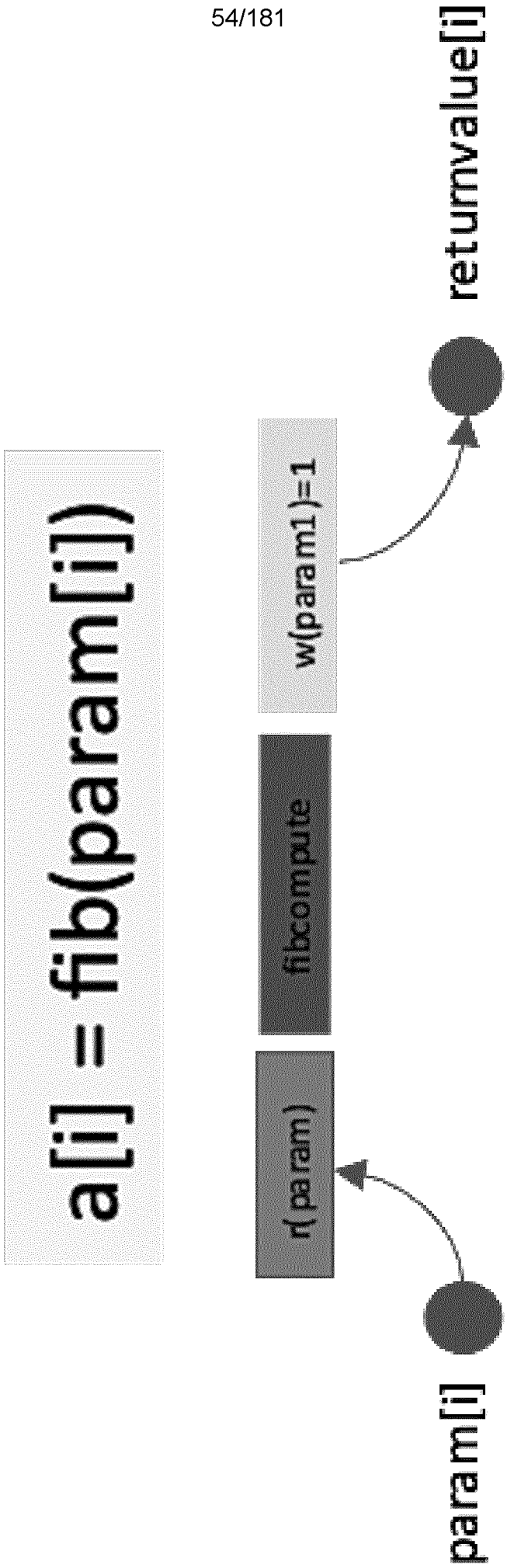

a[i] = fib(param[i])
w(param1)=1
fibcompute
r(param)
param[i]
returnvalue[i]

Fig. 54

initial block

```
# Set the initial condition
u.fill(u_initial)
```

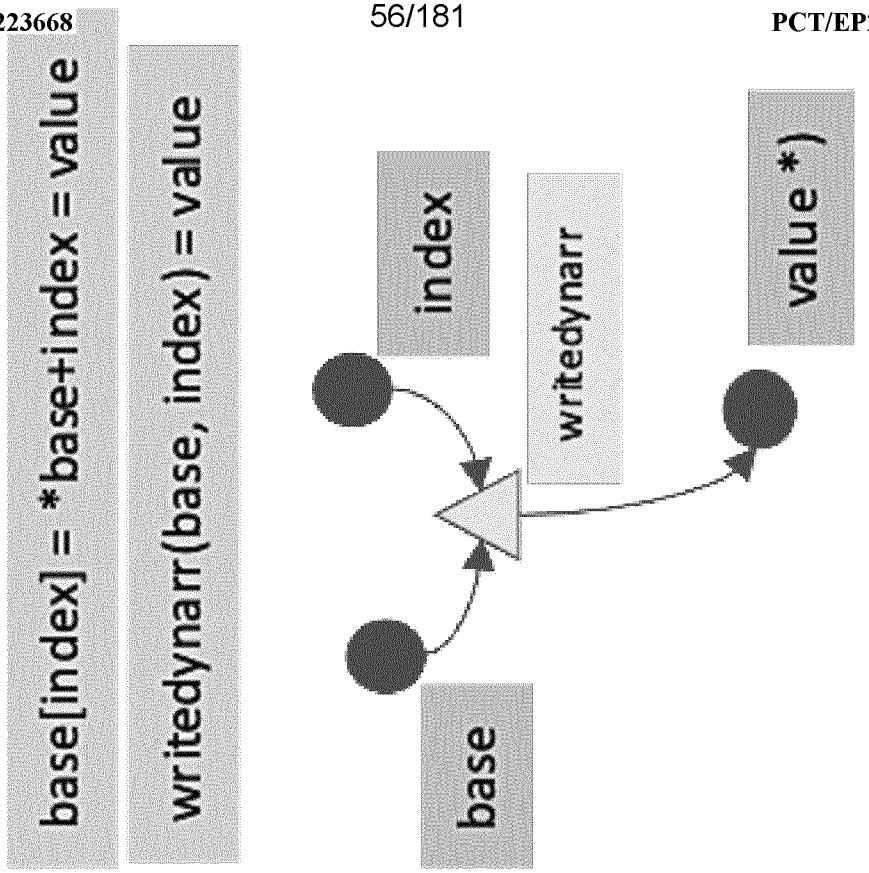
boundary conditions

```
# Set the boundary conditions
u[:, (plate_length-1):, : ] = u_top
u[:, :, :1] = u_left
u[:, :1, 1:] = u_bottom
u[:, :, (plate_length-1):] = u_right
```

computation block

```
for k in range(0, max_iter_time-1, 1):
    for i in range(1, plate_length-1, delta_x):
        for j in range(1, plate_length-1, delta_x):
            u[k + 1, i, j] = gamma * (u[k][i+1][j] + u[k][i-1][j] + u[k][i][j+1] + u[k][i][j-1] - 4*u[k][i][j]) + u[k][i][j]
```

Fig. 55


base[index] = *base+index = value
writedynarr(base, index) = value
index
writedynarr
value *)
base

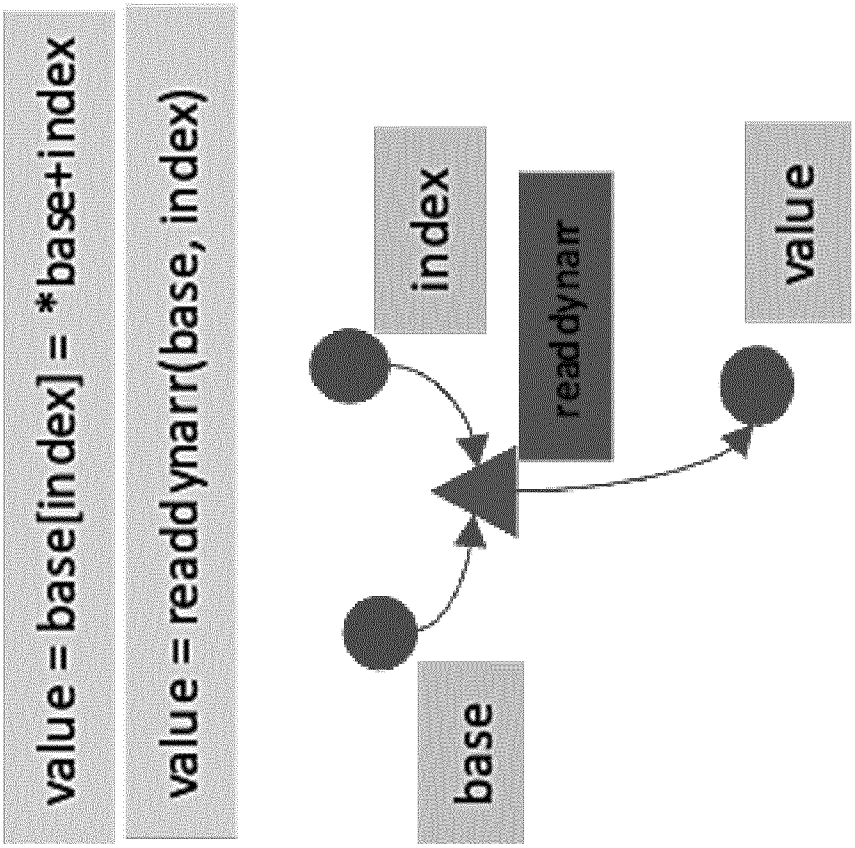

value = base[index] = *base+index
value = readdynarr(base, index)
index
readdynarr
value
base

Fig. 56

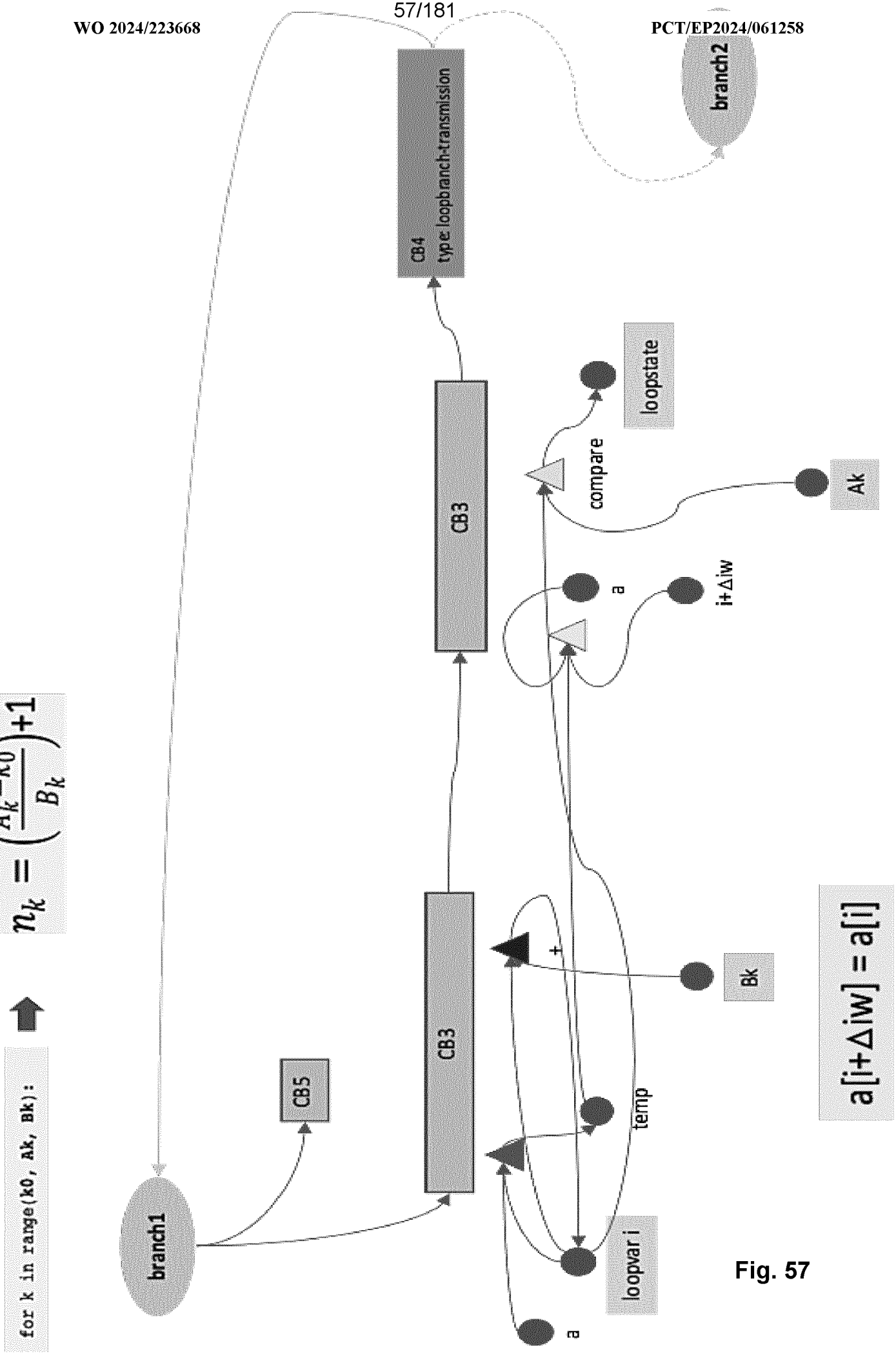

for k in range(k0, Ak, Bk):
$n_k = \left(\frac{A_k - k_0}{B_k}\right) + 1$
branch1
CB5
CB3
CB3
CB4
type loopbranch-transmission
branch2
loopstate
compare
Ak
a
i+Δiw
Bk
+
temp
loopvar i
a
a[i+Δiw] = a[i]

Fig. 57

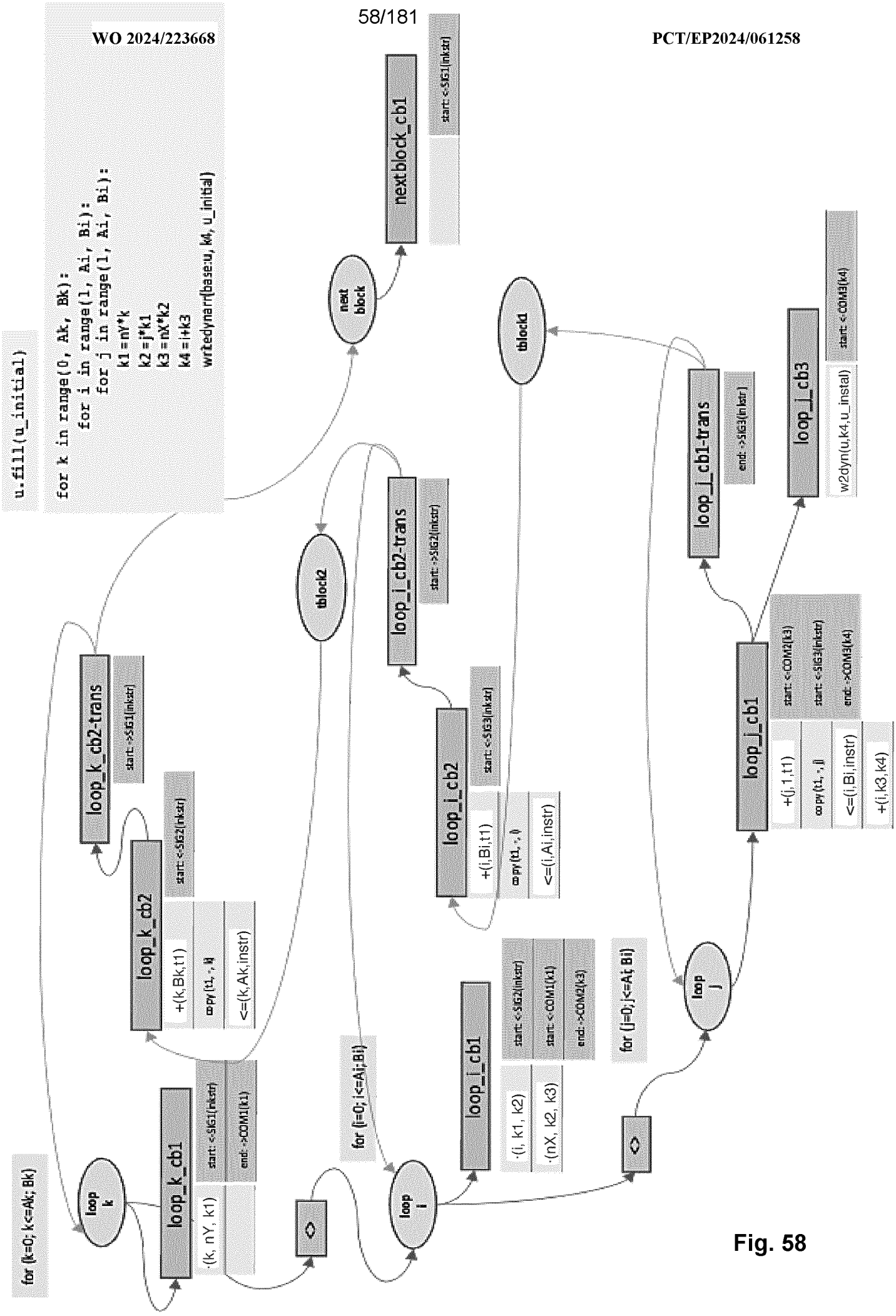

u.fill(u_initial)
for k in range(0, Ak, Bk):
for i in range(l, Ai, Bi):
for j in range(l, Ai, Bi):
k1 = nY*k
k2 = j*k1
k3 = nX*k2
k4 = i+k3
writedynarr(base:u, k4, u_inibial)
nextblock_cb1
start: ->SIG1[instr]
next block
tblock1
tblock2
loop_k_cb2-trans
start: ->SIG1[instr]
loop_k_cb2
start: ->SIG2[instr]
+(k, Bk, t1)
copy(t1, *, 4)
<=(k, Ak, instr)
loop_k_cb1
start: ->SIG1[instr]
end: ->COM1[k1]
·(k, nY, k1)
loop k
<>
for (i=0; i<=Ai; Bi)
loop_i_cb2-trans
start: ->SIG2[instr]
loop_i_cb2
start: ->SIG3[instr]
+(i, Bi, t1)
copy(t1, *, 4)
<=(i, Ai, instr)
loop_i_cb1
start: ->SIG2[instr]
start: ->COM1[k1]
end: ->COM2[k3]
·(i, k1, k2)
·(nX, k2, k3)
loop i
for (j=0; j<=Ai Bi)
loop_j_cb1-trans
end: ->SIG3[instr]
loop_j_cb1
start: ->COM2[k3]
start: ->SIG3[instr]
end: ->COM3[k4]
+(j, 1, t1)
copy(t1, *, j)
<=(i, Bi, instr)
+(i, k3, k4)
loop j
loop_j_cb3
start: ->COM3[k4]
w2dyn(u, k4, u_instal)
for (k=0; k<=Ak; Bk)

Fig. 58

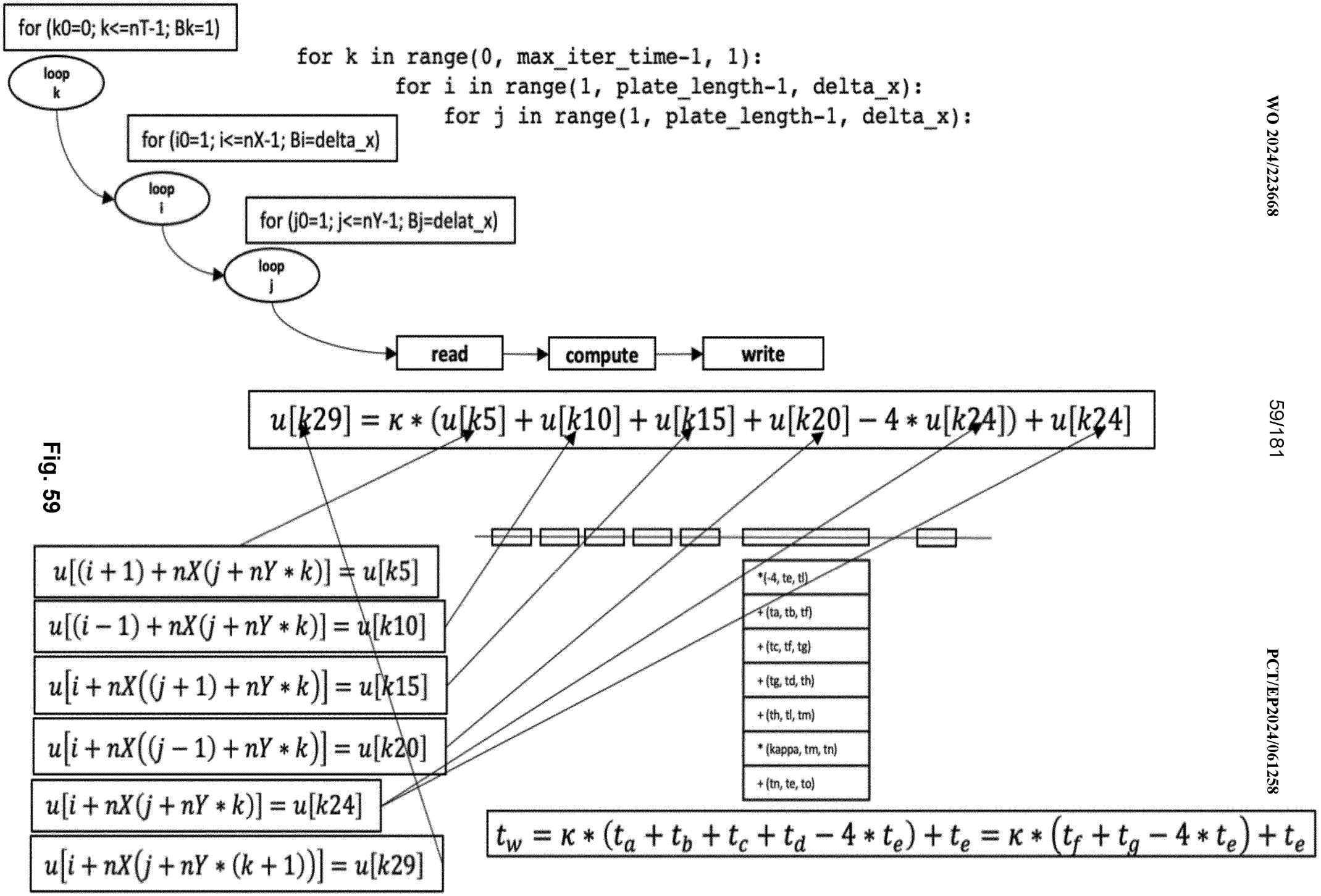

for (k0=0; k<=nT-1; Bk=1)
loop k
for (i0=1; i<=nX-1; Bi=delta_x)
loop i
for (j0=1; j<=nY-1; Bj=delat_x)
loop j
read
compute
write
for k in range(0, max_iter_time-1, 1):
for i in range(1, plate_length-1, delta_x):
for j in range(1, plate_length-1, delta_x):
$u[k29] = \kappa * (u[k5] + u[k10] + u[k15] + u[k20] - 4 * u[k24]) + u[k24]$
$u[(i+1) + nX(j + nY * k)] = u[k5]$
$u[(i-1) + nX(j + nY * k)] = u[k10]$
$u[i + nX((j+1) + nY * k)] = u[k15]$
$u[i + nX((j-1) + nY * k)] = u[k20]$
$u[i + nX(j + nY * k)] = u[k24]$
$u[i + nX(j + nY * (k+1))] = u[k29]$
*(-4, te, tl)
+ (ta, tb, tf)
+ (tc, tf, tg)
+ (tg, td, th)
+ (th, tl, tm)
* (kappa, tm, tn)
+ (tn, te, to)
$t_w = \kappa * (t_a + t_b + t_c + t_d - 4 * t_e) + t_e = \kappa * (t_f + t_g - 4 * t_e) + t_e$

Fig. 59

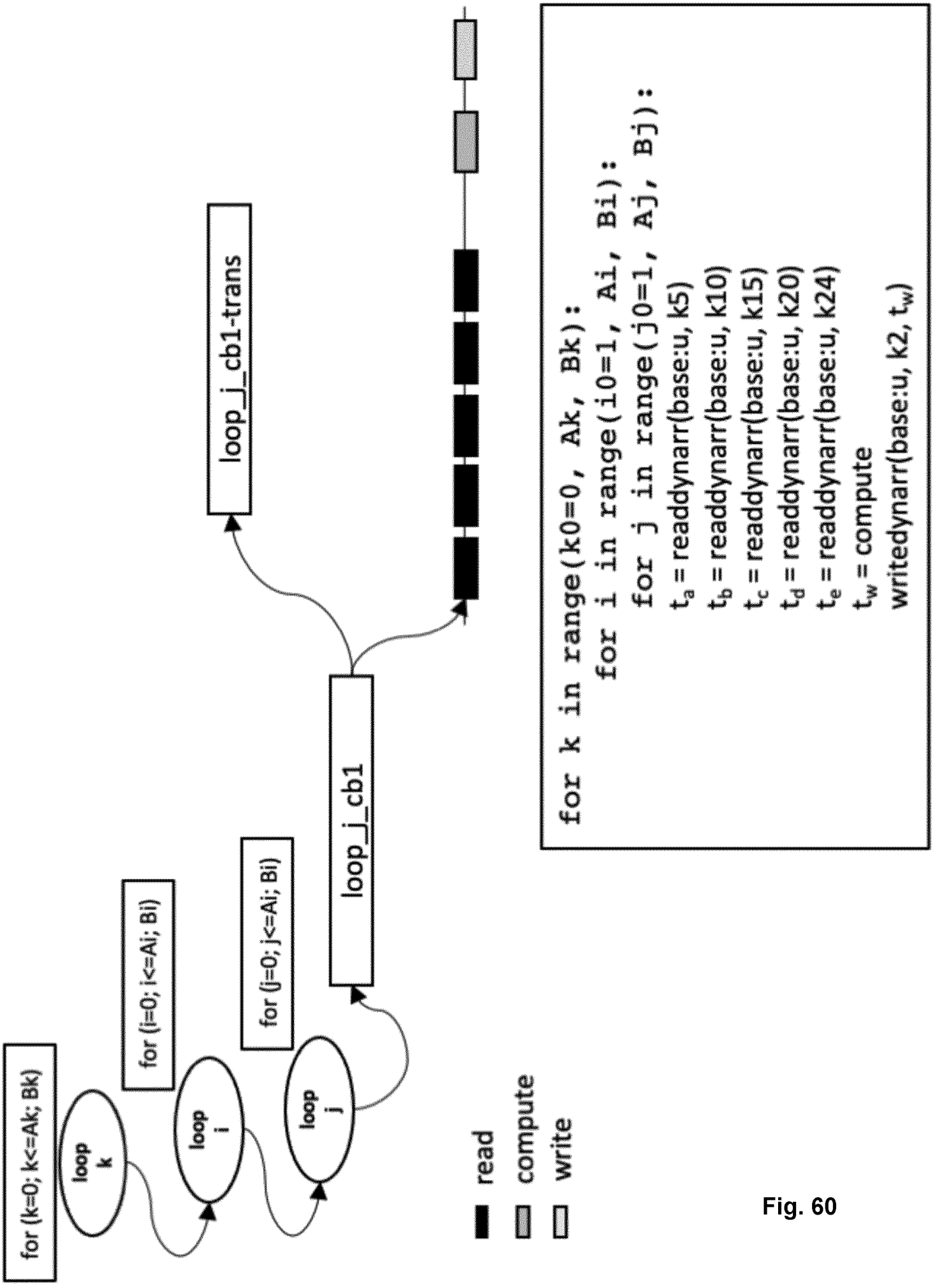

for (k=0; k<=Ak; Bk)
loop k
for (i=0; i<=Ai; Bi)
loop i
for (j=0; j<=Ai; Bi)
loop j
loop_j_cb1
loop_j_cb1-trans
read
compute
write
for k in range(k0=0, Ak, Bk):
for i in range(i0=1, Ai, Bi):
for j in range(j0=1, Aj, Bj):
t_a = readdynarr(base:u, k5)
t_b = readdynarr(base:u, k10)
t_c = readdynarr(base:u, k15)
t_d = readdynarr(base:u, k20)
t_e = readdynarr(base:u, k24)
t_w = compute
writedynarr(base:u, k2, t_w)

Fig. 60

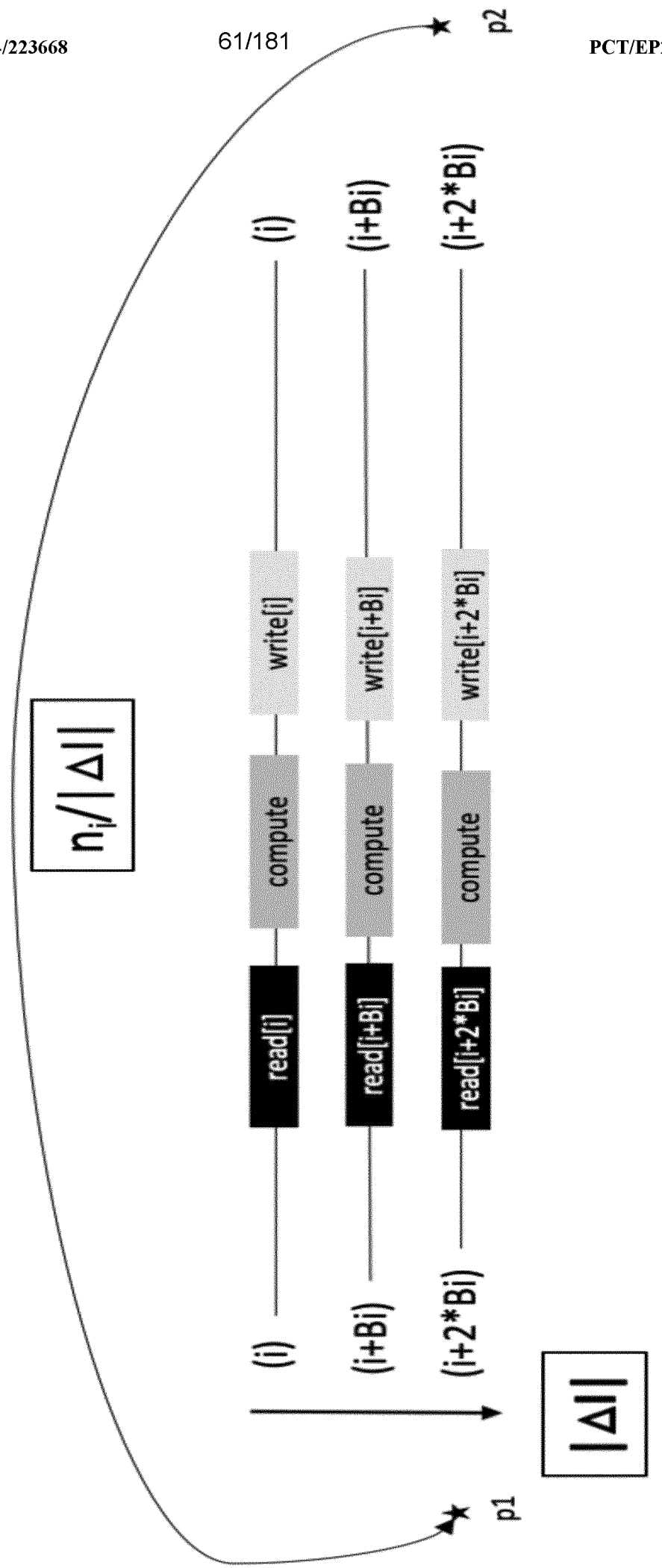
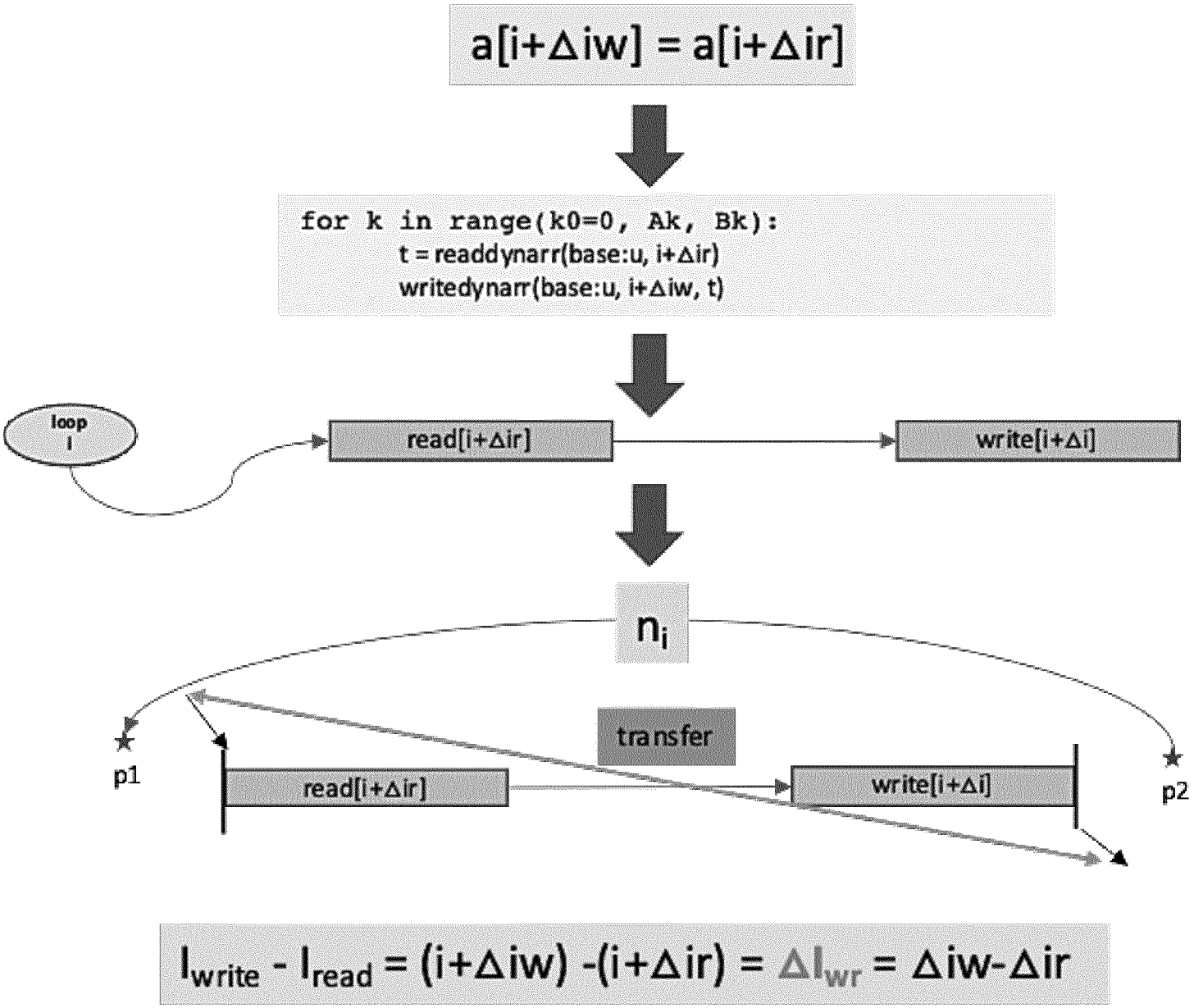

p2
p1
$n_i/|\Delta I|$
$|\Delta I|$
(i)
(i+Bi)
(i+2*Bi)
read[i]
read[i+Bi]
read[i+2*Bi]
compute
compute
compute
write[i]
write[i+Bi]
write[i+2*Bi]

Fig. 61



$$I_{write} - I_{read} = (i+\Delta iw) - (i+\Delta ir) = \Delta I_{wr} = \Delta iw - \Delta ir$$

Fig. 62

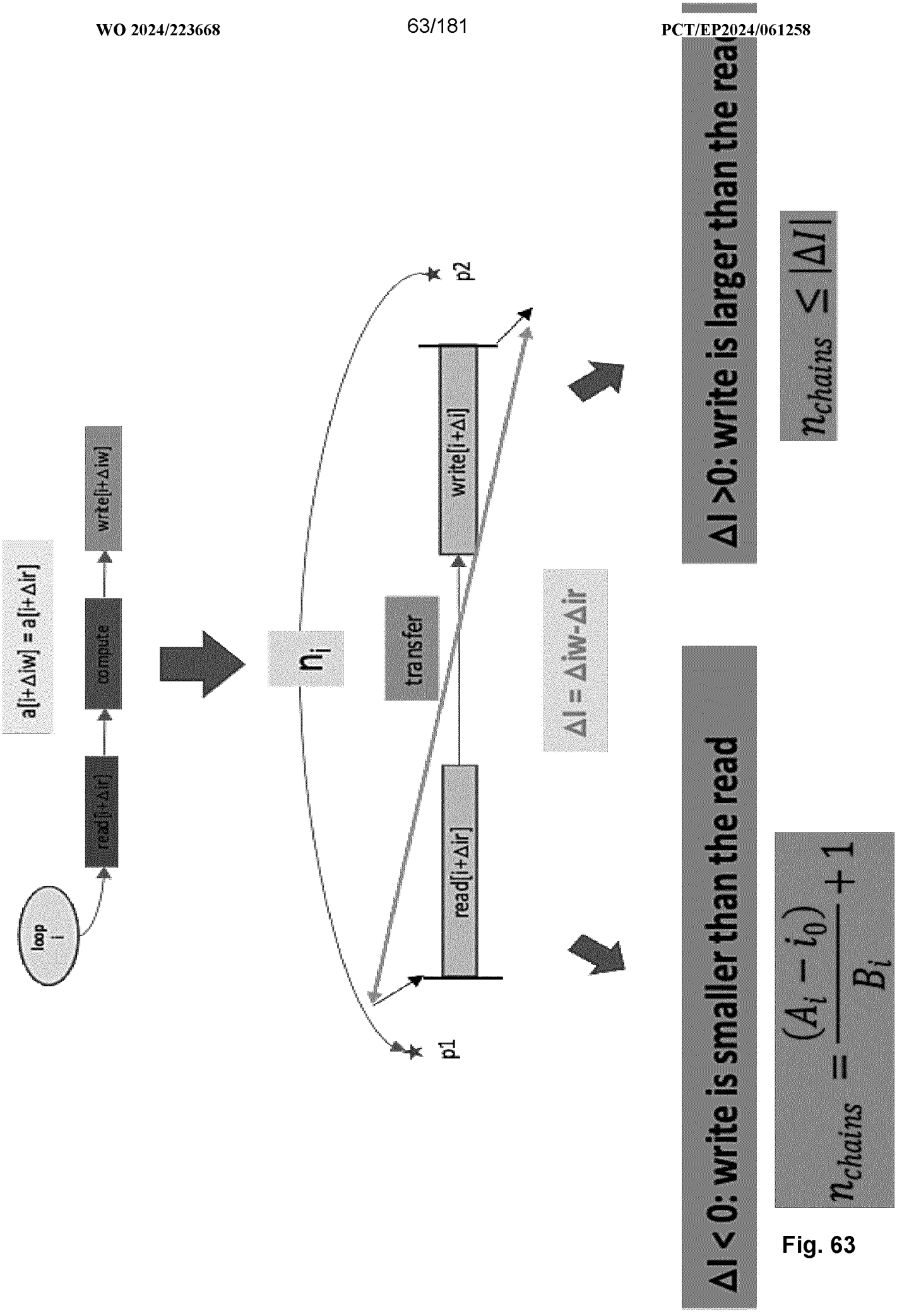

loop i
read[i+Δir]
compute
write[i+Δiw]
a[i+Δiw] = a[i+Δir]
n_i
p1
p2
transfer
read[i+Δir]
write[i+Δi]
ΔI = Δiw-Δir
ΔI <0: write is smaller than the read
ΔI >0: write is larger than the rea
$n_{chains} \leq |\Delta I|$
$n_{chains} = \frac{(A_i - i_0)}{B_i} + 1$

Fig. 63

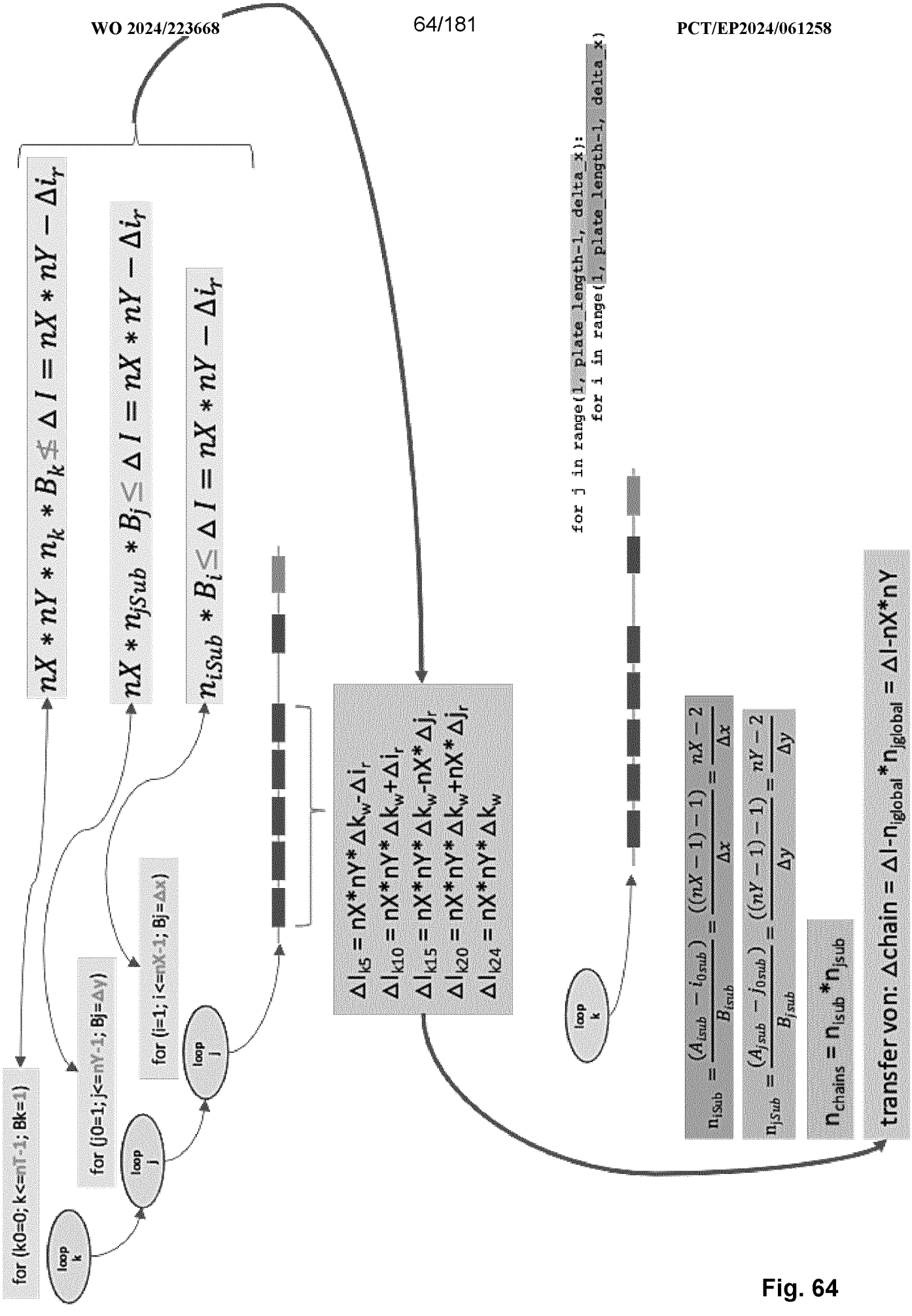

for (k0=0; k<=nT-1; Bk=1)
loop k
for (j0=1; j<=nY-1; Bj=ΔY)
loop j
for (i=1; i <=nX-1; Bj=Δx)
loop j
$nX * nY * n_k * B_k \nleq \Delta I = nX * nY - \Delta i_r$
$nX * n_{jSub} * B_j \leq \Delta I = nX * nY - \Delta i_r$
$n_{iSub} * B_i \leq \Delta I = nX * nY - \Delta i_r$
$\Delta I_{k5} = nX*nY*\Delta k_w - \Delta i_r$
$\Delta I_{k10} = nX*nY*\Delta k_w + \Delta i_r$
$\Delta I_{k15} = nX*nY*\Delta k_w - nX*\Delta j_r$
$\Delta I_{k20} = nX*nY*\Delta k_w + nX*\Delta j_r$
$\Delta I_{k24} = nX*nY*\Delta k_w$
for j in range(1, plate_length-1, delta_x):
for i in range(1, plate_length-1, delta_x)
loop k
$n_{iSub} = \frac{(A_{iSub} - i_{0Sub})}{B_{iSub}} = \frac{((nX-1)-1)}{\Delta x} = \frac{nX-2}{\Delta x}$
$n_{jSub} = \frac{(A_{jSub} - j_{0Sub})}{B_{jSub}} = \frac{((nY-1)-1)}{\Delta y} = \frac{nY-2}{\Delta y}$
$n_{chains} = n_{iSub} * n_{jSub}$
transfer von: $\Delta chain = \Delta I - n_{jglobal} * n_{jglobal} = \Delta I - nX * nY$

Fig. 64

Δxchain
gap
Δd_{i→j}
for (j=0; j<=Aj; Bj)
loop j
for (i=0; i<=Ai; Bi)
loop i
Δd_{i→j}
nY_Sub
n_iSub=nX_Sub
gap
n_iSub
«i-loop»
«j-loop»
gap start i-loop
$\Delta s_i = \frac{(i_{0sub} - i_0)}{B_{isub}} = \frac{1}{\Delta x}$
gap end i-loop
$\Delta e_i = \frac{(A_i - A_{isub})}{B_{isub}} = \frac{1}{\Delta x}$
«symmetry-distance»
$\Delta d_{i\to j} = n_{isub} * B_{isub} + \Delta e_i + \Delta s_i = \left(\frac{(A_{isub} - i_{0sub})}{B_{isub}}\right) * B_{isub} + \frac{(A_i - A_{isub})}{B_{isub}} + \frac{(i_{0sub} - i_0)}{B_{isub}} = \frac{nX - 2}{\Delta x} + \frac{2}{\Delta x}$

Fig. 65

$\Delta s_i = \frac{1}{\Delta x}$
$\Delta e_i = \frac{1}{\Delta x}$
$\Delta d_{i\to j} = \frac{nX-2}{\Delta x} + \frac{2}{\Delta x}$
$\Delta s_i = \frac{1}{\Delta x} < \Delta chain < \frac{nX-2}{\Delta x} + \frac{1}{\Delta x}$
$\Phi = \Delta chain$
$\Omega = 0$
divmod(|Δchain|, Δdj->i) = (Ω, Φ)
$\Phi \leq 0$
AND
$|\Phi| \leq \frac{1}{\Delta y}$
#gaphit = |Φ|
$\Phi \geq nY - 2 + \Delta y$
#gaphit = |Φ| − nY − 2 + Δy
#gaphit = 0
$n_{internal_total} = \lceil n_i - \#gaphit \rceil * \lceil j - \Omega \rceil = \left\lceil \frac{nX-2}{\Delta x} - \#gaphit \right\rceil * \left\lceil \frac{nY-2}{\Delta y} - \Omega \right\rceil$
$n_{external_total} = n_j * n_i - n_{internal}$
Transfers / Kommunikation
hilbap
n_j
->CONST
+ sign
core
core
- sign
Ω
n_j
Ω
CONST
CONST
$n_{external} = \#gaphit$
$n_{internal} = n_i - \#gaphit$
$n_{external} = \#gaphit$
->CONST
->Φ
->CONST
+ sign
- sign
<-Φ

Fig. 66

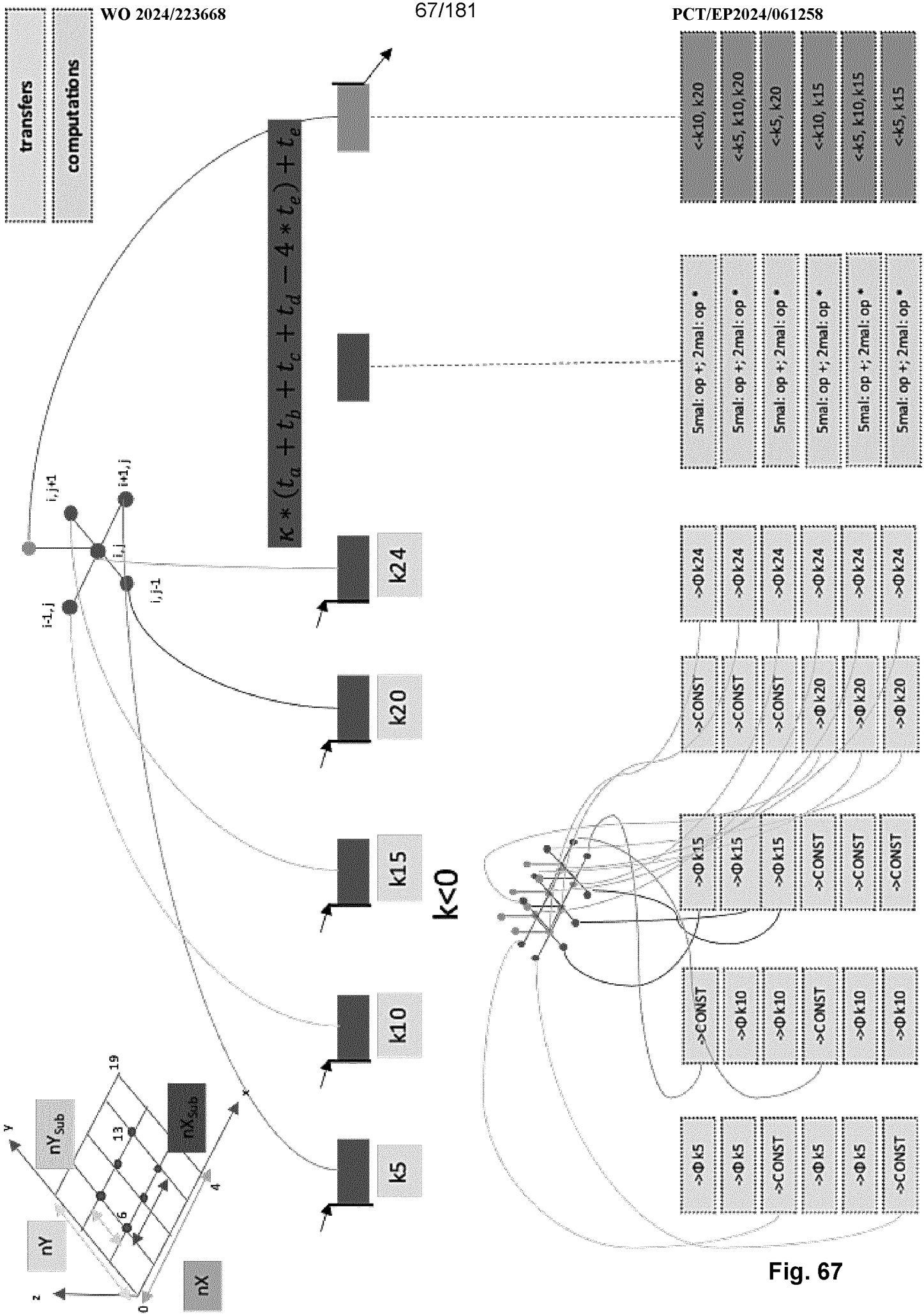

transfers
computations
$K * (t_a + t_b + t_c + t_d - 4 * t_e) + t_e$
k24
k20
k15
k10
k5
k<0
i-1,j
i,j+1
i,j
i+1,j
i,j-1
nX
nY
nZ
nX_Sub
nY_Sub
0
4
6
13
19
x
y
z
<-k10, k20
<-k5, k10,k20
<-k5, k20
<-k10, k15
<-k5, k10,k15
<-k5, k15
5mal: op +; 2mal: op *
->Φ k24
->CONST
->Φ k20
->Φ k15
->Φ k10
->Φ k5

Fig. 67

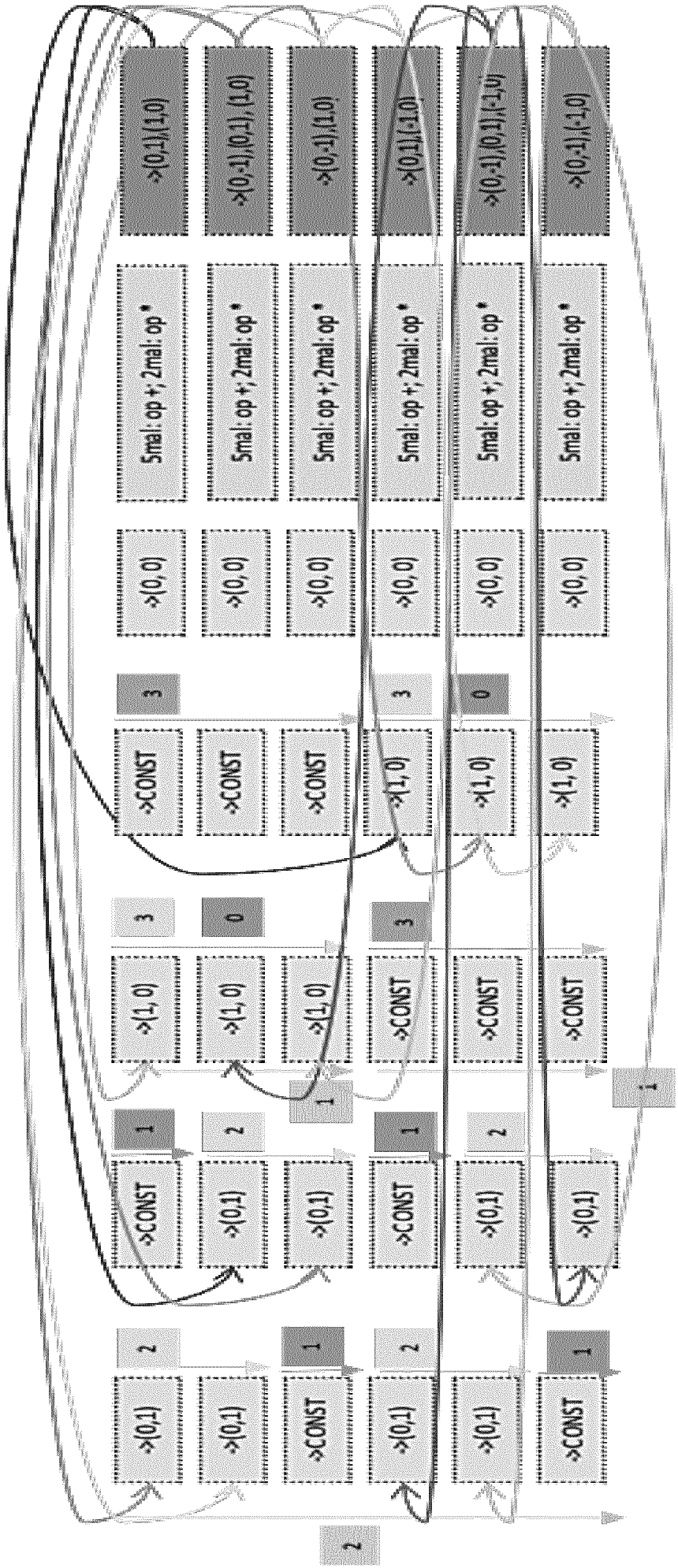


Fig. 68

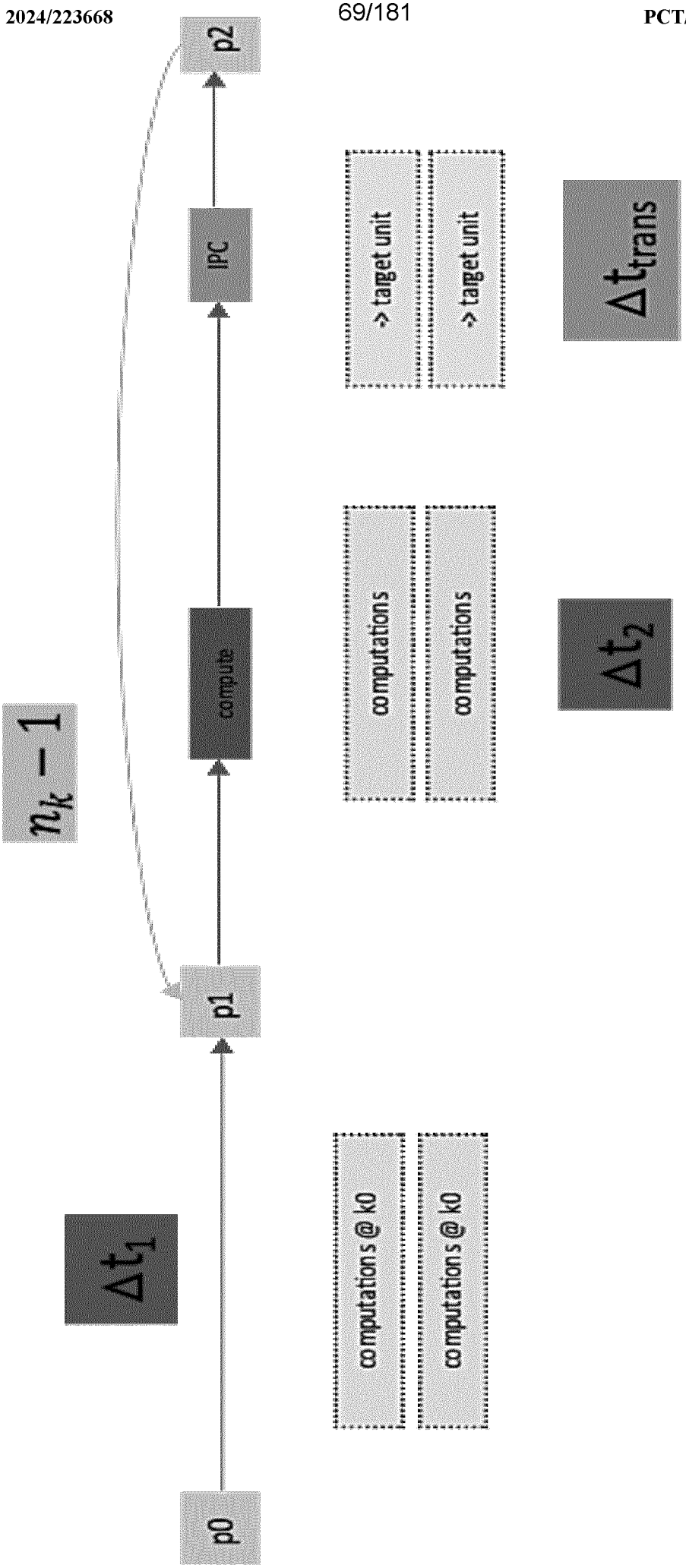

p2
IPC
compute
p1
p0
$n_k - 1$
Δt_1
computations @ k0
computations @ k0
computations
computations
-> target unit
-> target unit
Δt_2
Δt_{trans}

Fig. 69

2->1	0	4->1	0	1->1	5mal: op +; 2mal: op *	send: ->2, ->4
3->2	1->2	5->2	0	2->2	5mal: op +; 2mal: op *	send: ->1, ->3, ->5
0	2->3	6->3	0	3->3	5mal: op +; 2mal: op *	send: ->2, ->6
5->4	0	0	1->4	4->4	5mal: op +; 2mal: op *	send: ->4, ->1
6->5	4->5	0	2->5	5->5	5mal: op +; 2mal: op *	send: ->4, ->6, ->2
0	5->6	0	3->6	6->6	5mal: op +; 2mal: op *	send: ->5, ->3

$$\begin{pmatrix} 2\to1 \\ 3\to2 \\ 0 \\ 5\to4 \\ 6\to4 \\ 0 \end{pmatrix} \cdot \Delta t_{trans}$$

$$\begin{pmatrix} 0 \\ 1\to2 \\ 2\to3 \\ 0 \\ 4\to5 \\ 5\to6 \end{pmatrix} \cdot \Delta t_{trans}$$

$$\begin{pmatrix} 4\to1 \\ 5\to2 \\ 6\to3 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot \Delta t_{trans}$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1\to4 \\ 2\to5 \\ 3\to6 \end{pmatrix} \cdot \Delta t_{trans}$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot \Delta t_{trans}$$

$$\begin{pmatrix} \Gamma \\ \Gamma \\ \Gamma \\ \Gamma \\ \Gamma \\ \Gamma \end{pmatrix} \cdot \Delta t_{latency}$$

$$\Gamma = 5+, 2*$$

$$transfer_size = \begin{pmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{pmatrix} float_size$$

$$cache_size = \begin{pmatrix} 6 \\ 6 \\ 6 \\ 6 \\ 6 \\ 6 \end{pmatrix} float_size$$

meta-values

Fig. 70

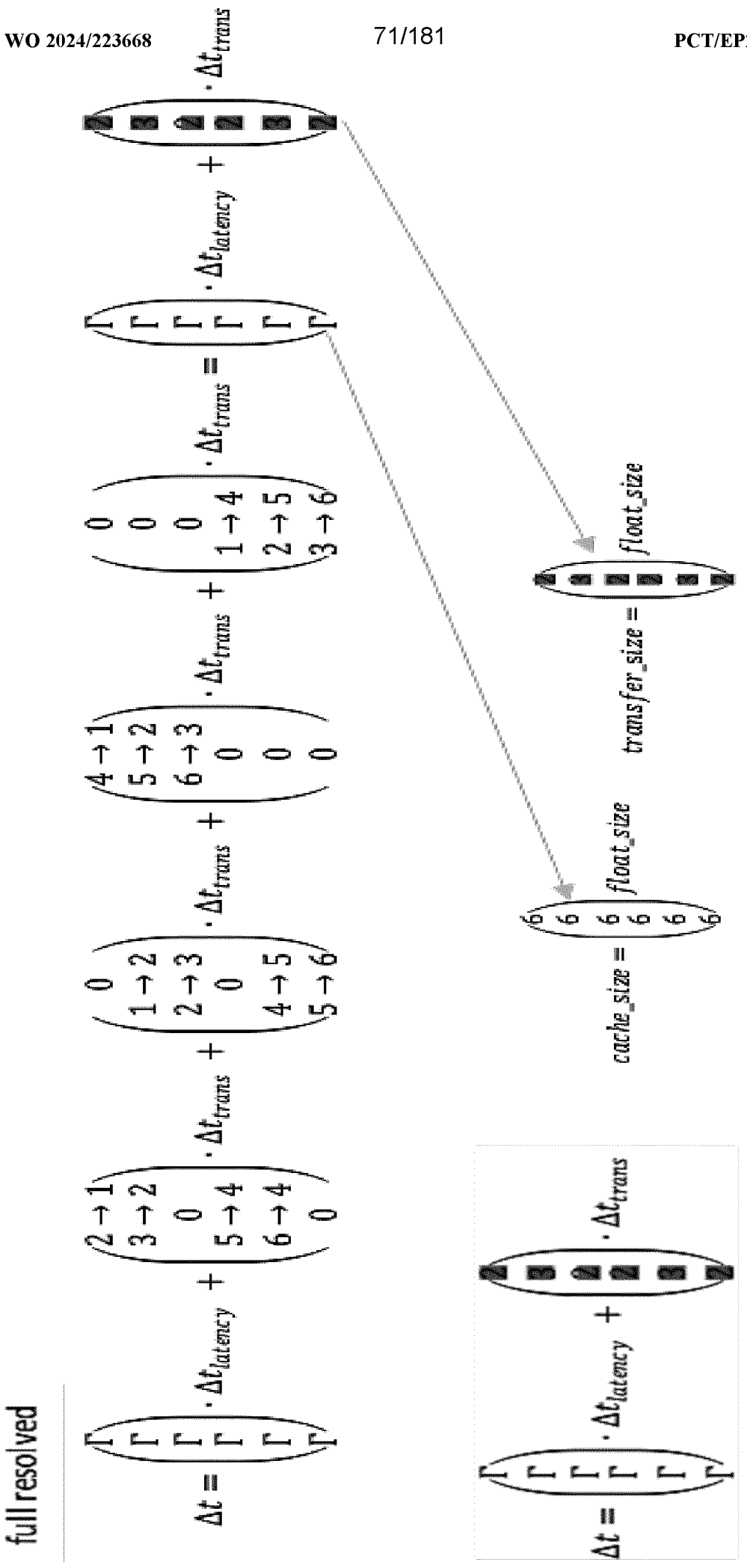
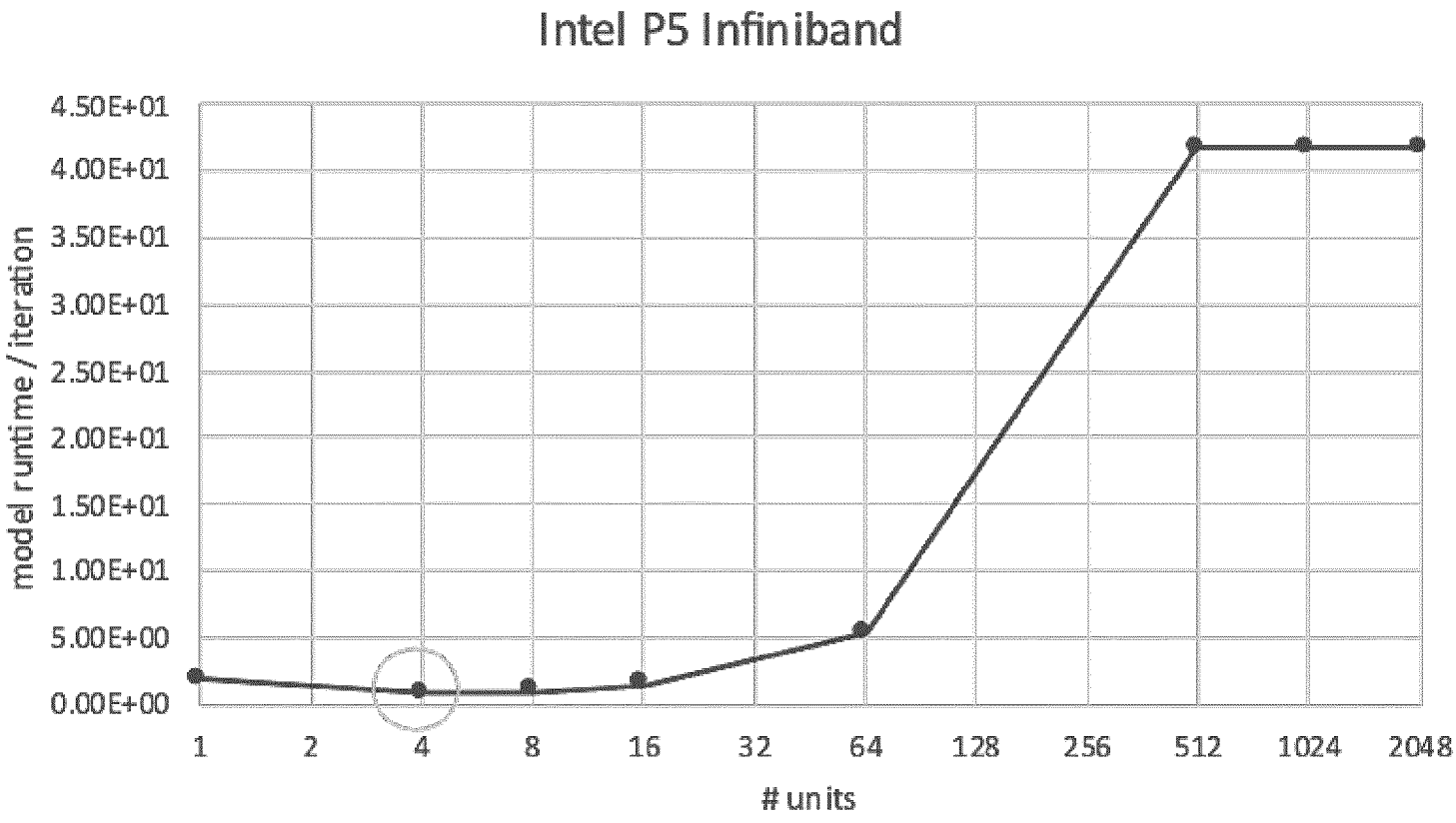

full resolved
$\Delta t = \Gamma_{\Gamma\Gamma\Gamma\Gamma\Gamma\Gamma} \cdot \Delta t_{latency} + \begin{pmatrix} 2\to1 & & & \\ 3\to2 & 0 & & \\ & 5\to4 & 6\to4 & \\ & & 0 & \end{pmatrix} \cdot \Delta t_{trans} + \begin{pmatrix} 0 & & & \\ 1\to2 & 2\to3 & & \\ & 0 & 4\to5 & \\ & & 5\to6 & \end{pmatrix} \cdot \Delta t_{trans} + \begin{pmatrix} 4\to1 & & & \\ 5\to2 & 6\to3 & & \\ & 0 & 0 & \\ & & 0 & \end{pmatrix} \cdot \Delta t_{trans} + \begin{pmatrix} 0 & & & \\ & 0 & 1\to4 & \\ & & 2\to5 & \\ & & 3\to6 & \end{pmatrix} \cdot \Delta t_{trans} = \Gamma_{\Gamma\Gamma\Gamma\Gamma\Gamma\Gamma} \cdot \Delta t_{latency} + \begin{pmatrix} 2 & 3 & 2 & 3 & 2 & 3 \\ 3 & 2 & 3 & 2 & 3 & 2 \end{pmatrix} \cdot \Delta t_{trans}$
$cache_size = \begin{pmatrix} 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix} float_size$
$transfer_size = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} float_size$
$\Delta t = \Gamma_{\Gamma\Gamma\Gamma\Gamma\Gamma\Gamma} \cdot \Delta t_{latency} + \begin{pmatrix} 3 & 2 & 3 & 2 & 3 & 2 \\ 2 & 3 & 2 & 3 & 2 & 3 \end{pmatrix} \cdot \Delta t_{trans}$

Fig. 71


Intel P5 Infiniband
model runtime / iteration
4.50E+01
4.00E+01
3.50E+01
3.00E+01
2.50E+01
2.00E+01
1.50E+01
1.00E+01
5.00E+00
0.00E+00
1
2
4
8
16
32
64
128
256
512
1024
2048
units

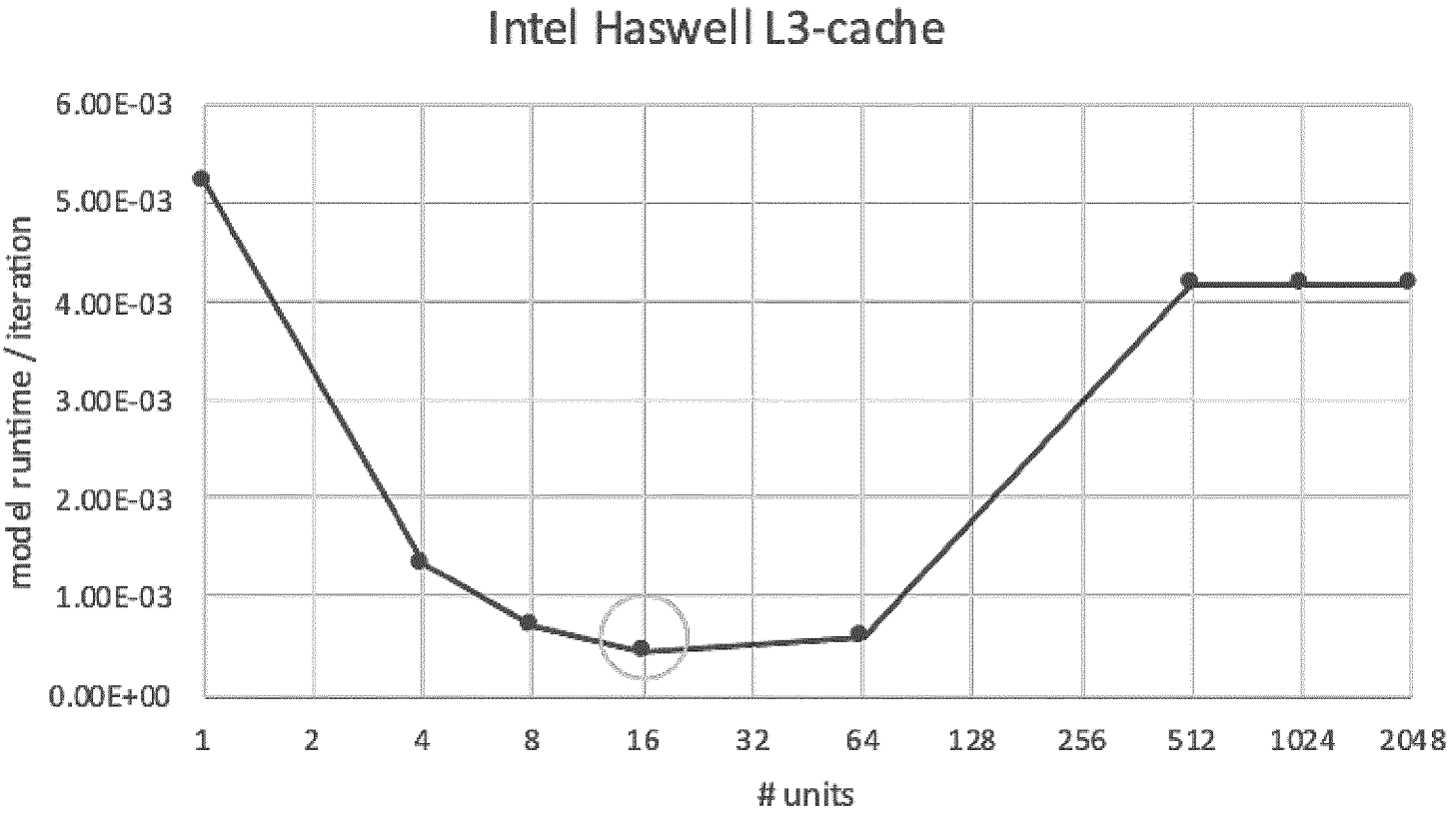
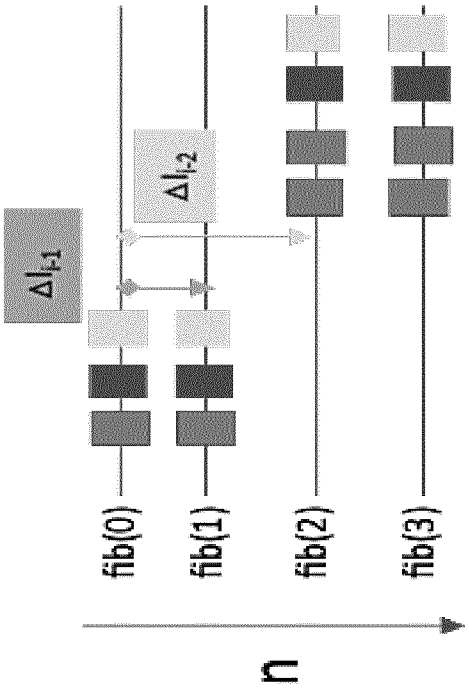

Intel Haswell L3-cache
model runtime / iteration
6.00E-03
5.00E-03
4.00E-03
3.00E-03
2.00E-03
1.00E-03
0.00E+00
1
2
4
8
16
32
64
128
256
512
1024
2048
units

Fig. 72



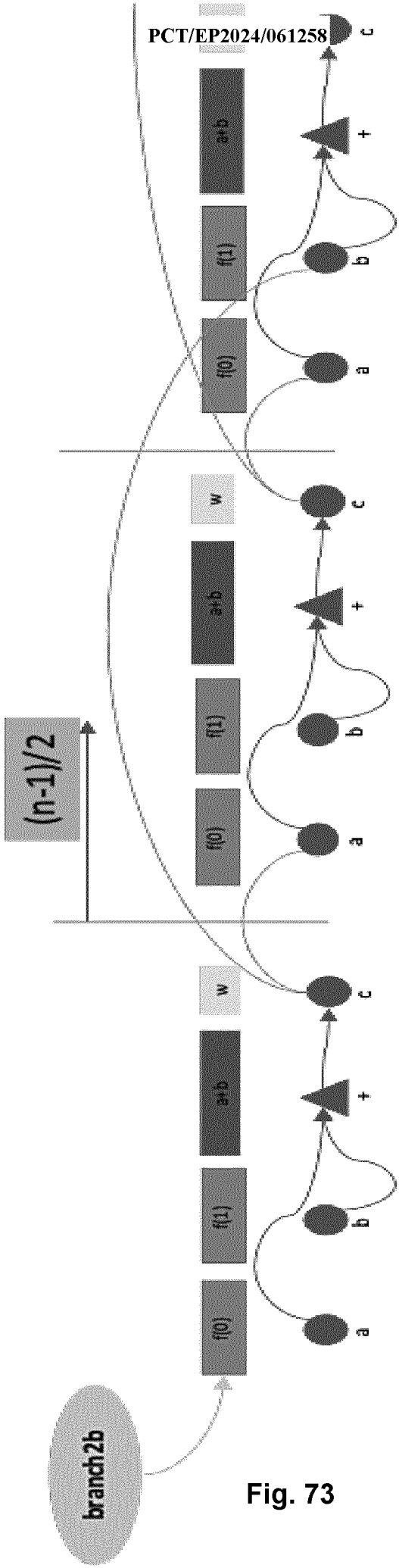


Fig. 73

```
#include<stdio.h>
int f1(int *n, int *b)
{
n[0] = 1;
b[0] = 3;
return n[0]+b[0];
}
int main ()
{
int n[2];
printf("%d", f1(n,n));
getchar();
return 0;
}
```

method

CB1	e: COM1->	s:->COM1	CB3	e: COM2->	s:->COM2	CB8
carr (gld6,2,gld4) call (gld1,ds5,gld7)			warr (gld1,1,0) rarr (gld1,0,gld3) warr (gld2,3,0) rarr (gld2,0,gld4) +(gld3,gld4,gld5)			return(0)-> endof()

```
#include <stdio.h>
int main() {
        int gld6[2];
        int *gld1 = gld6;
        *(gld1+0) = 1;
        int gld3 = *(gld1+0);
        *(gld1+0) = 3;
        int gld4 = *(gld1+0);
        int gld5 = gld3+gld4;
        int gld7 = gld5;
        printf("%d", gld7);
        return 0;
}
```

clang 12.0 with -O3

```
main: # @main
push rax
mov edi, offset .L.str
mov esi, 4
xor eax, eax
call printf
xor eax, eax
pop rcx
ret
.L.str:
.asciz "%d"
```

clang 12.0 with -O3

```
f1: # @f1
mov dword ptr [rdi], 1
mov dword ptr [rsi], 3
mov eax, dword ptr [rdi]
add eax, 3
ret
main: # @main
push rax
mov edi, offset .L.str
mov esi, 6
xor eax, eax
call printf
mov rdi, qword ptr [rip + stdin]
call getc
xor eax, eax
pop rcx
ret
.L.str:
.asciz "%d"
```

Fig. 74

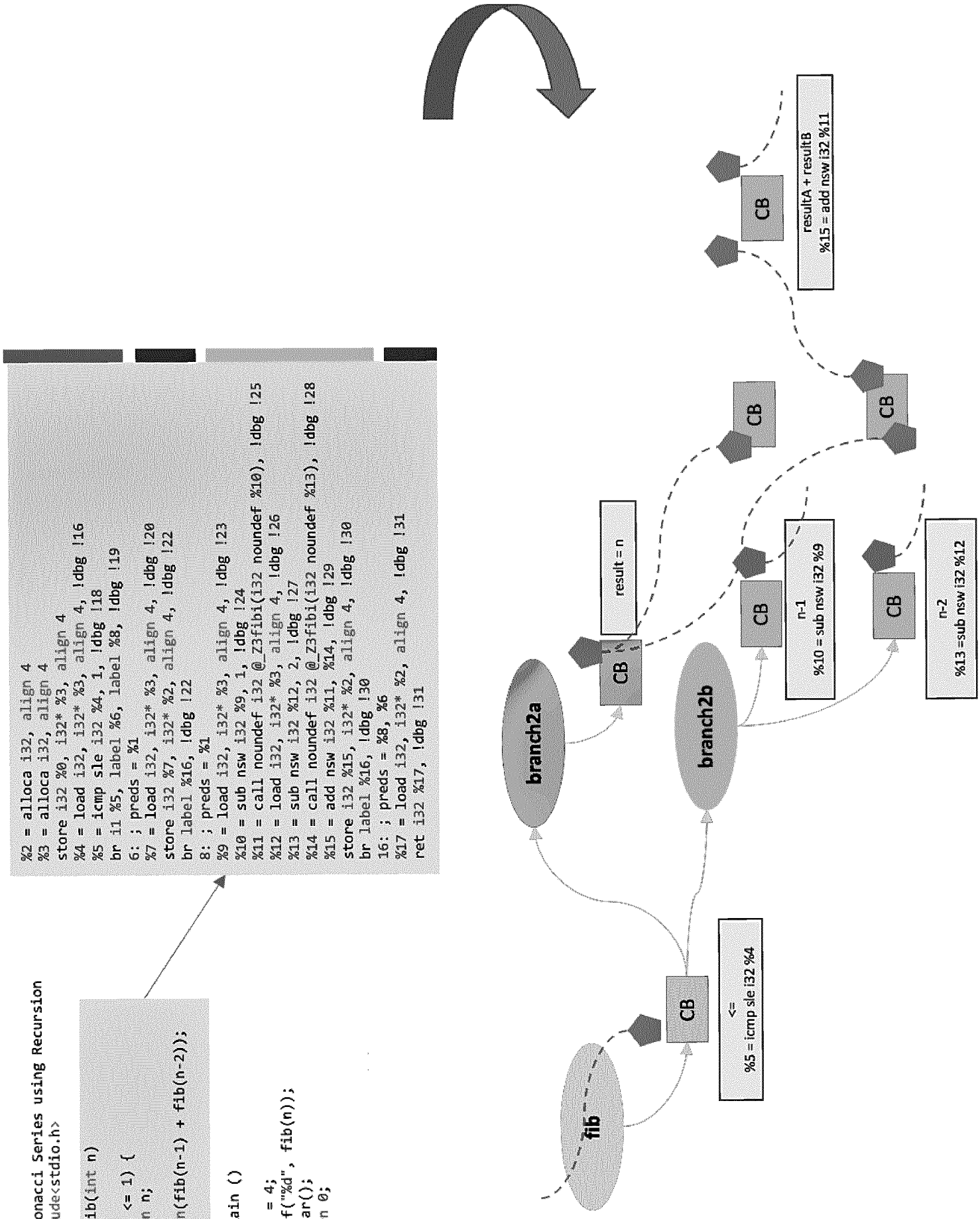

//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
if (n <= 1) {
return n;
}
return(fib(n-1) + fib(n-2));
}
int main ()
{
int n = 4;
printf("%d", fib(n));
getchar();
return 0;
}
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 %0, i32* %3, align 4
%4 = load i32, i32* %3, align 4, !dbg !16
%5 = icmp sle i32 %4, 1, !dbg !18
br i1 %5, label %6, label %8, !dbg !19
6: ; preds = %1
%7 = load i32, i32* %3, align 4, !dbg !20
store i32 %7, i32* %2, align 4, !dbg !22
br label %16, !dbg !22
8: ; preds = %1
%9 = load i32, i32* %3, align 4, !dbg !23
%10 = sub nsw i32 %9, 1, !dbg !24
%11 = call noundef i32 @_Z3fibi(i32 noundef %10), !dbg !25
%12 = load i32, i32* %3, align 4, !dbg !26
%13 = sub nsw i32 %12, 2, !dbg !27
%14 = call noundef i32 @_Z3fibi(i32 noundef %13), !dbg !28
%15 = add nsw i32 %11, %14, !dbg !29
store i32 %15, i32* %2, align 4, !dbg !30
br label %16, !dbg !30
16: ; preds = %8, %6
%17 = load i32, i32* %2, align 4, !dbg !31
ret i32 %17, !dbg !31
fib
CB
<=
%5 = icmp sle i32 %4
branch2a
branch2b
result = n
n-1
%10 = sub nsw i32 %9
n-2
%13 =sub nsw i32 %12
resultA + resultB
%15 = add nsw i32 %11

Fig. 75

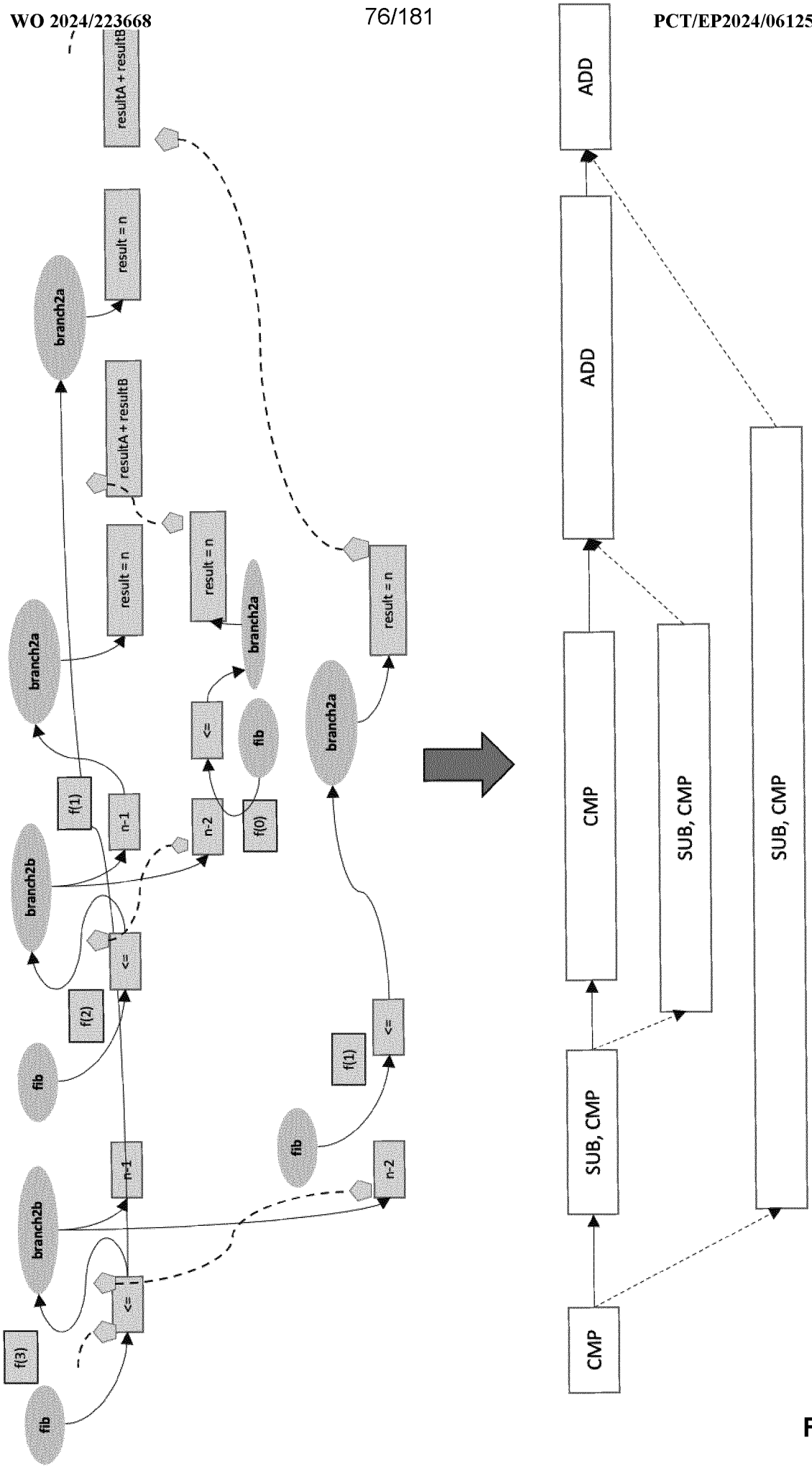
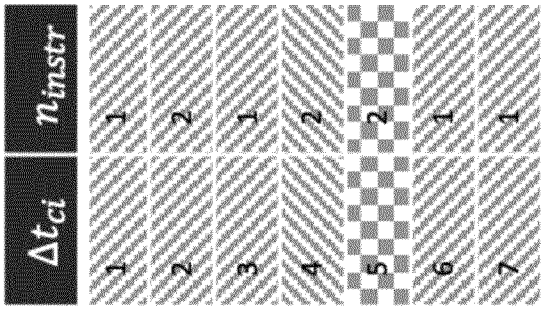

resultA + resultB
result = n
branch2a
resultA + resultB
result = n
result = n
branch2a
branch2a
result = n
<=
fib
branch2a
f(1)
n-1
n-2
f(0)
branch2b
<=
f(2)
fib
<=
f(1)
fib
n-1
n-2
branch2b
<=
f(3)
fib
ADD
ADD
CMP
SUB, CMP
SUB, CMP
SUB, CMP
CMP

Fig. 76


n_{instr}
Δt_{ci}
1 2 1 2 2 1 1
1 2 3 4 5 6 7

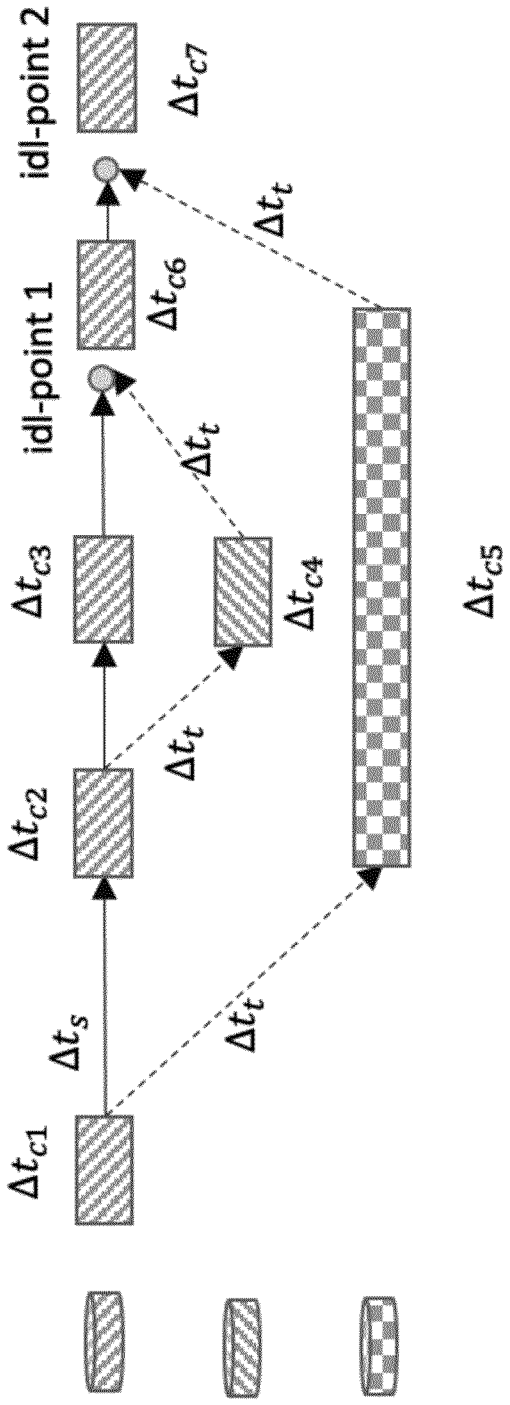
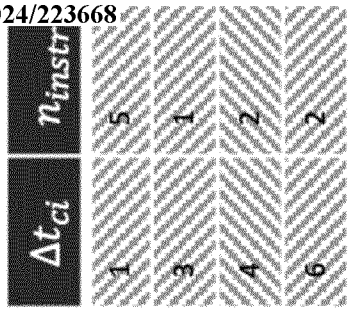
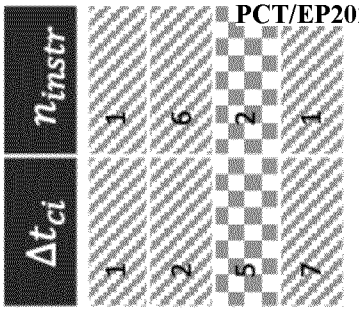
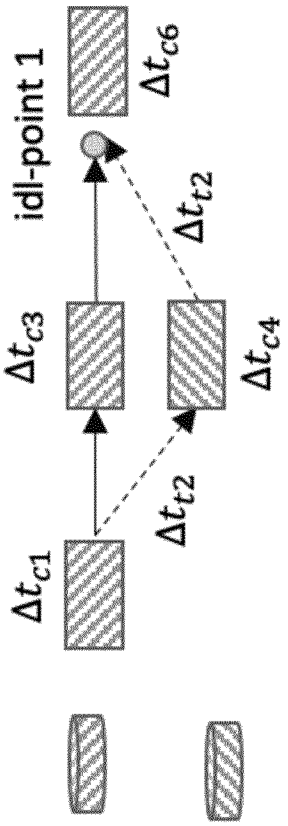

idl-point 2
idl-point 1
Δt_{c1}
Δt_{c2}
Δt_{c3}
Δt_{c4}
Δt_{c5}
Δt_{c6}
Δt_{c7}
Δt_s
Δt_t
Δt_t
Δt_t
Δt_t

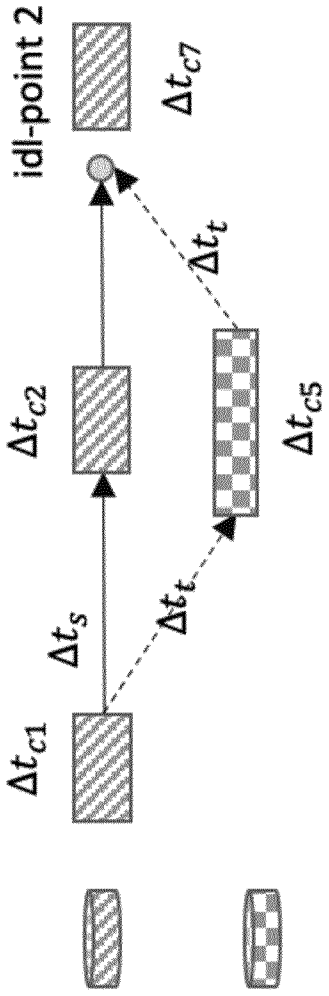
Fig. 77


n_{instr}
Δt_{ci}
5 1 2 2
1 3 4 6


n_{instr}
Δt_{ci}
1 6 2 1
1 2 5 7

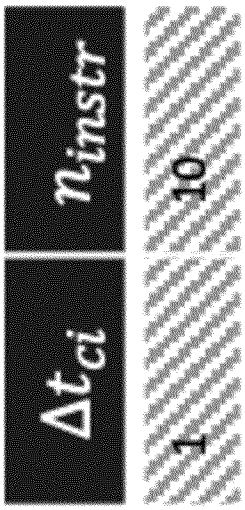

idl-point 1
Δt_{c6}
Δt_{t2}
Δt_{c3}
Δt_{c4}
Δt_{t2}
Δt_{c1}

c1


idl-point 2
Δt_{c7}
Δt_t
Δt_{c2}
Δt_{c5}
Δt_s
Δt_t
Δt_{c1}

c2

Fig. 78



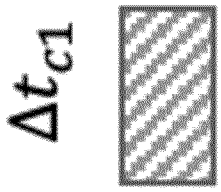




Fig. 79

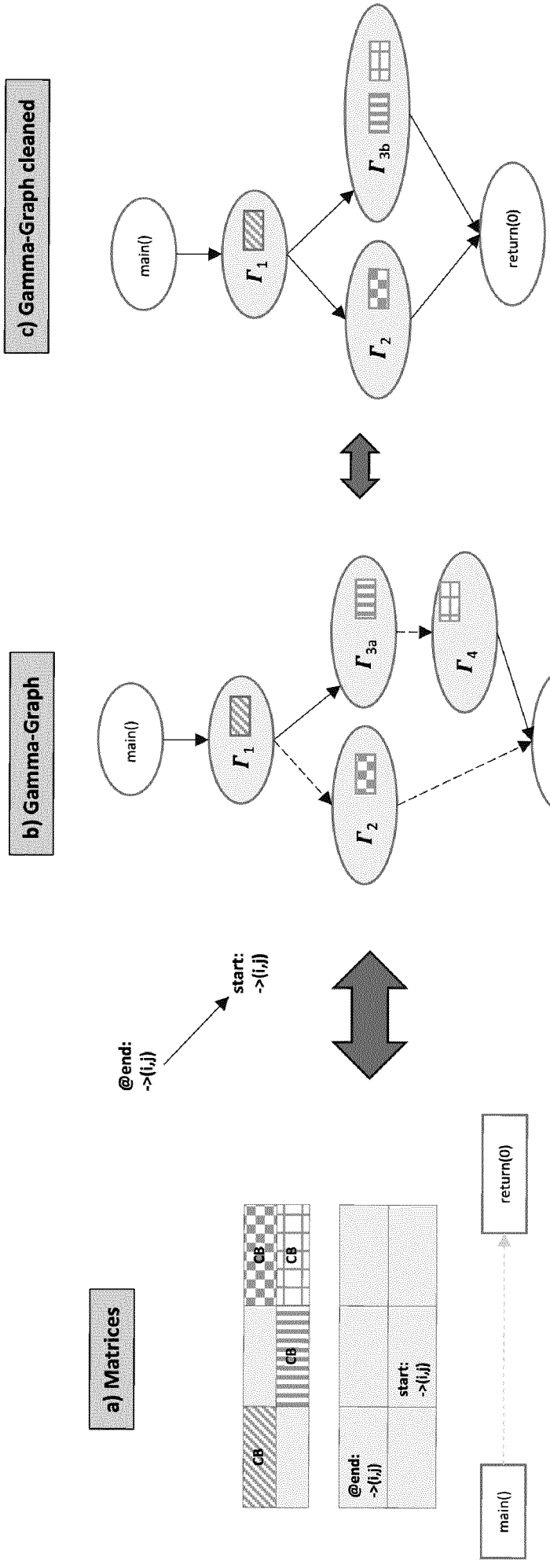

c) Gamma-Graph cleaned
main()
Γ1
Γ2
Γ3b
return(0)
b) Gamma-Graph
main()
Γ1
Γ2
Γ3a
Γ4
return(0)
@end:
->(i,j)
start:
->(i,j)
a) Matrices
CB
CB
CB
@end:
->(i,j)
start:
->(i,j)
return(0)
main()

Fig. 80

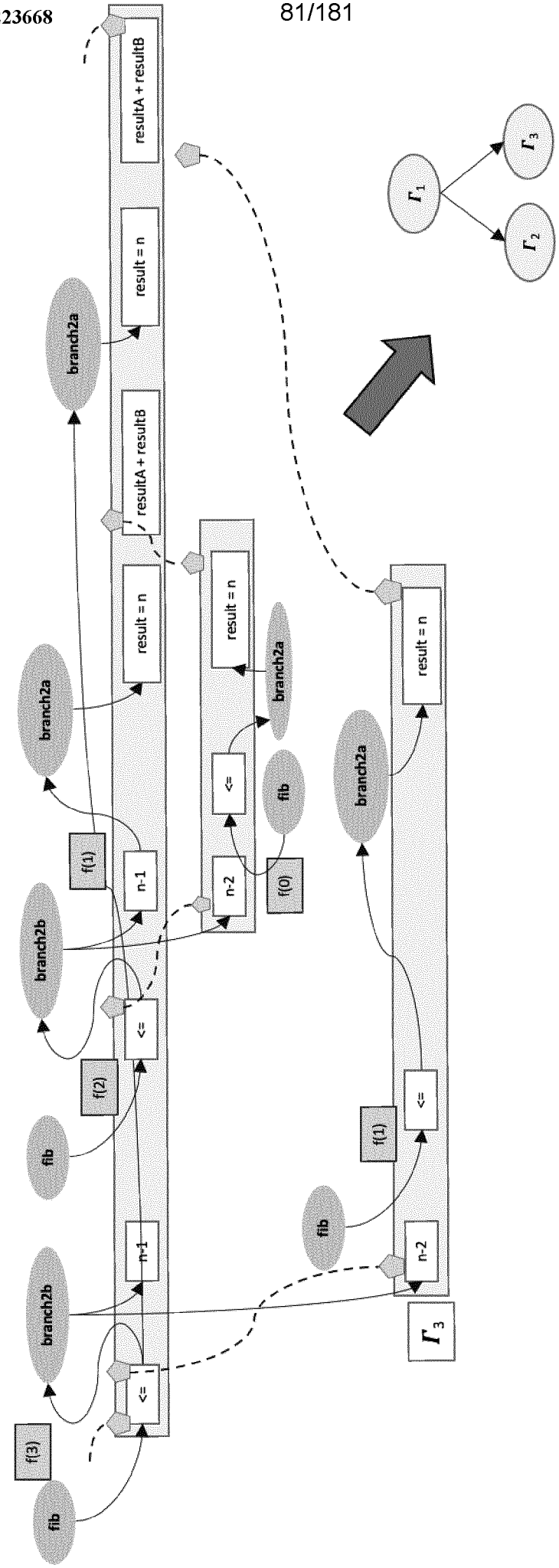
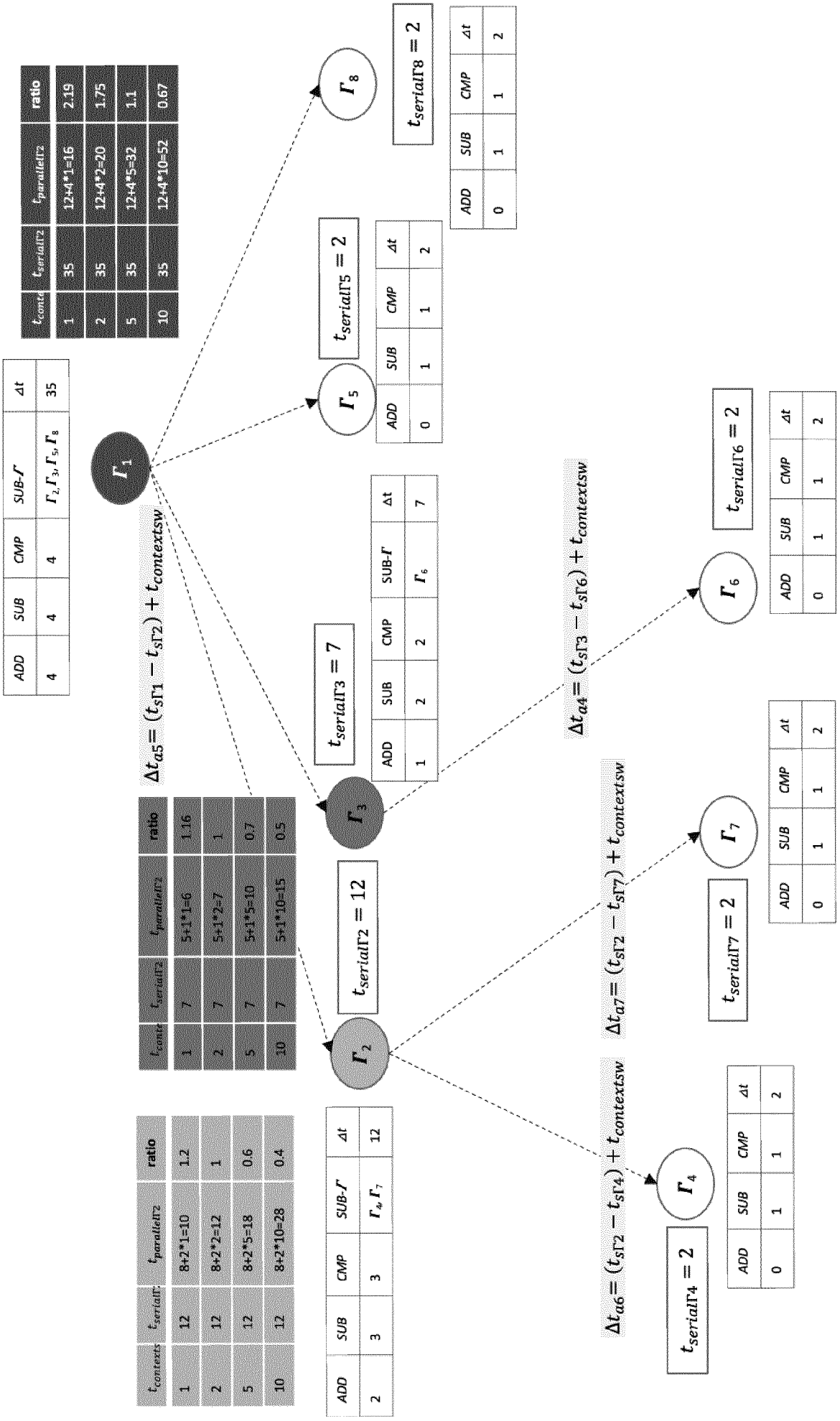

resultA + resultB
result = n
branch2a
resultA + resultB
result = n
branch2a
branch2b
f(1)
n-1
<=
f(2)
fib
n-1
branch2b
<=
f(3)
fib
result = n
branch2a
<=
n-2
f(0)
fib
result = n
branch2a
<=
f(1)
fib
n-2
Γ3
Γ1
Γ3
Γ2

Fig. 81



ADD	SUB	CMP	SUB-Γ	Δt
4	4	4	$\Gamma_2, \Gamma_3, \Gamma_5, \Gamma_8$	35

t_{contx}	$t_{serial\Gamma 2}$	$t_{parallel\Gamma 2}$	ratio
1	35	12+4*1=16	2.19
2	35	12+4*2=20	1.75
5	35	12+4*5=32	1.1
10	35	12+4*10=52	0.67

ADD	SUB	CMP	Δt
0	1	1	2

ADD	SUB	CMP	SUB-Γ	Δt
1	2	2	Γ_6	7

ADD	SUB	CMP	Δt
0	1	1	2

t_{contx}	$t_{serial\Gamma 2}$	$t_{parallel\Gamma 2}$	ratio
1	7	5+1*1=6	1.16
2	7	5+1*2=7	1
5	7	5+1*5=10	0.7
10	7	5+1*10=15	0.5

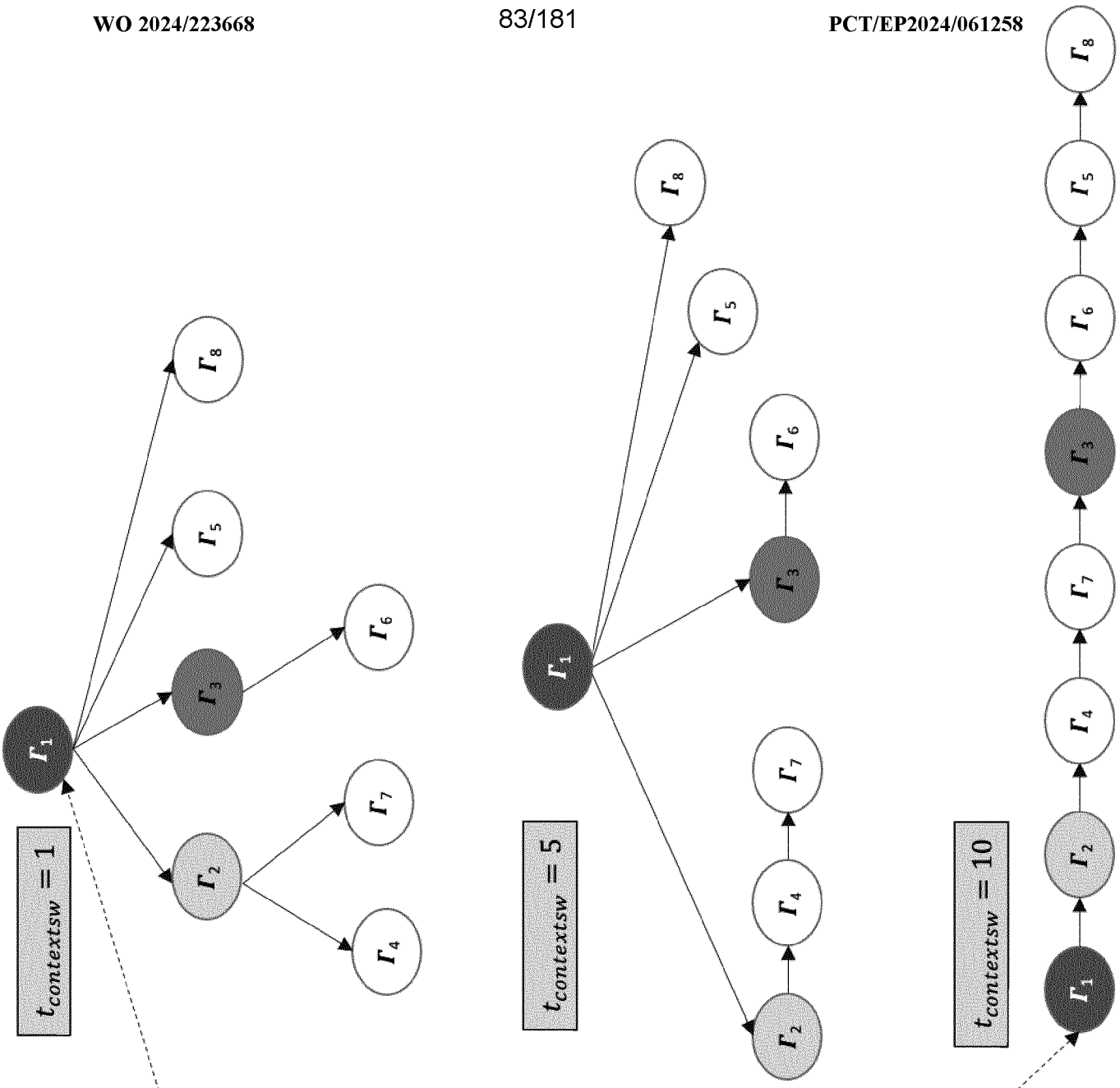
t_{contx}	$t_{serial\Gamma}$	$t_{parallel\Gamma 2}$	ratio
1	12	8+2*1=10	1.2
2	12	8+2*2=12	1
5	12	8+2*5=18	0.6
10	12	8+2*10=28	0.4

ADD	SUB	CMP	SUB-Γ	Δt
2	3	3	Γ_4, Γ_7	12

ADD	SUB	CMP	Δt
0	1	1	2

ADD	SUB	CMP	Δt
0	1	1	2

Fig. 82



t_{conte}	$t_{serialΓ2}$	$t_{parallelΓ2}$	ratio
1	35	12+4*1=16	2.19
2	35	12+4*2=20	1.75
5	35	12+4*5=32	1.1
10	35	12+4*10=52	0.67

t_{conte}	$t_{serialΓ2}$	$t_{parallelΓ2}$	ratio
1	12	8+2*1=10	1.2
2	12	8+2*2=12	1
5	12	8+2*5=18	0.6
10	12	8+2*10=28	0.4

t_{conte}	$t_{serialΓ2}$	$t_{parallelΓ2}$	ratio
1	7	5+1*1=6	1.16
2	7	5+1*2=7	1
5	7	5+1*5=10	0.7
10	7	5+1*10=15	0.5

Fig. 83

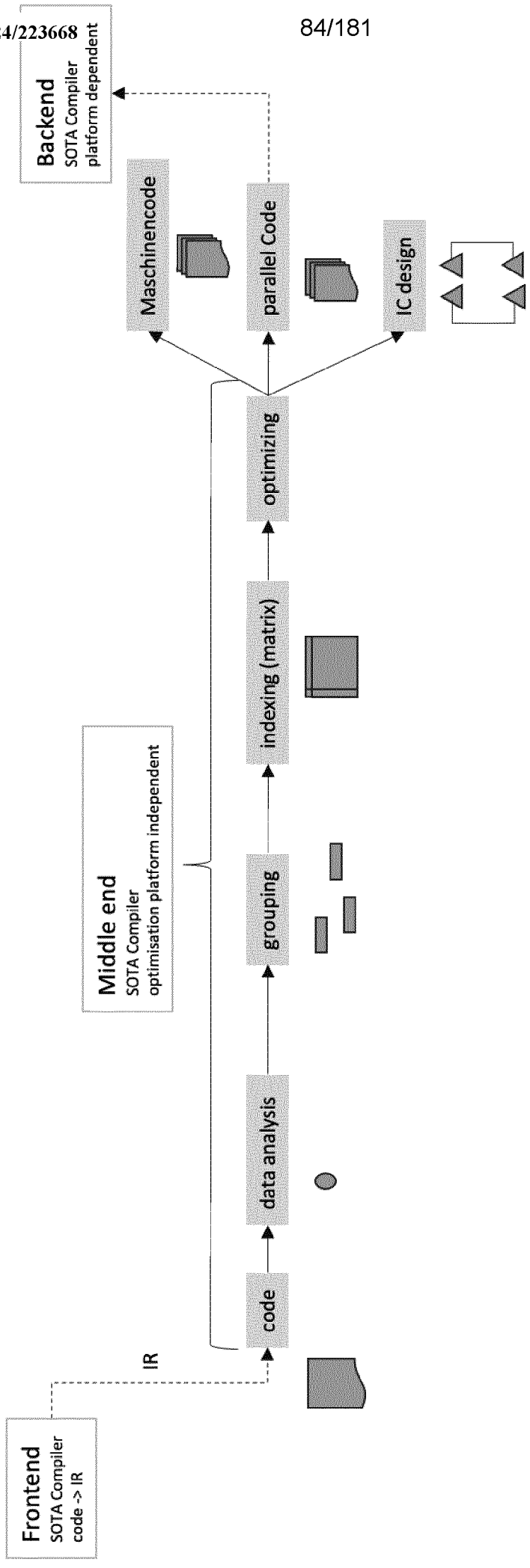

Backend
SOTA Compiler
platform dependent
Maschinencode
parallel Code
IC design
optimizing
indexing (matrix)
grouping
data analysis
code
Middle end
SOTA Compiler
optimisation platform independent
IR
Frontend
SOTA Compiler
code -> IR

Fig. 84

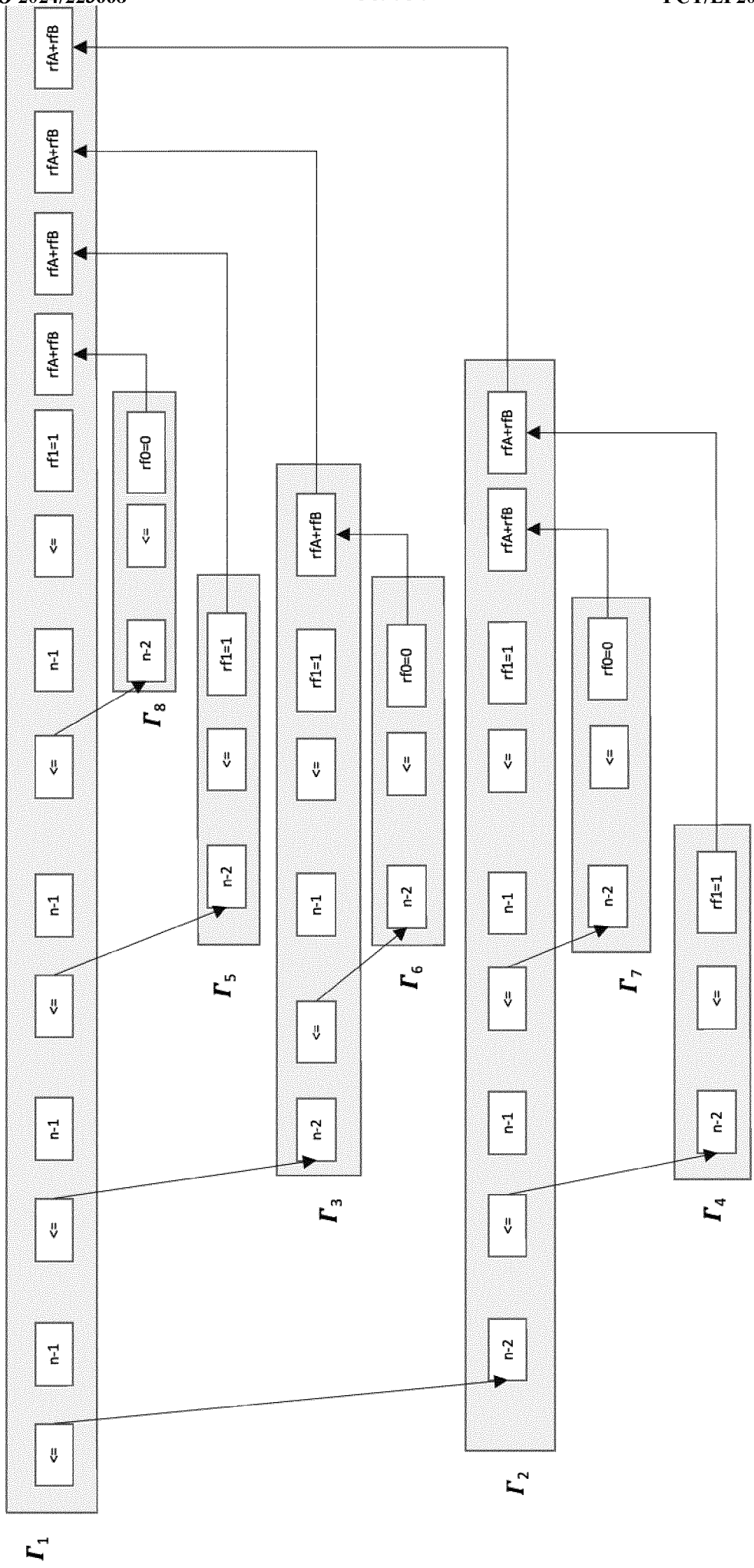
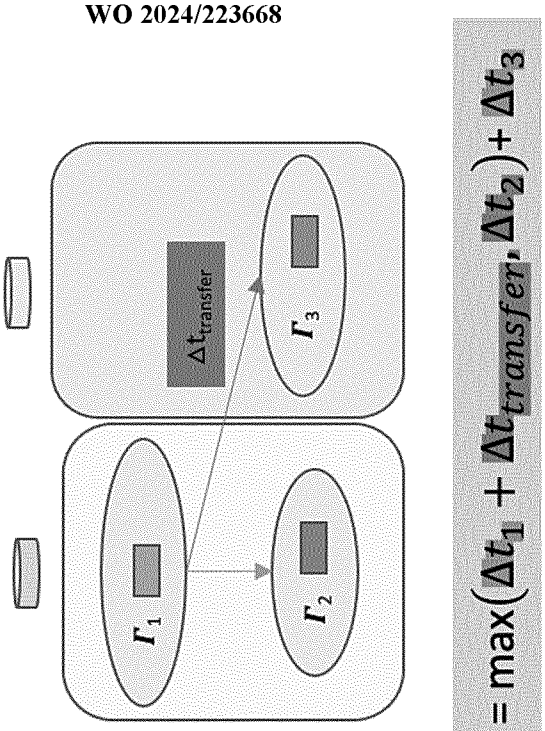
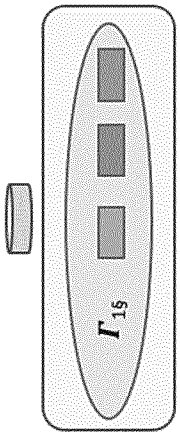


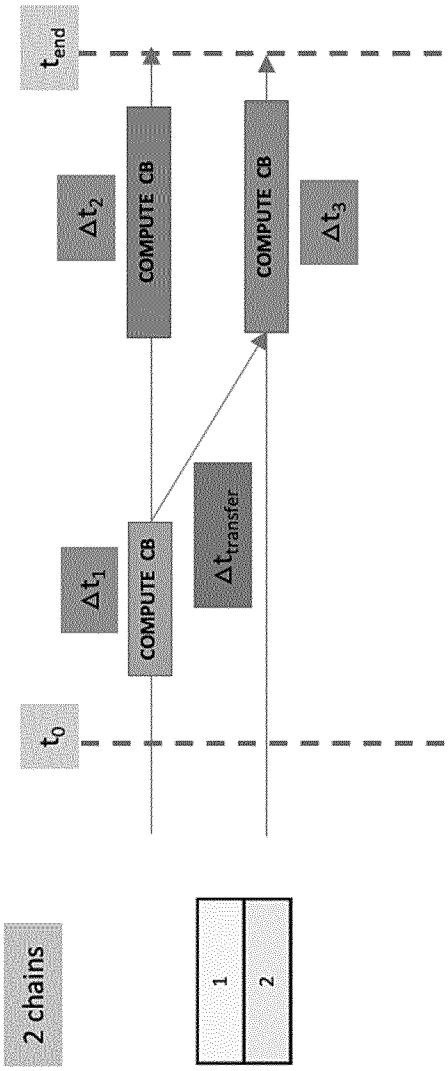
Fig. 85



$$t_{end1} = \max(\Delta t_1 + \Delta t_{transfer}, \Delta t_2) + \Delta t_3$$



$$t_{end2} = \Delta t_1$$



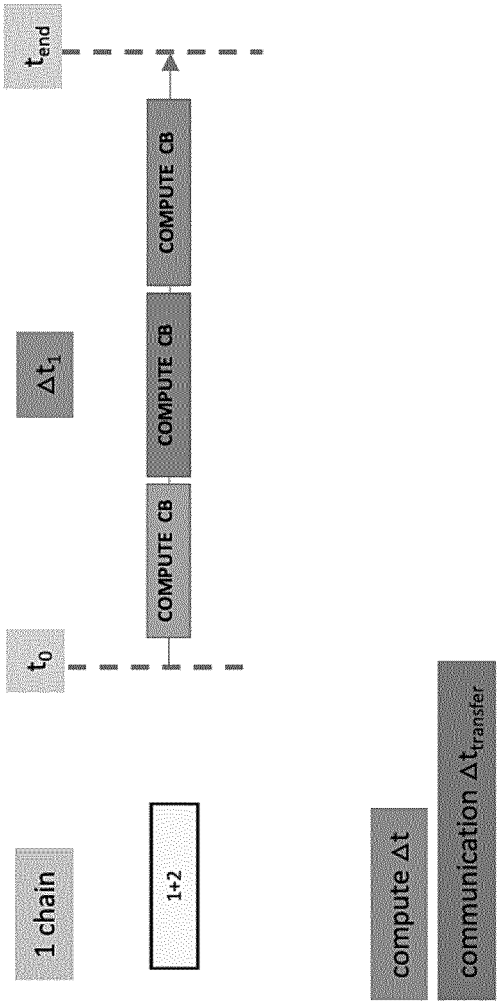


Fig. 86

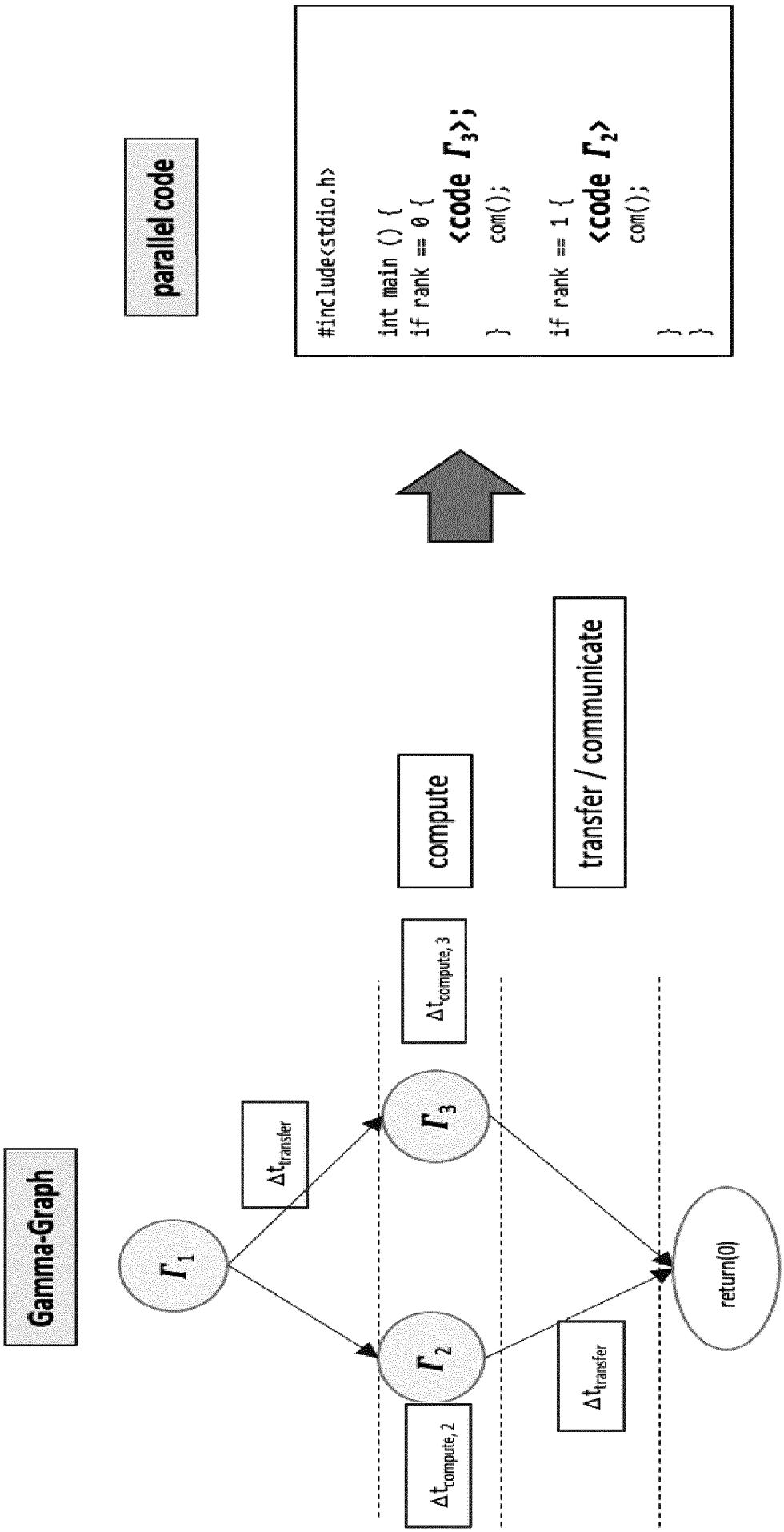

Gamma-Graph
Γ1
Δttransfer
Γ3
Γ2
Δtcompute, 2
Δtcompute, 3
compute
Δttransfer
transfer / communicate
return(0)
parallel code
#include<stdio.h>
int main () {
if rank == 0 {
<code Γ3>;
com();
}
if rank == 1 {
<code Γ2>
com();
}
}

Fig. 87

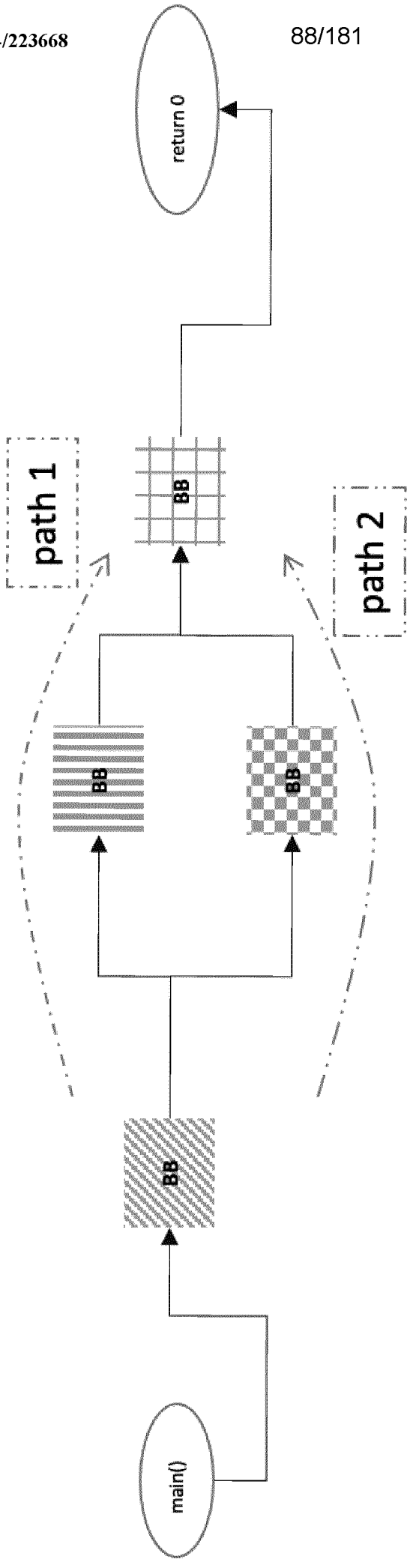


Fig. 88

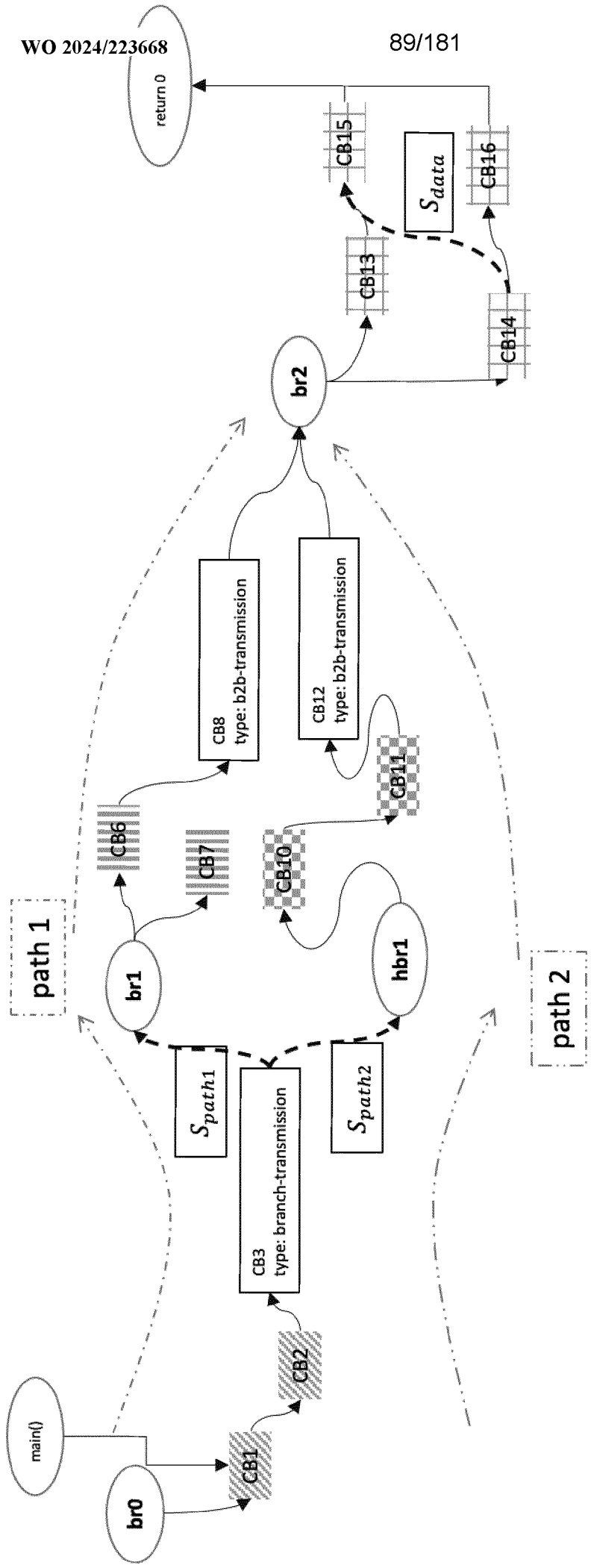

main()
br0
CB1
CB2
CB3
type: branch-transmission
S_{path1}
S_{path2}
br1
hbr1
CB6
CB7
CB8
type: b2b-transmission
CB10
CB11
CB12
type: b2b-transmission
br2
CB13
S_{data}
CB14
CB15
CB16
return 0
path 1
path 2

Fig. 89

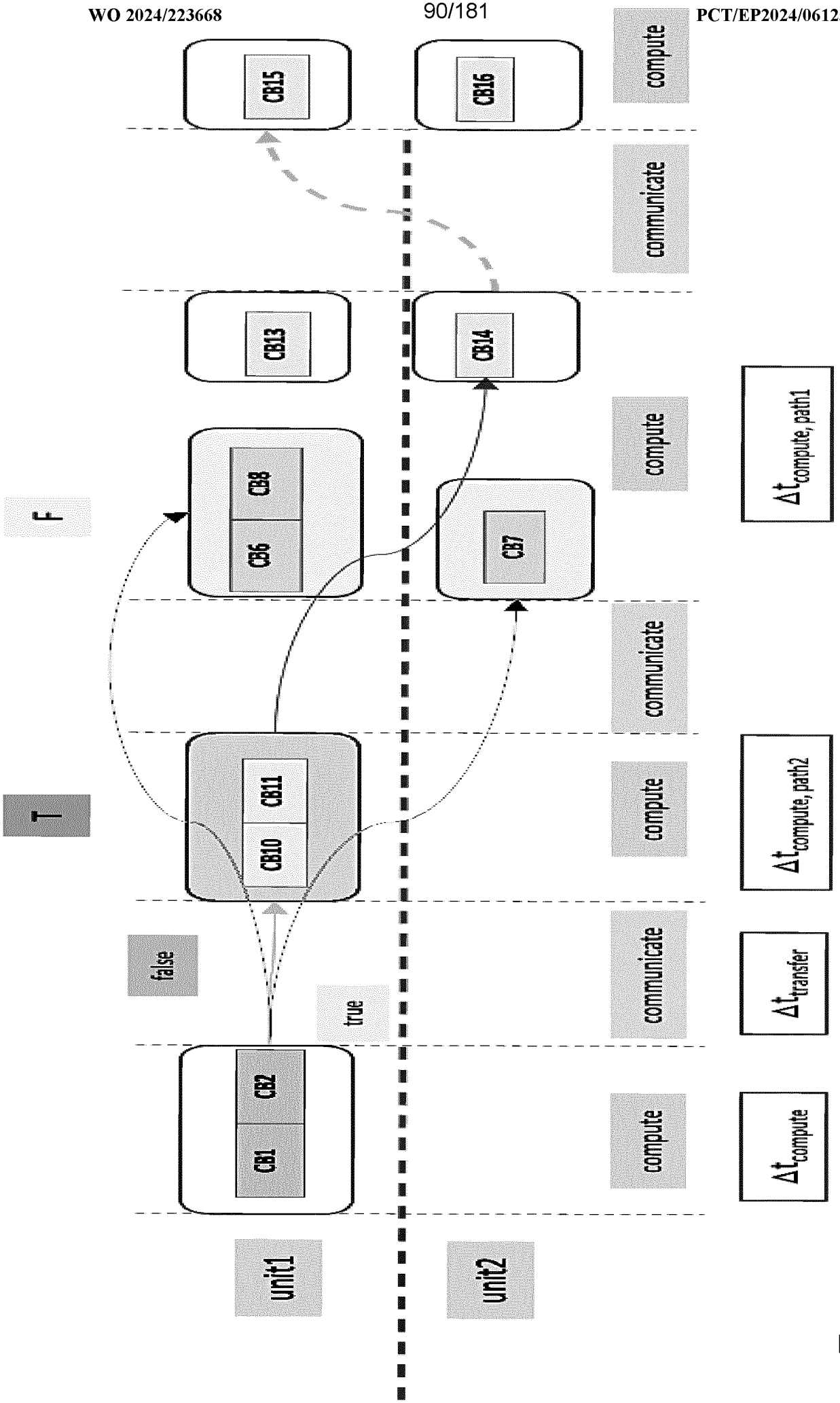

unit1
unit2
CB1
CB2
CB10
CB11
CB6
CB8
CB7
CB13
CB14
CB15
CB16
false
true
T
F
compute
communicate
compute
communicate
compute
communicate
compute
$\Delta t_{compute}$
$\Delta t_{transfer}$
$\Delta t_{compute, path2}$
$\Delta t_{compute, path1}$

Fig. 90

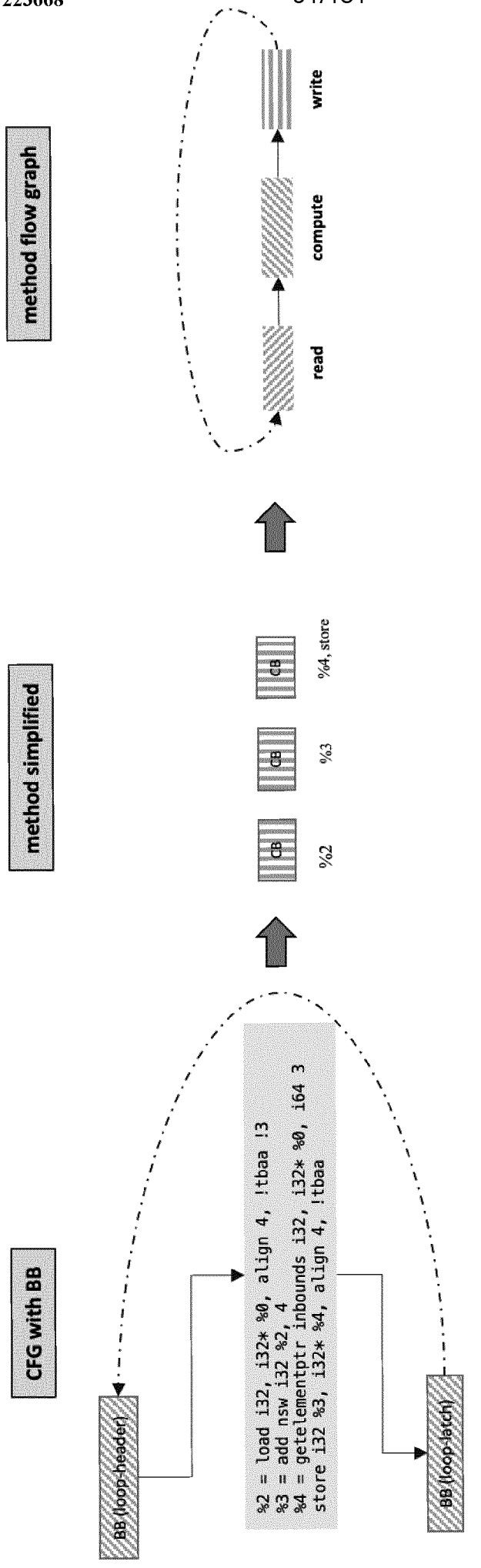

method flow graph
read
compute
write
method simplified
%2
%3
%4, store
CB
CB
CB
CFG with BB
BB (loop header)
%2 = load i32, i32* %0, align 4, !tbaa !3
%3 = add nsw i32 %2, 4
%4 = getelementptr inbounds i32, i32* %0, i64 3
store i32 %3, i32* %4, align 4, !tbaa
BB (loop latch)

Fig. 91

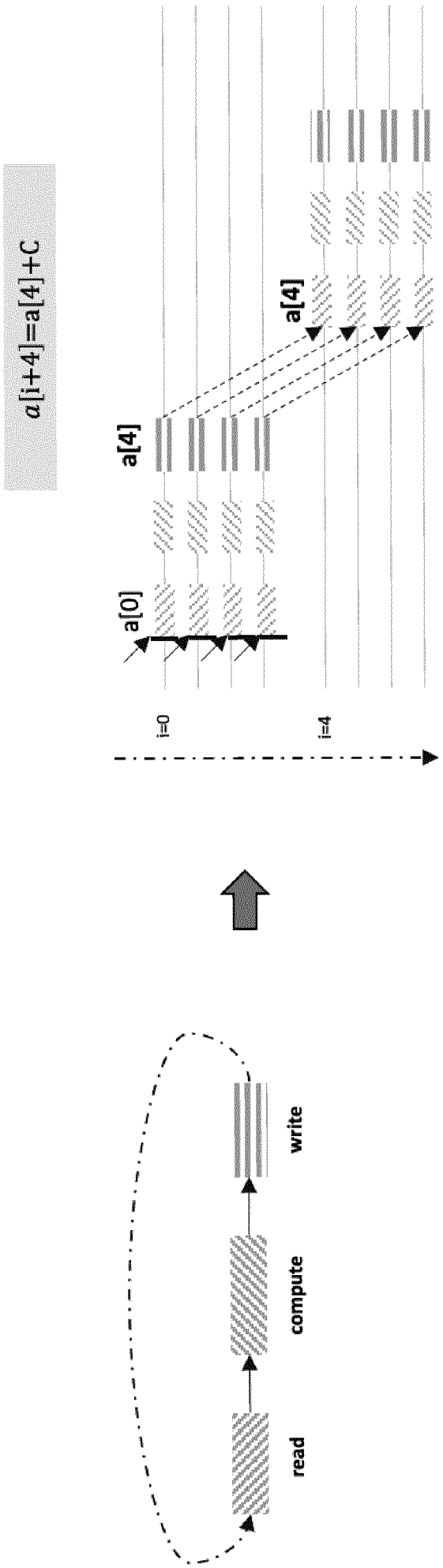


Fig. 92

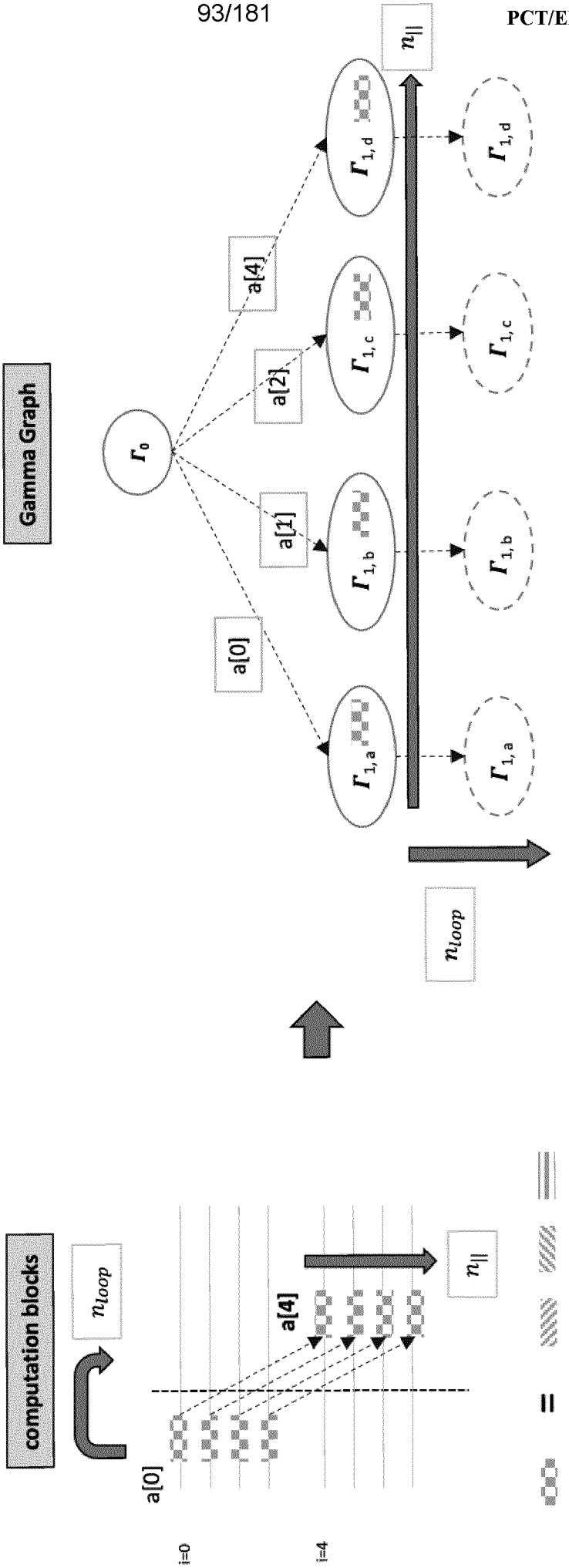

Gamma Graph
Γ_0
a[0]
a[1]
a[2]
a[4]
$\Gamma_{1,a}$
$\Gamma_{1,b}$
$\Gamma_{1,c}$
$\Gamma_{1,d}$
$n_{||}$
n_{loop}
computation blocks
n_{loop}
a[0]
a[4]
$n_{||}$
i=0
i=4
II

Fig. 93

#include<stdio.h>
int main ()
{
int n = 8, i;
int a[n]=(0);
for (i=0; i<=n-3; i++) {
a[i+3]=a[i]+4;
}
return 0;
}
A - resolved
B - model
B1: parameter extraction
a[i+3] = a[i]+4;
a[i+Δi] = a[i]
ΔIwr =Δi = 3
nloop = 5
B2: read-write model for transfer and compute as a function of ΔIwr and nloop
transfer
read@i
compute (ADD, 4)
write@i+Δi
ΔIwr=3 -> 3 Tasks
each with compute +4 -> ADD
nloop / ΔIwr instructions per task
B3: form computation graph
A1: list of every instruction
....
a[3] = a[0]+4
a[4] = a[1]+4
a[5] = a[2]+4
a[6] = a[3]+4
a[7] = a[4]+4
a[8] = a[5]+4
A2: apply rules for every instruction
A3: retrieve computation graph
Γ1
Γ2
Γ3
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[8]={}
+4
ΔIwr=3

Fig. 94

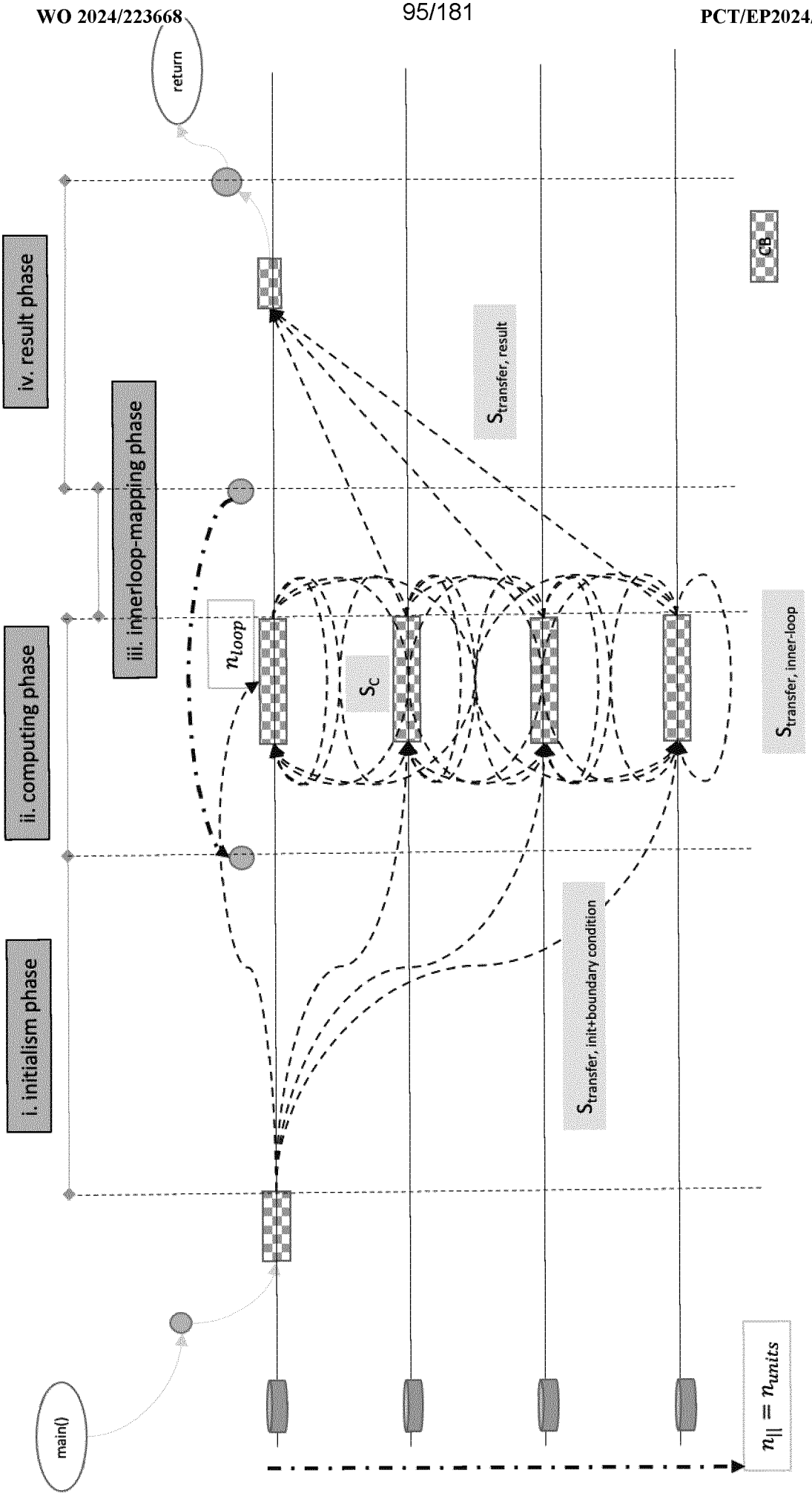

return
iv. result phase
iii. innerloop-mapping phase
ii. computing phase
i. initialism phase
main()
CB
$S_{transfer,\ result}$
$S_{transfer,\ inner\text{-}loop}$
$S_{transfer,\ init+boundary\ condition}$
n_{loop}
S_C
$n_{||} = n_{units}$

Fig. 95

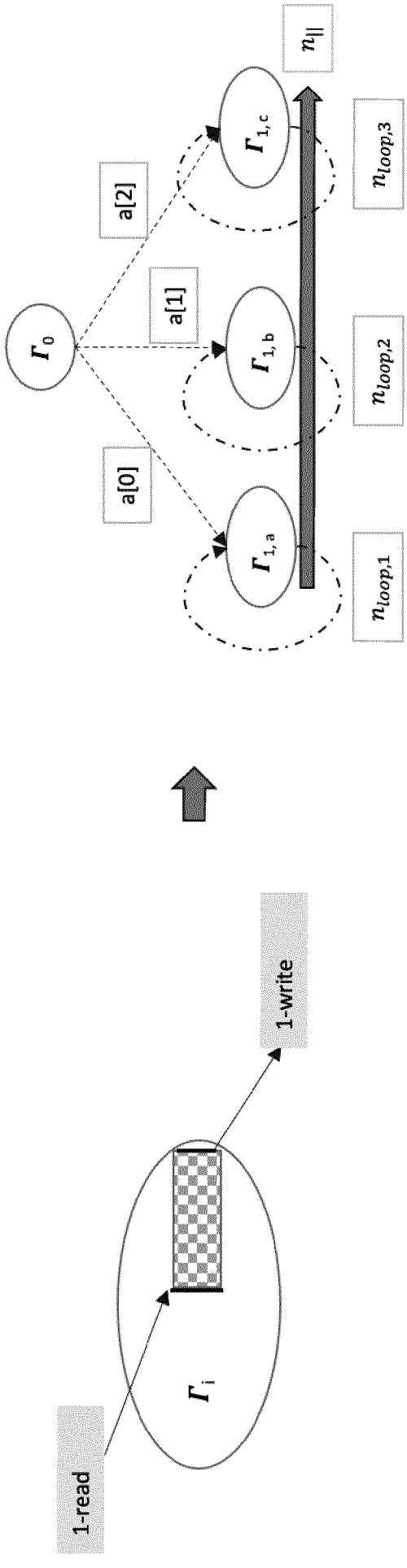

$n_{||}$
$\Gamma_{1,c}$
$n_{loop,3}$
a[2]
a[1]
Γ_0
$\Gamma_{1,b}$
$n_{loop,2}$
a[0]
$\Gamma_{1,a}$
$n_{loop,1}$
1-write
1-read
Γ_i

Fig. 96

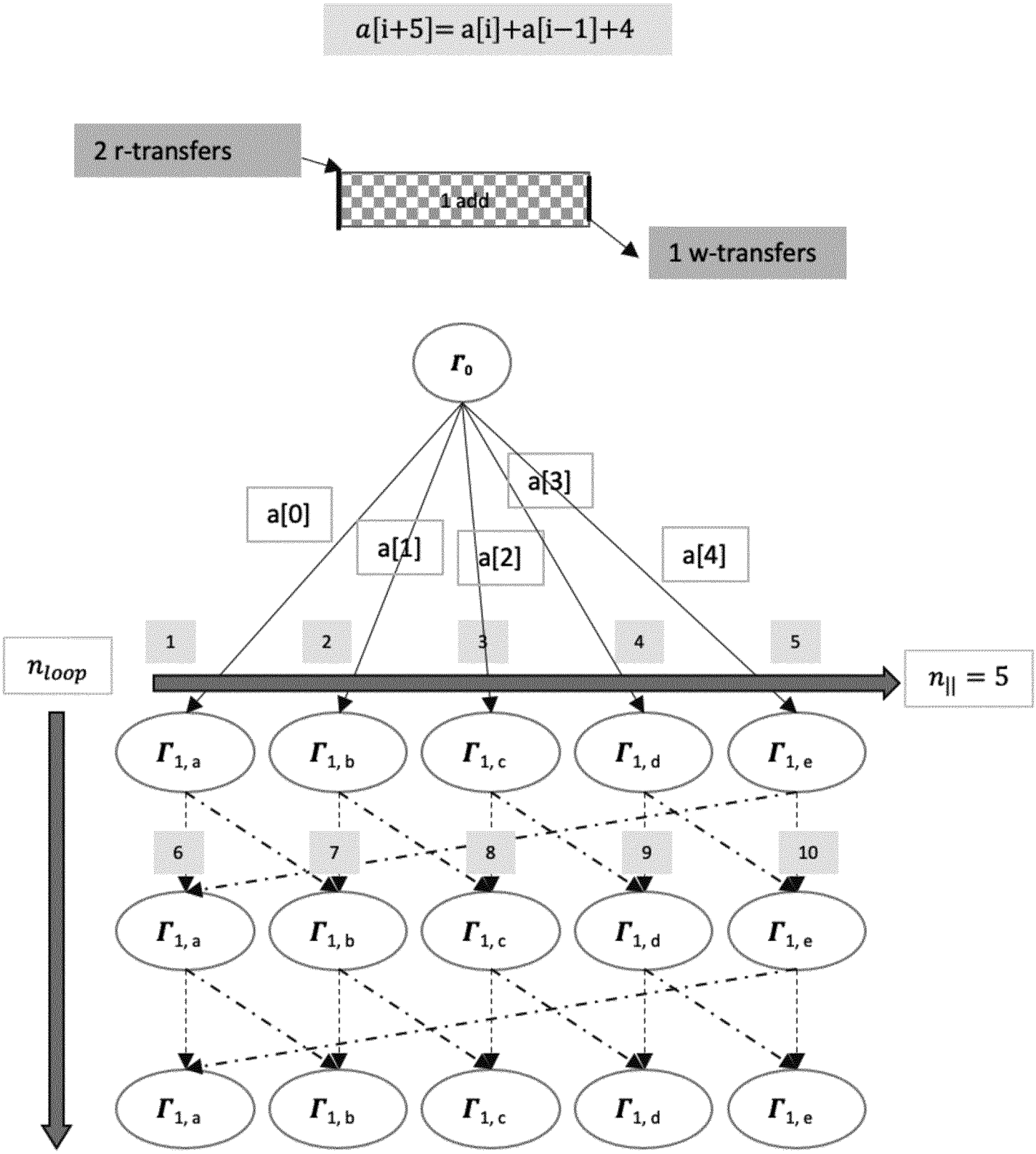

a[i+5]= a[i]+a[i−1]+4
2 r-transfers
1 add
1 w-transfers
r_0
a[0]
a[1]
a[2]
a[3]
a[4]
1
2
3
4
5
n_{loop}
$n_{||} = 5$
$r_{1,a}$
$r_{1,b}$
$r_{1,c}$
$r_{1,d}$
$r_{1,e}$
6
7
8
9
10

Fig. 97

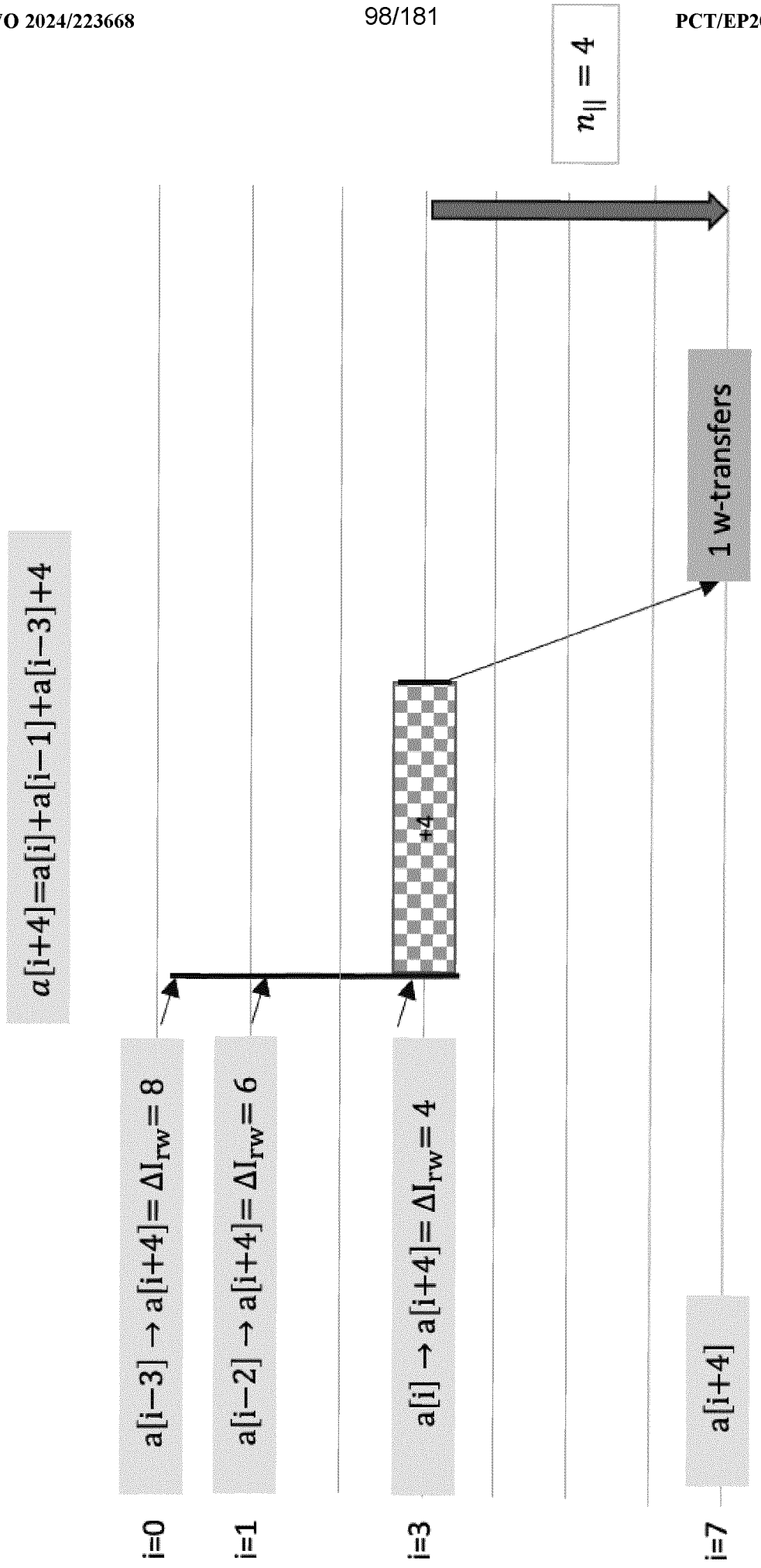
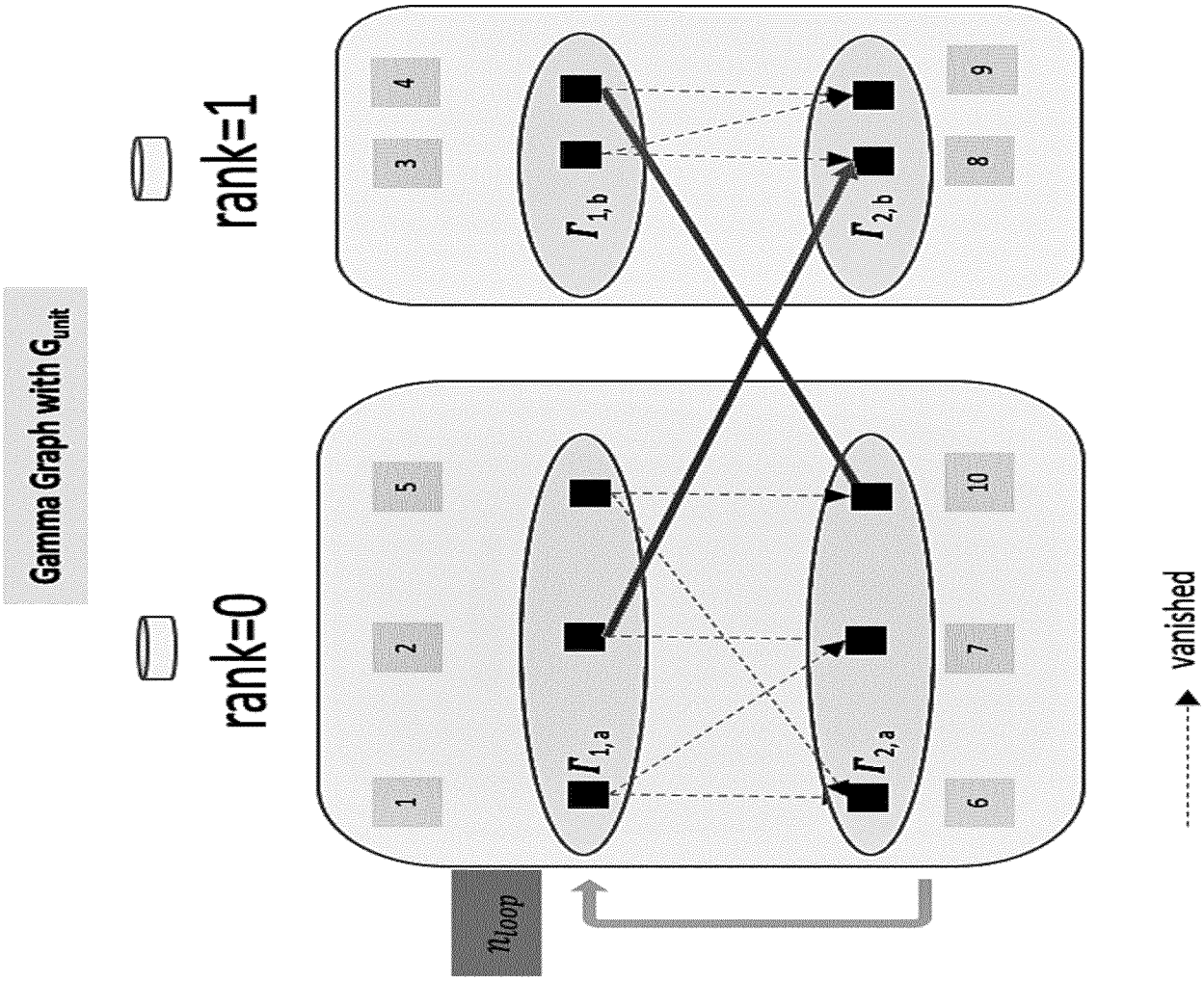

$n_{||} = 4$
$a[i+4]=a[i]+a[i-1]+a[i-3]+4$
1 w-transfers
+4
a[i-3] → a[i+4]= ΔI_rw= 8
a[i-2] → a[i+4]= ΔI_rw= 6
a[i] → a[i+4]= ΔI_rw= 4
a[i+4]
i=0
i=1
i=3
i=7

Fig. 98


Gamma Graph with G_{unit}
rank=1
rank=0
$r_{1,b}$
$r_{2,b}$
$r_{1,a}$
$r_{2,a}$
n_{loop}
vanished

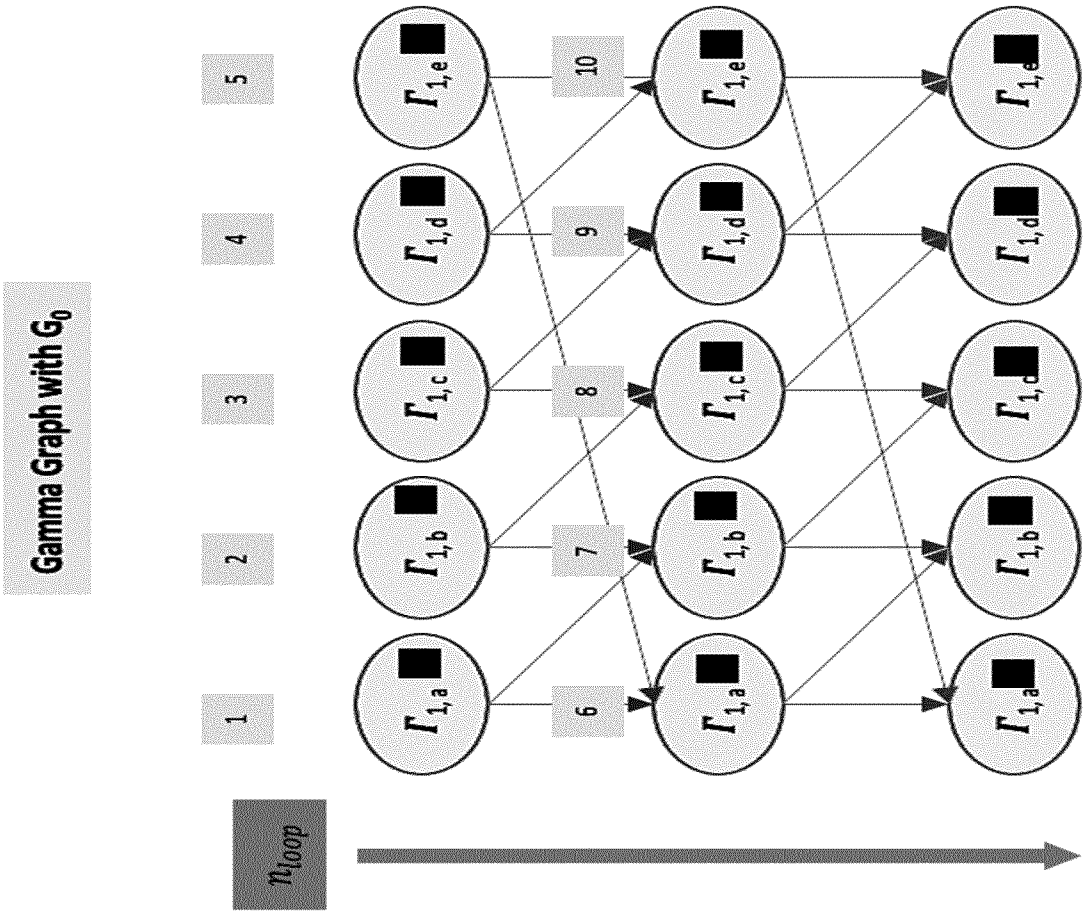

Gamma Graph with G_0
$r_{1,a}$
$r_{1,b}$
$r_{1,c}$
$r_{1,d}$
$r_{1,e}$
n_{loop}

Fig. 99

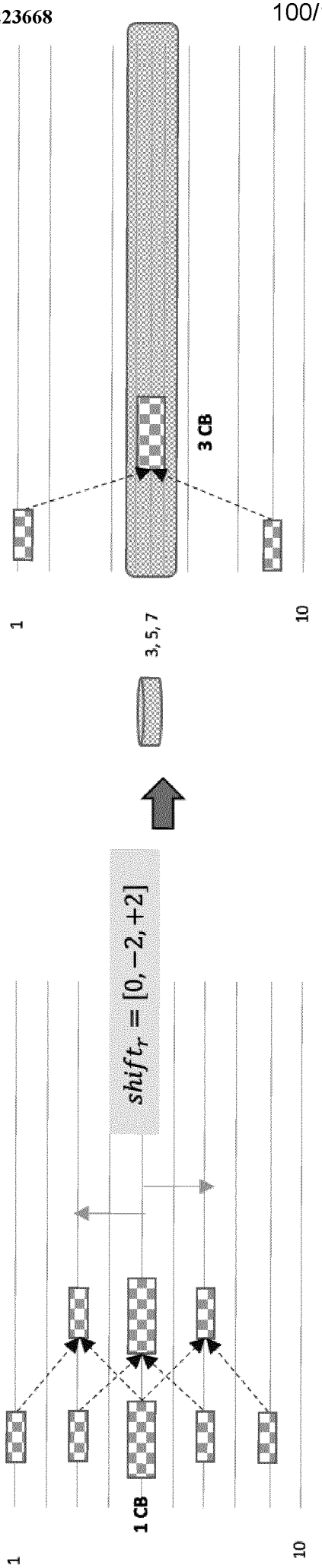


Fig. 100

a[i][k+1]=a[i+5][k]+a[i−2][k]
boundary A
boundary B
2 lines
$n_{comb,const}$
5 lines
$shift_{r1} = -2$
$shift_{r2} = +5$
1
10
$S_{T,const} = 7T$
$S_{T,1CB} = 2T$
S_T
n_{comb}
8
1
$n_{comb,const}$

Fig. 101

$a[i][k+1]=a[i+5][k]+a[i-2][k]$

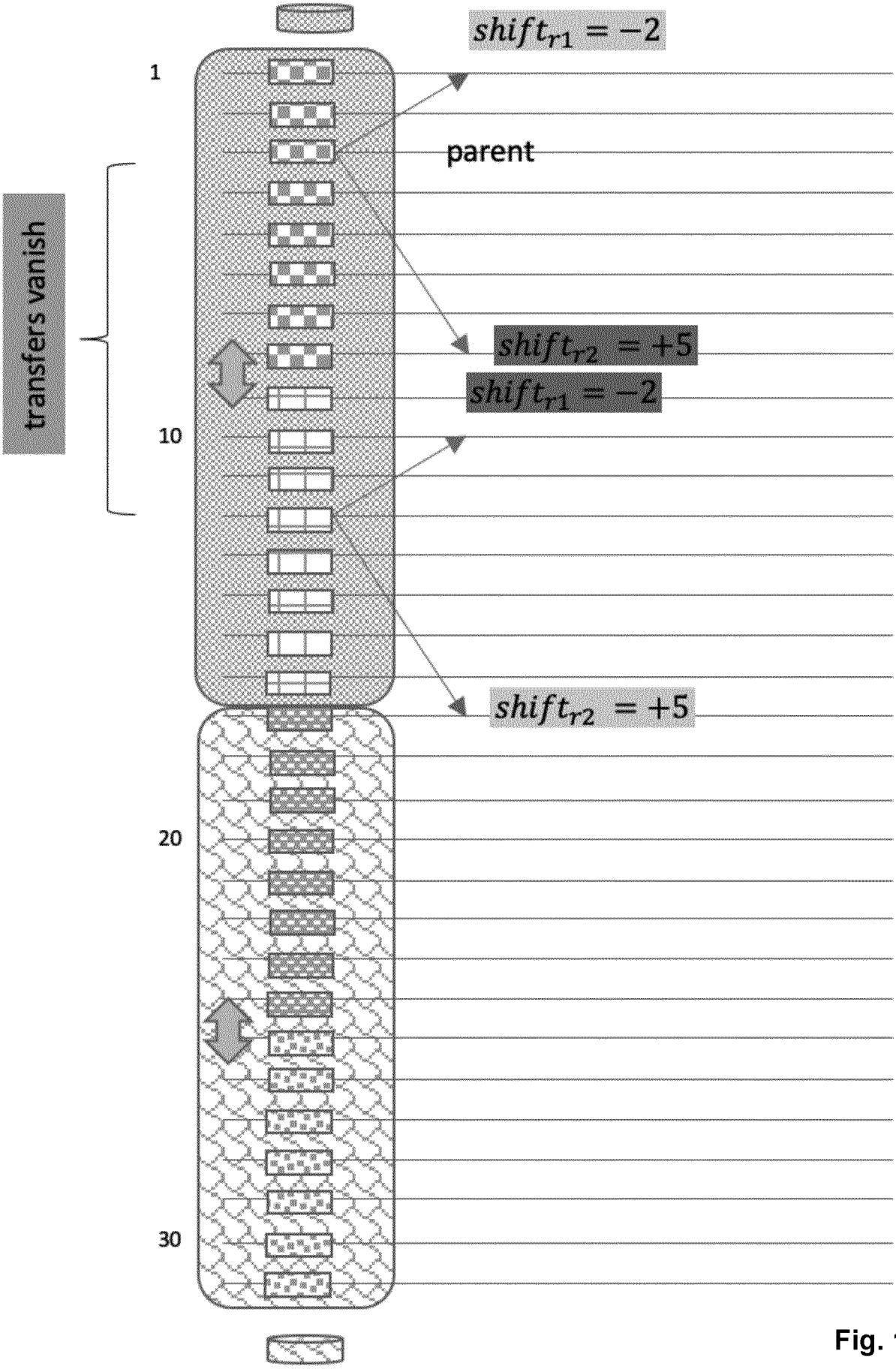


Fig. 102

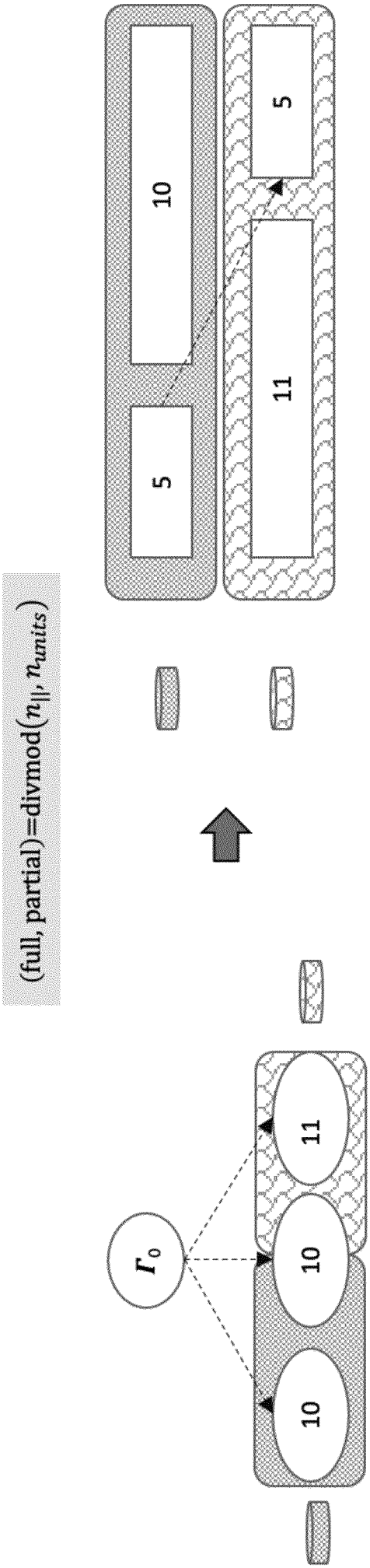


Fig. 103

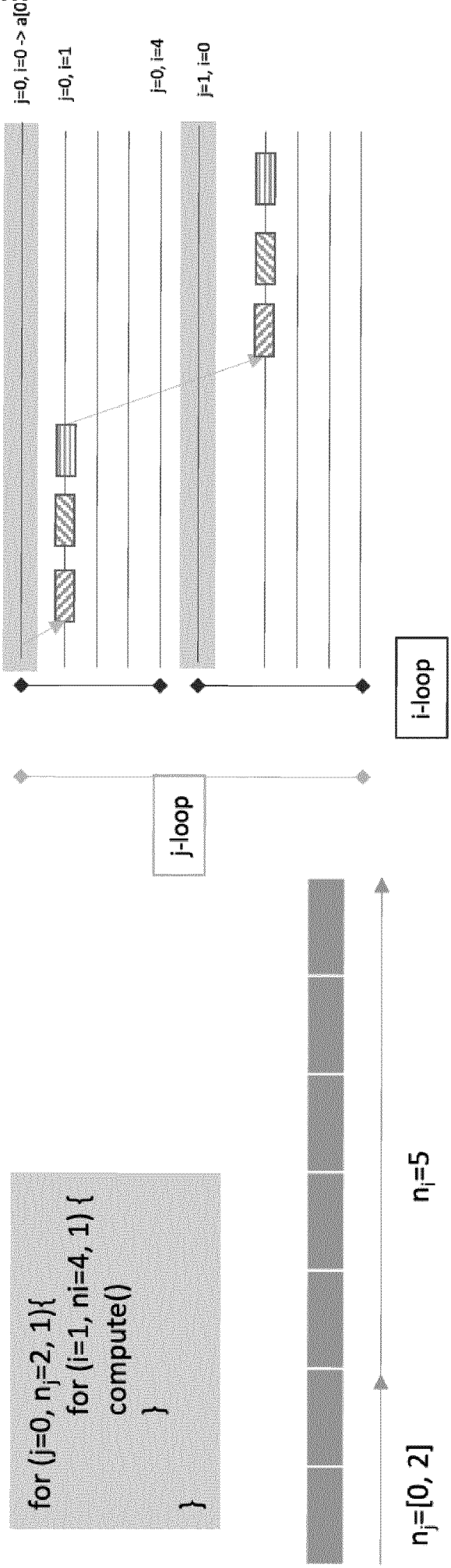

j=0, i=0 -> a[0]
j=0, i=1
j=0, i=4
j=1, i=0
i-loop
j-loop
for (j=0, nj=2, 1){
for (i=1, ni=4, 1) {
compute()
}
}
nj=5
nj=[0, 2]

Fig. 104

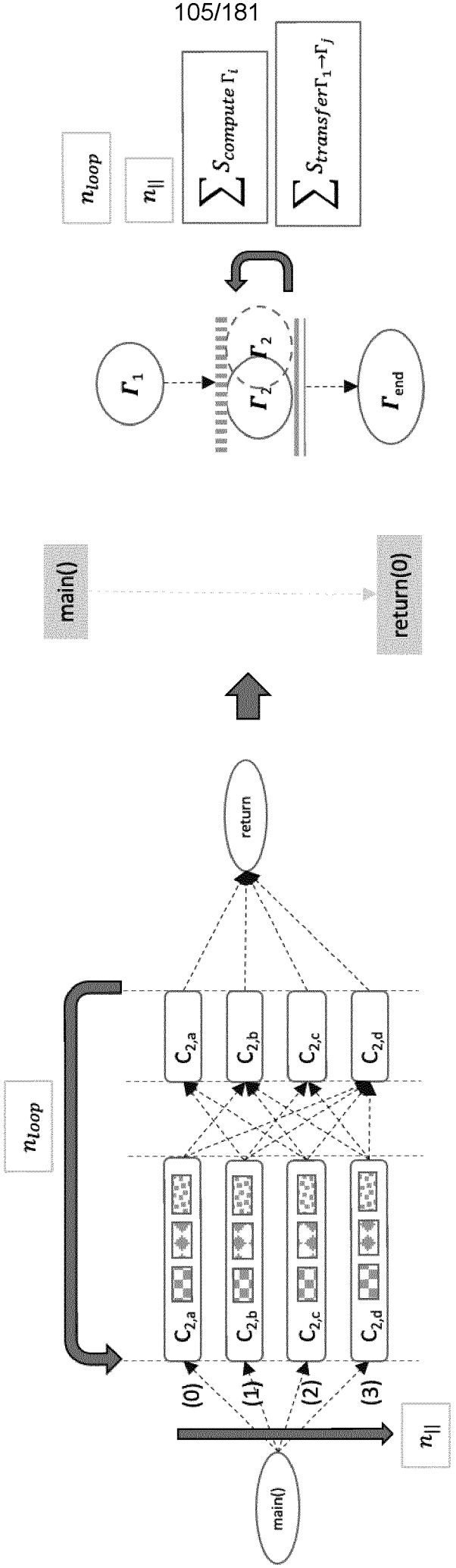

n_{loop}
$n_{||}$
$\sum S_{compute\,\Gamma_i}$
$\sum S_{transfer\,\Gamma_1 \to \Gamma_j}$
Γ_1
Γ_2
Γ_2'
Γ_{end}
main()
return(0)
return
$C_{2,a}$
$C_{2,b}$
$C_{2,c}$
$C_{2,d}$
(0)
(1)
(2)
(3)
main()

Fig. 105

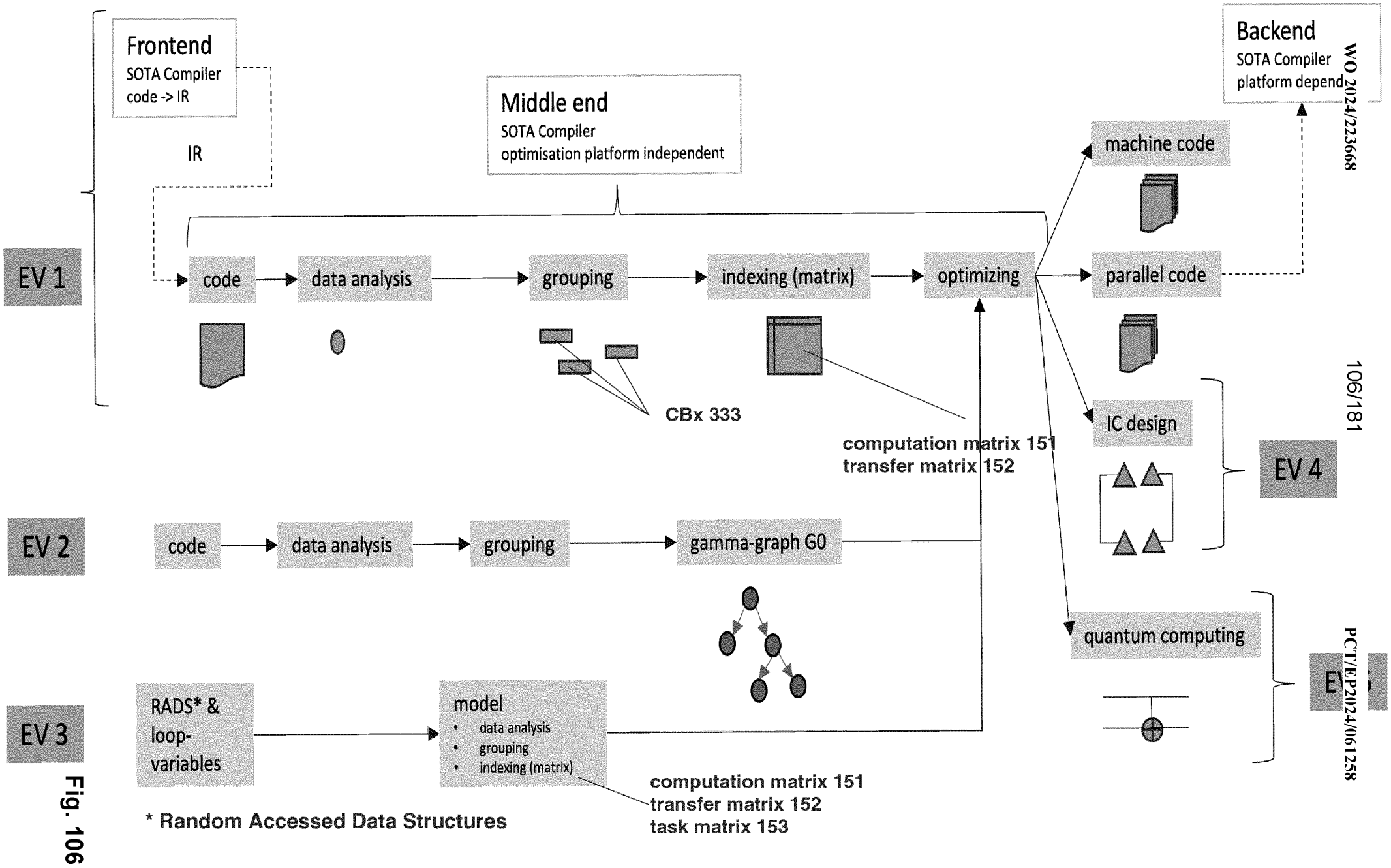

Frontend
SOTA Compiler
code -> IR
IR
Middle end
SOTA Compiler
optimisation platform independent
Backend
SOTA Compiler
platform depend
EV 1
code
data analysis
grouping
indexing (matrix)
optimizing
machine code
parallel code
IC design
CBx 333
computation matrix 151
transfer matrix 152
EV 4
EV 2
code
data analysis
grouping
gamma-graph G0
quantum computing
EV 3
RADS* & loop-variables
model
• data analysis
• grouping
• indexing (matrix)
computation matrix 151
transfer matrix 152
task matrix 153
* Random Accessed Data Structures

Fig. 106

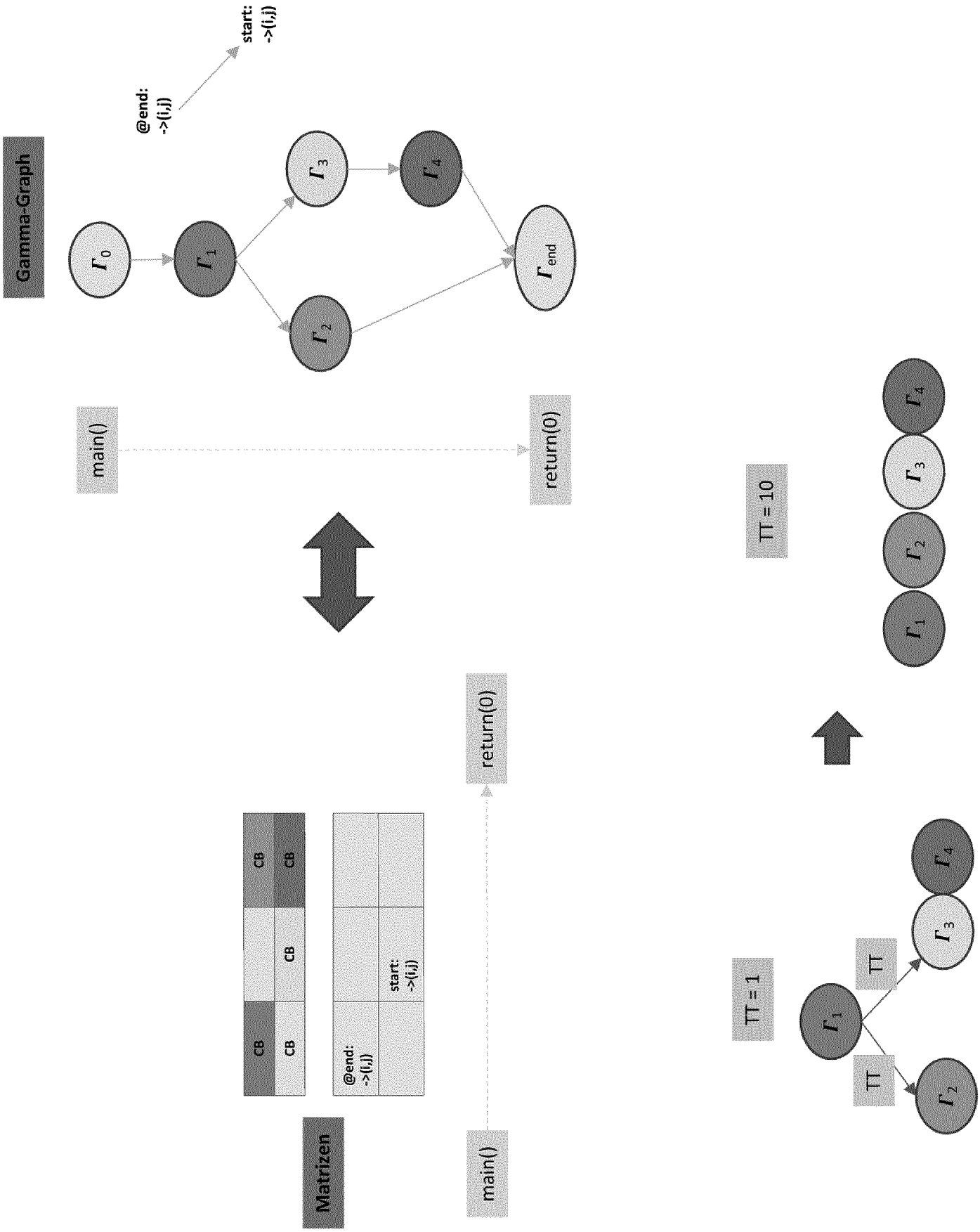

Gamma-Graph
@end:
->(i,j)
start:
->(i,j)
Γ0
Γ1
Γ2
Γ3
Γ4
Γend
main()
return(0)
CB
CB
CB
CB
@end:
->(i,j)
start:
->(i,j)
Matrizen
main()
return(0)
TT = 10
TT = 1
TT
TT

Fig. 107

#define nX 4
#define nY 5
#define nT 3
void f(int *a)
{
for (int k = 0; k < nT; ++k) {
for (int j = 1; j < nY-1; ++j) {
for (int i = 1; i < nX-1; ++i) {
u[i+nX*(j+nY*(k+1))] = (u[i+1+nX*(j+nY*(k))]+u[i-1+nX*(j+nY*(k))]+u[i+nX*(j+1+nY*(k))]+u[i+nX*(j-1+nY*(k))]-4*u[i+nX*(j+nY*(k))])+u[i+nX*(j+nY*(k))];
}
}
}
}
$n=f(\Delta I_{wr})$
unit 1
unit 2
r_0
r_1
r_2
r_{end}
n_{loop}
$\Delta I_{wr}=\Delta i=3$
$n_{loop}=5$
5 «reading»
5 add, 2 mul
1 «writing»
$t_f=(t_a+t_b+t_c+t_d-4*t_e)+t_e$

Fig. 108

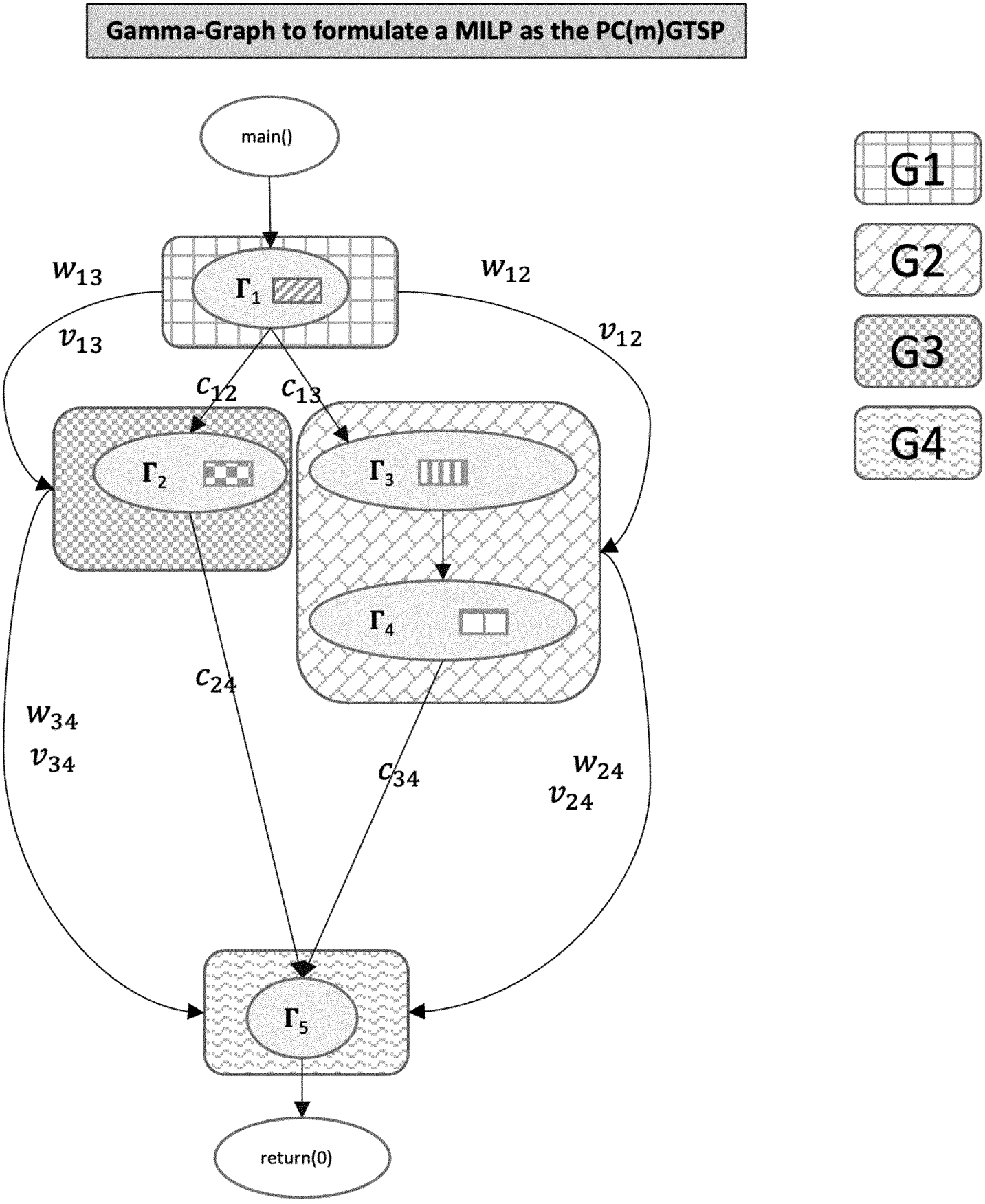

Gamma-Graph to formulate a MILP as the PC(m)GTSP
main()
Γ_1
w_{13}
v_{13}
w_{12}
v_{12}
c_{12}
c_{13}
Γ_2
Γ_3
Γ_4
c_{24}
w_{34}
v_{34}
c_{34}
w_{24}
v_{24}
Γ_5
return(0)
G1
G2
G3
G4

Fig. 109a

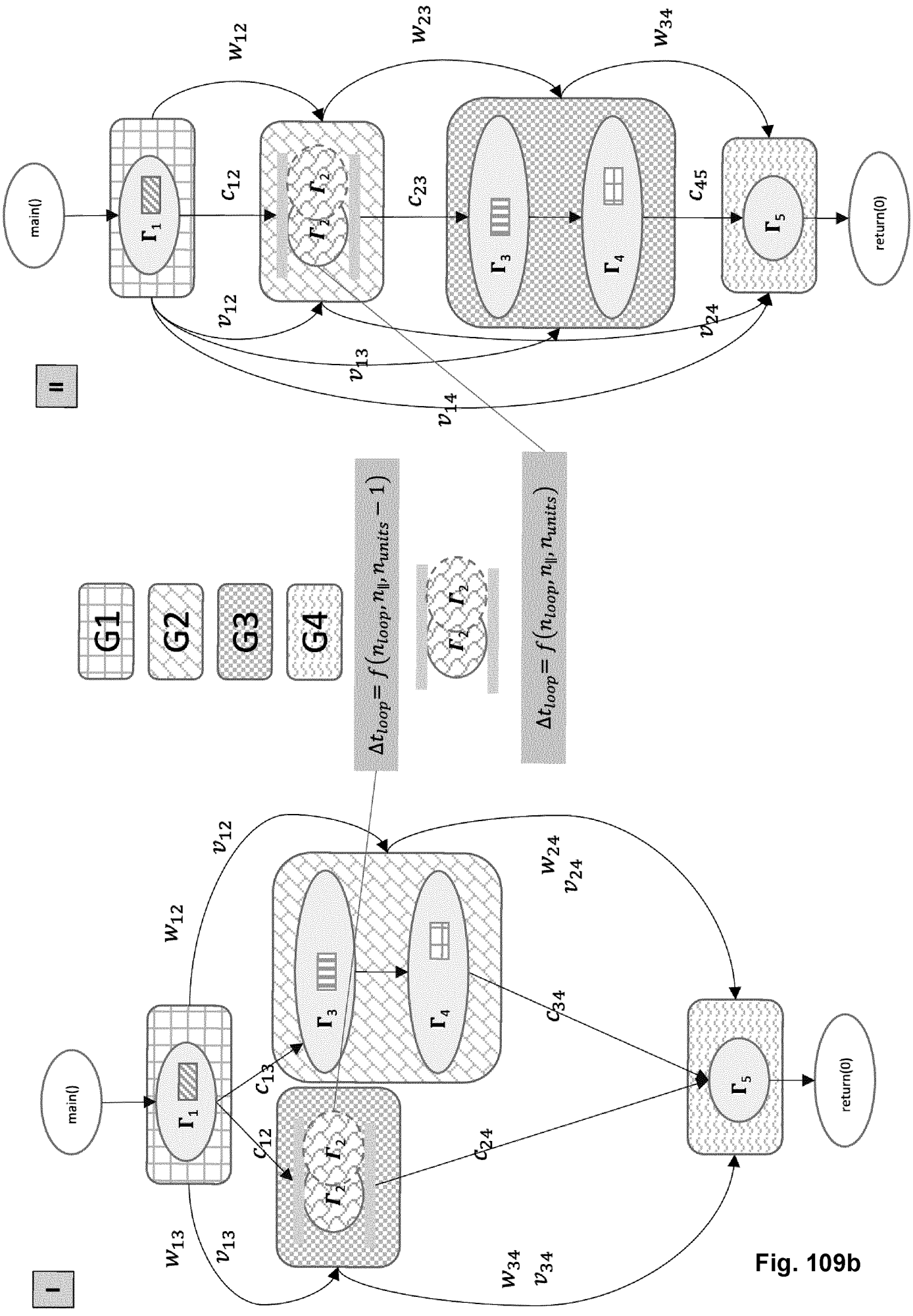

II
main()
Γ_1
Γ_2
Γ_2
Γ_3
Γ_4
Γ_5
return(0)
w_{12}
w_{23}
w_{34}
c_{12}
c_{23}
c_{45}
v_{12}
v_{13}
v_{14}
v_{24}
G1
G2
G3
G4
$\Delta t_{loop} = f(n_{loop}, n_{\|}, n_{units} - 1)$
$\Delta t_{loop} = f(n_{loop}, n_{\|}, n_{units})$
I
main()
Γ_1
Γ_2
Γ_2
Γ_3
Γ_4
Γ_5
return(0)
w_{12}
w_{13}
w_{24}
w_{34}
v_{12}
v_{13}
v_{24}
v_{34}
c_{12}
c_{13}
c_{24}
c_{34}

Fig. 109b

All transfer properties are known -> Size of variables

```
entity entity_name is
  Port declaration;
end entity_name;
```

An entity declaration should start with 'entity' and end with 'end' keywords. The direction will be input, output, inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source.

Each gamma node comprises to a step in the program all serial instructions

Logic Operation – AND GATE

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Symbol:


```
VHDL Code:
Library ieee;
use ieee.std_logic_1164.all;
entity and1 is
    port(x,y:in bit; z:out bit);
end and1;
architecture virat of and1 is
begin
    z<= and y;
end virat;
```


Gamma-Graph

main()

```
#include<stdio.h>
int main ()
{
int n = 8, i;
int a[n]={0};
for (i=0; i<=n-3; i++) {
        a[i+3]=a[i]+4;
}
return 0;
}
```

return(0)

Fig. 110

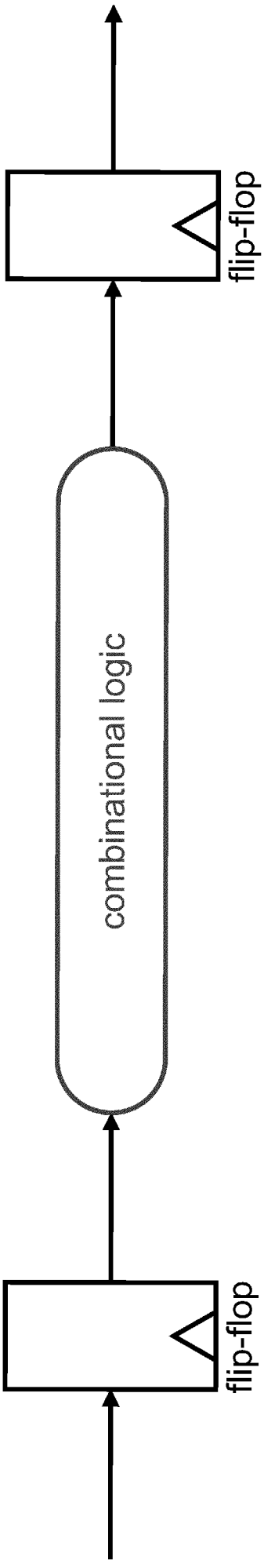


Fig. 111a

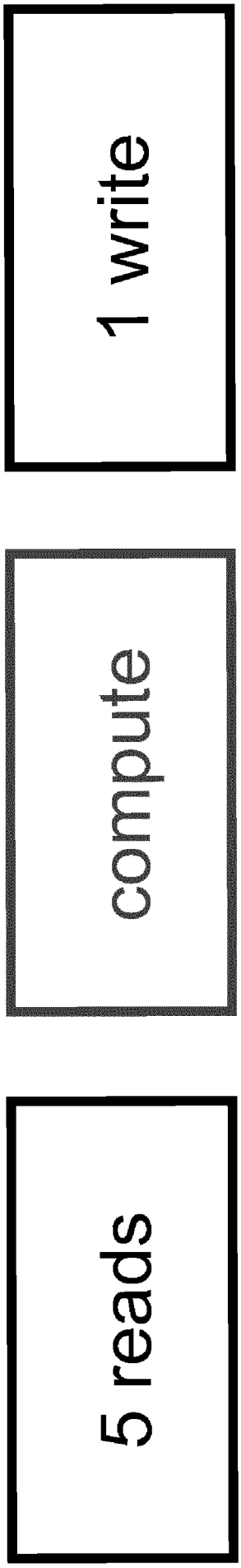


Fig. 111b

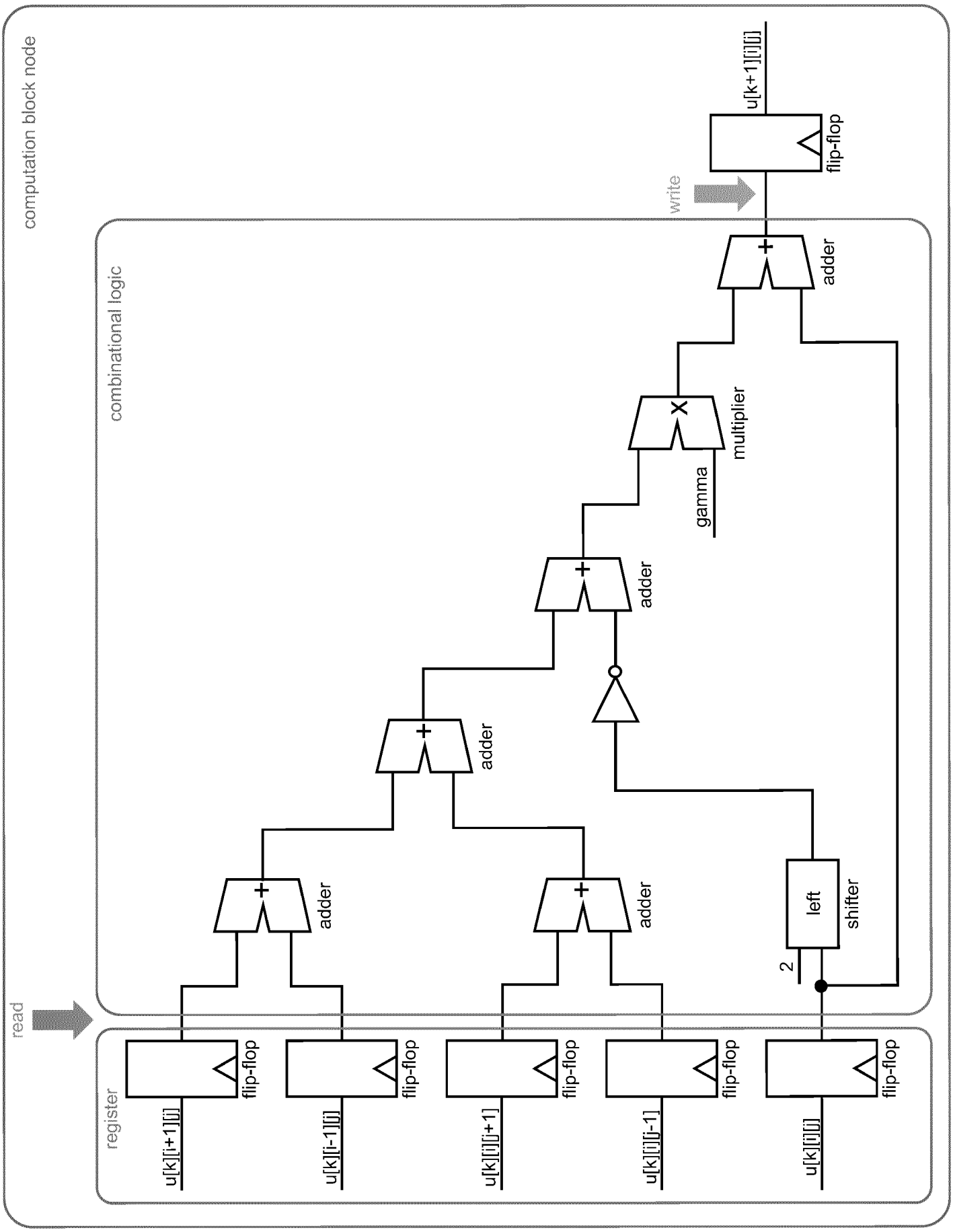

computation block node
combinational logic
register
read
write
u[k][i+1][j]
u[k][i-1][j]
u[k][i][j+1]
u[k][i][j-1]
u[k][i][j]
flip-flop
adder
multiplier
gamma
left
shifter
2
u[k+1][i][j]

Fig. 111c

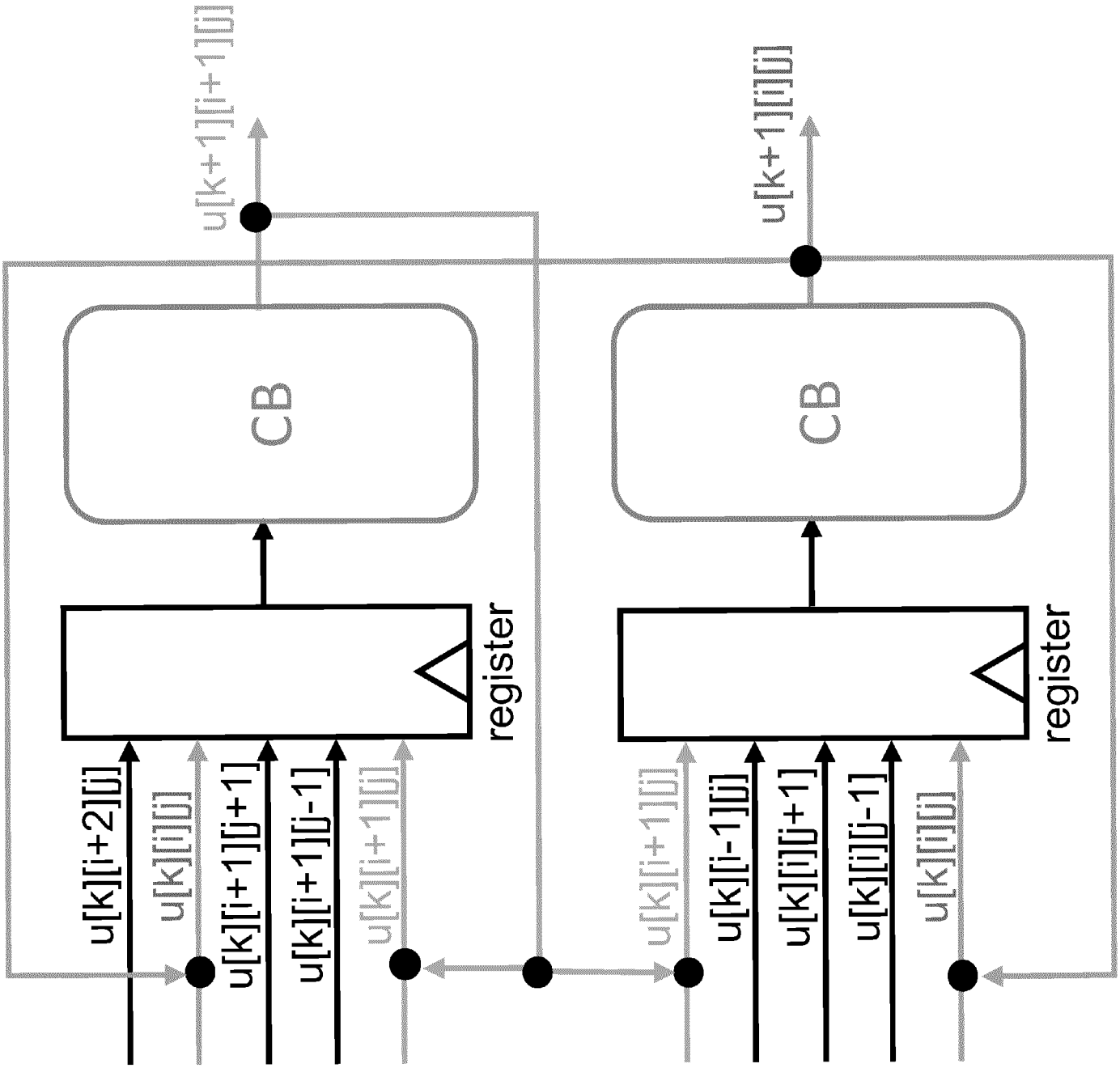

u[k+1][i+1][j]
u[k+1][i][j]
CB
CB
register
register
u[k][i+2][j]
u[k][i][j]
u[k][i+1][j+1]
u[k][i+1][j-1]
u[k][i+1][j]
u[k][i-1][j]
u[k][i][j+1]
u[k][i][j-1]
u[k][i][j]

Fig. 111d

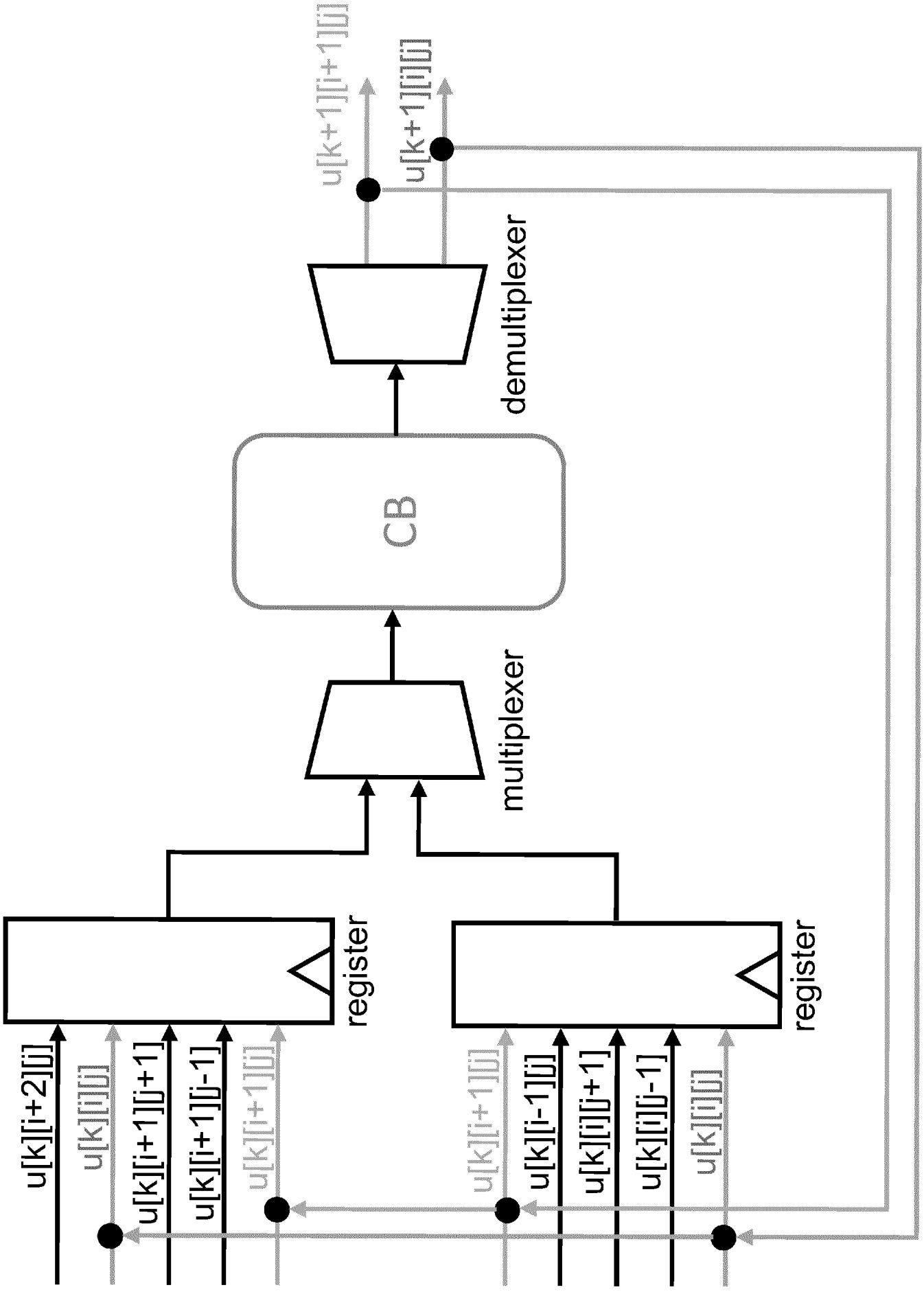

u[k+1][i+1][j]
u[k+1][i][j]
demultiplexer
CB
multiplexer
register
register
u[k][i+2][j]
u[k][i][j]
u[k][i+1][j+1]
u[k][i+1][j-1]
u[k][i+1][j]
u[k][i+1][j]
u[k][i-1][j]
u[k][i][j+1]
u[k][i][j-1]
u[k][i][j]

Fig. 111e

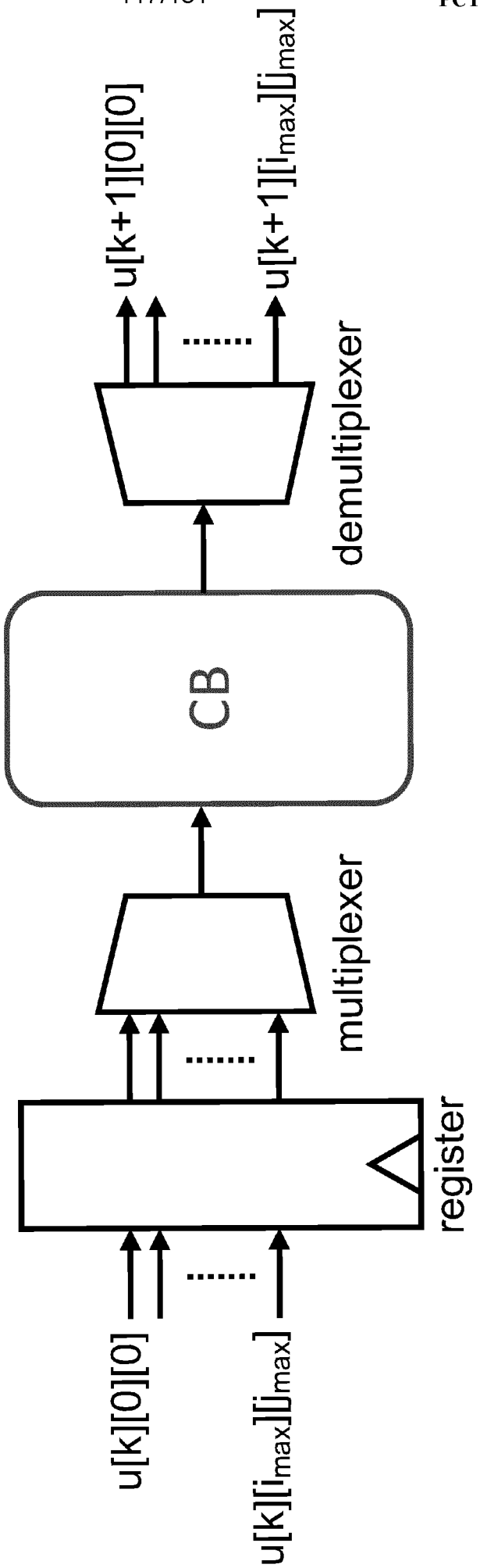

u[k][0][0]
u[k][i_max][j_max]
register
multiplexer
CB
demultiplexer
u[k+1][0][0]
u[k+1][i_max][j_max]

Fig. 111f

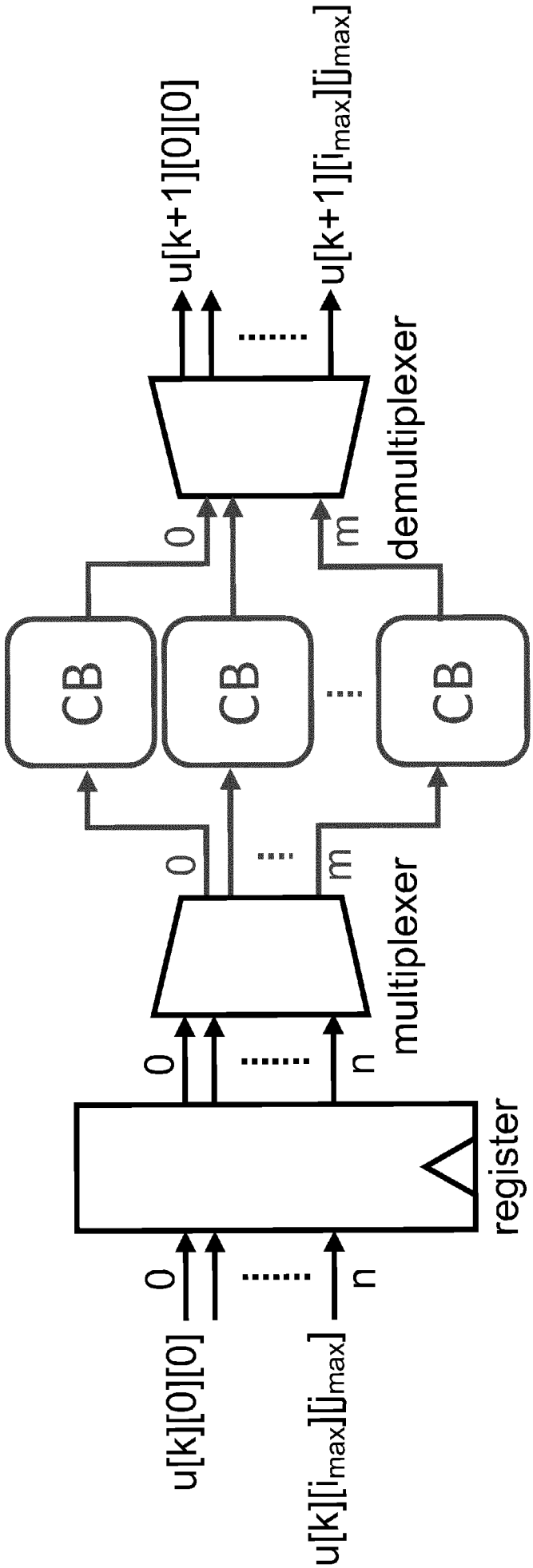

u[k][0][0]
u[k][i_max][j_max]
0
n
register
multiplexer
demultiplexer
CB
CB
CB
u[k+1][0][0]
u[k+1][i_max][j_max]

Fig. 111g

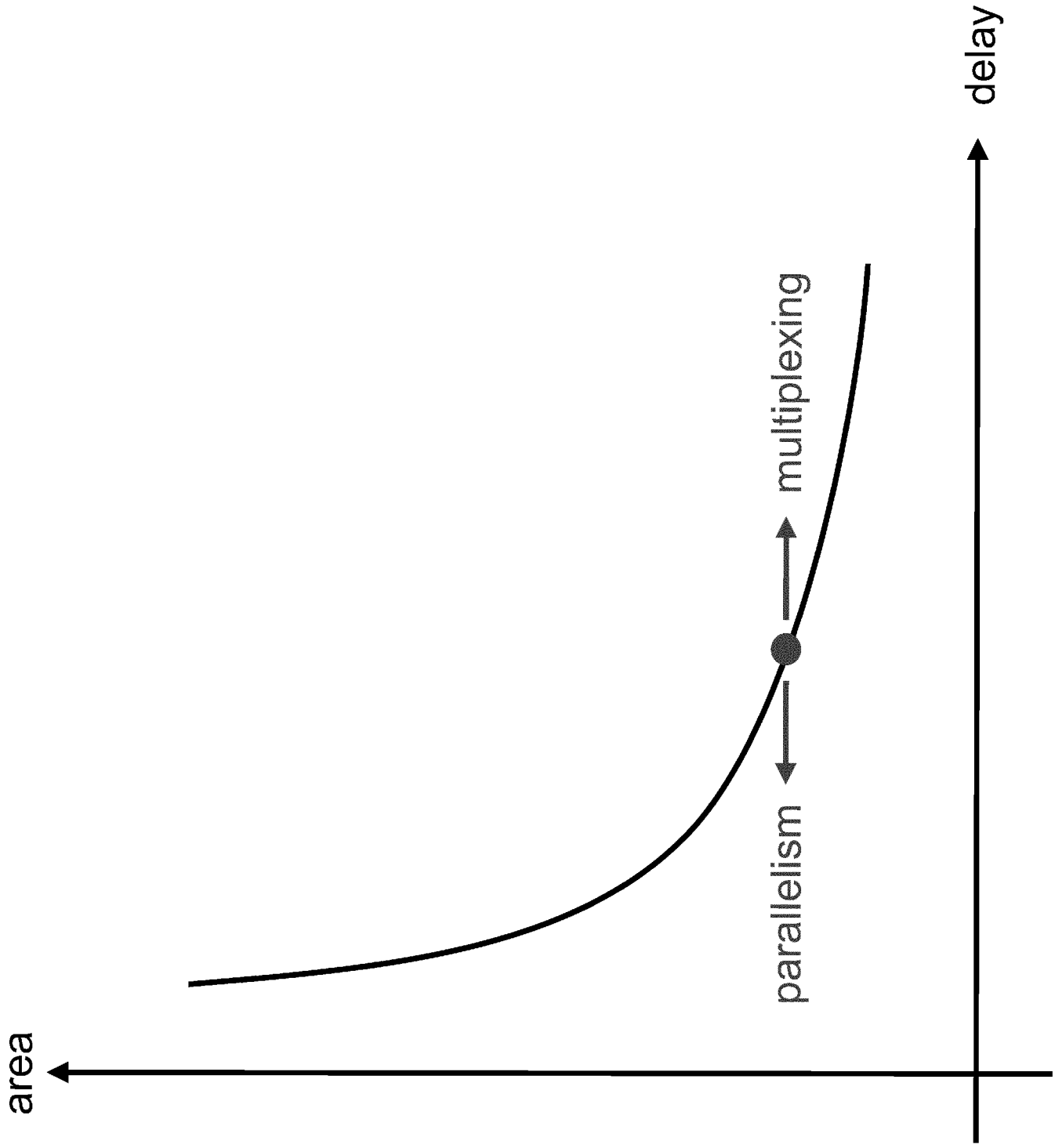


Fig. 111h

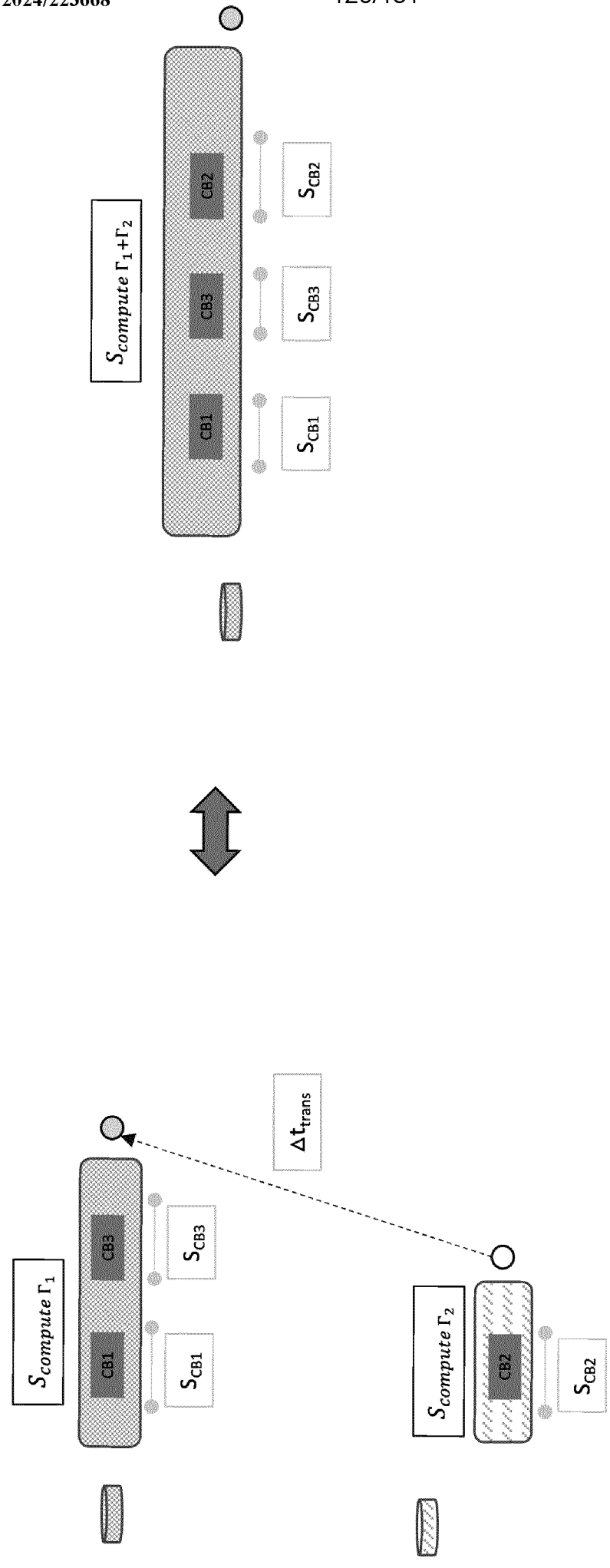
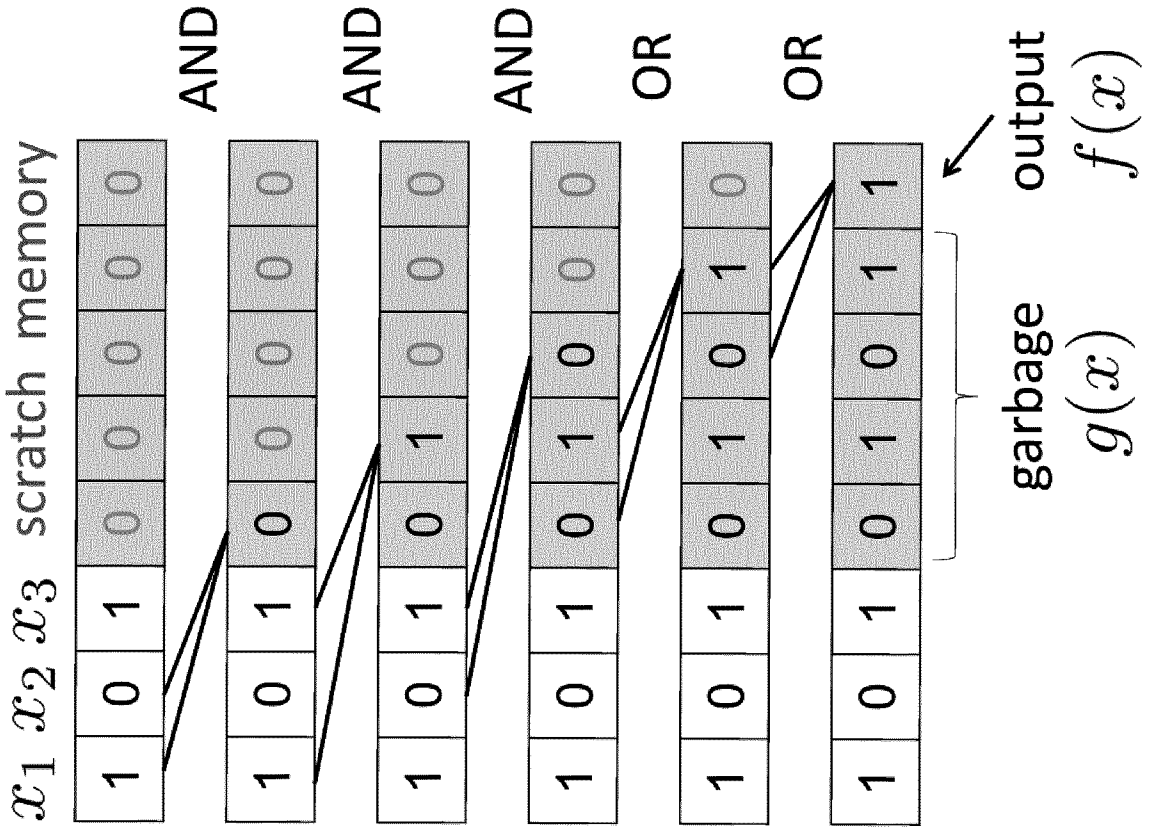

$S_{compute\ \Gamma_1+\Gamma_2}$
CB1
CB3
CB2
S_{CB1}
S_{CB3}
S_{CB2}
Δt_{trans}
$S_{compute\ \Gamma_1}$
CB1
CB3
S_{CB1}
S_{CB3}
$S_{compute\ \Gamma_2}$
CB2
S_{CB2}

Fig. 112


x_1 x_2 x_3 scratch memory
AND
AND
AND
OR
OR
output $f(x)$
garbage $g(x)$

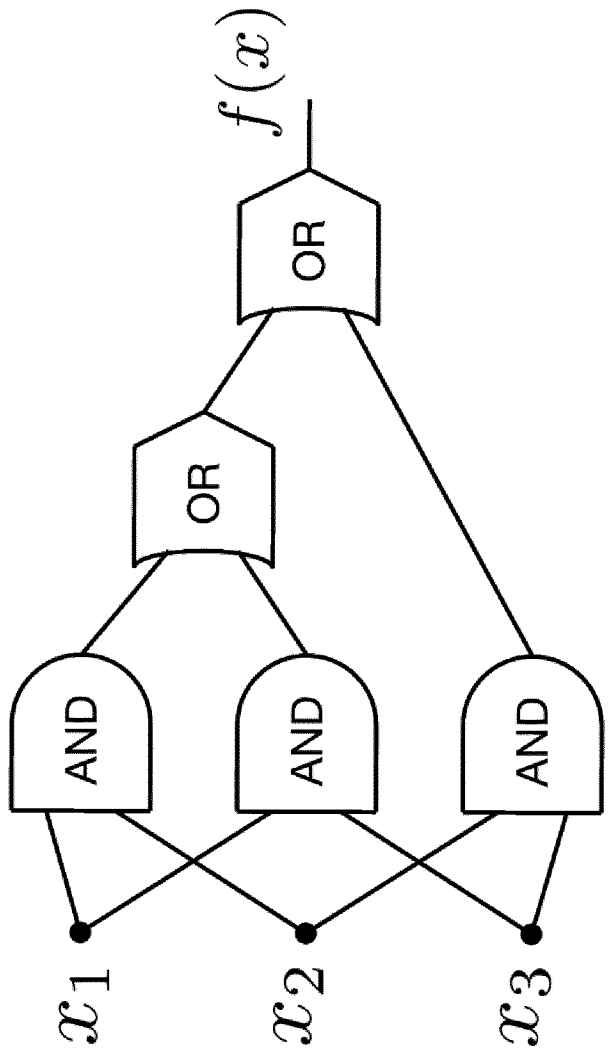

$f(x)$
OR
OR
AND
AND
AND
x_1
x_2
x_3

Fig. 113

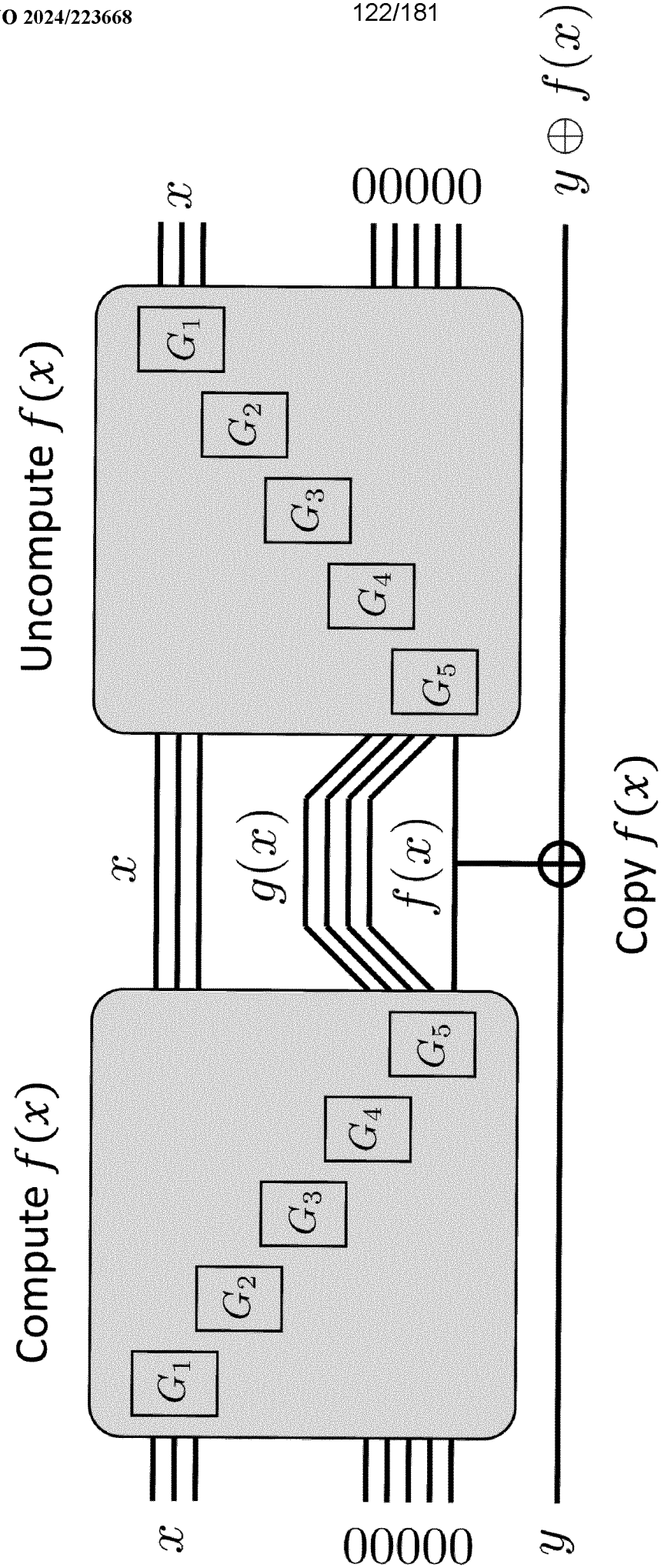
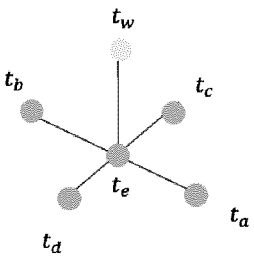

$y \oplus f(x)$
x
00000
Uncompute $f(x)$
G_1
G_2
G_3
G_4
G_5
x
$g(x)$
$f(x)$
Copy $f(x)$
Compute $f(x)$
G_5
G_4
G_3
G_2
G_1
x
00000
y

Fig. 114

Fig. 59: $t_w = \kappa * (t_a + t_b + t_c + t_d - 4 * t_e) + t_e$

stencil



read

write

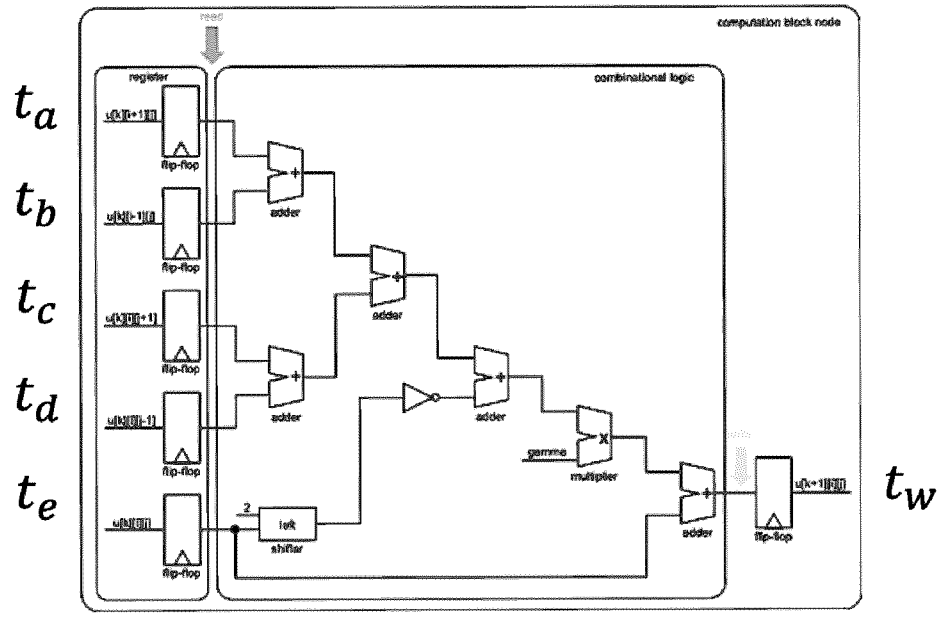


Fig. 111c

combinational ic-circuit

Fig. 115

reversible circuit mapping input strings to output

garbage g(x)

ta	tb	tc	td	te	g1	g2	g3	g4	g5	g6	
0/1	0/1	0/1	0/1	0/1	0	0	0	0	0	0	

input

adder

adder

shifter

adder

adder

multiplier

adder

total space: $\sum(t_i + g_k) \cdot dr$

output f(x)

$i \in [a, b, c, d, e]$

$k \in [0,6]$

$dr = \begin{cases} 32 \\ 64 \end{cases}$

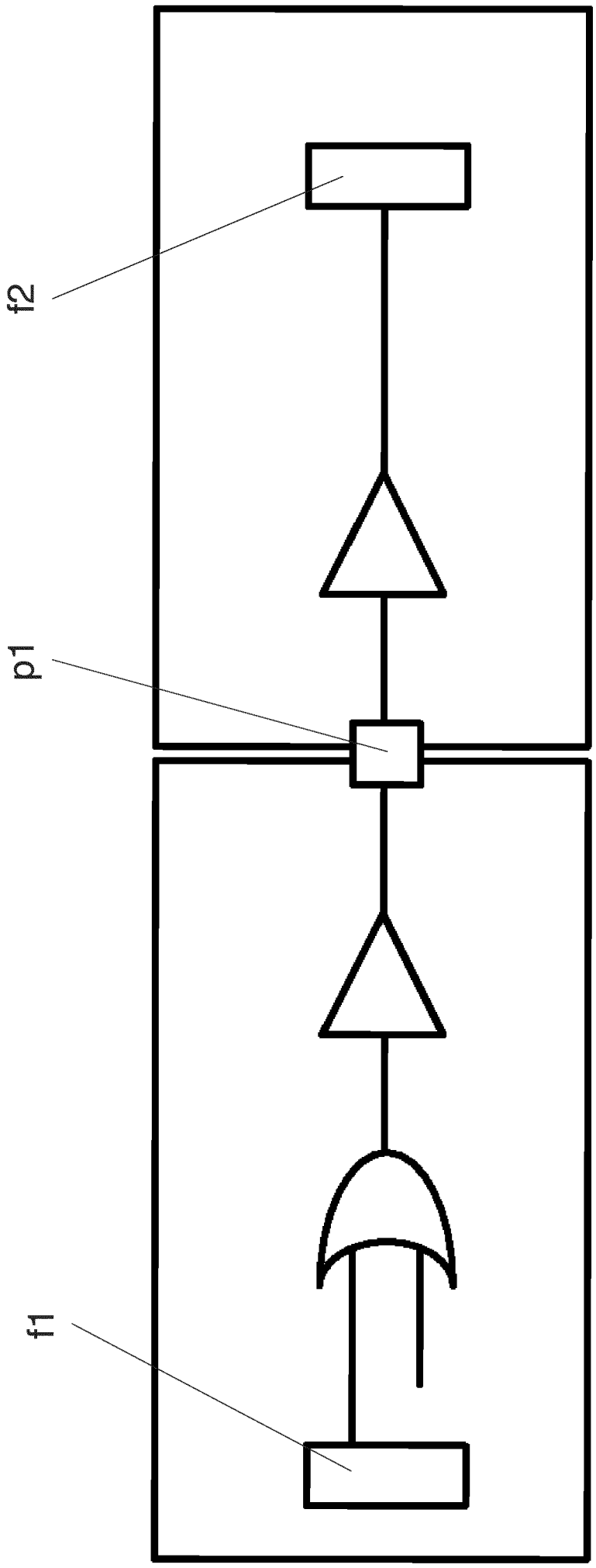


Fig. 116

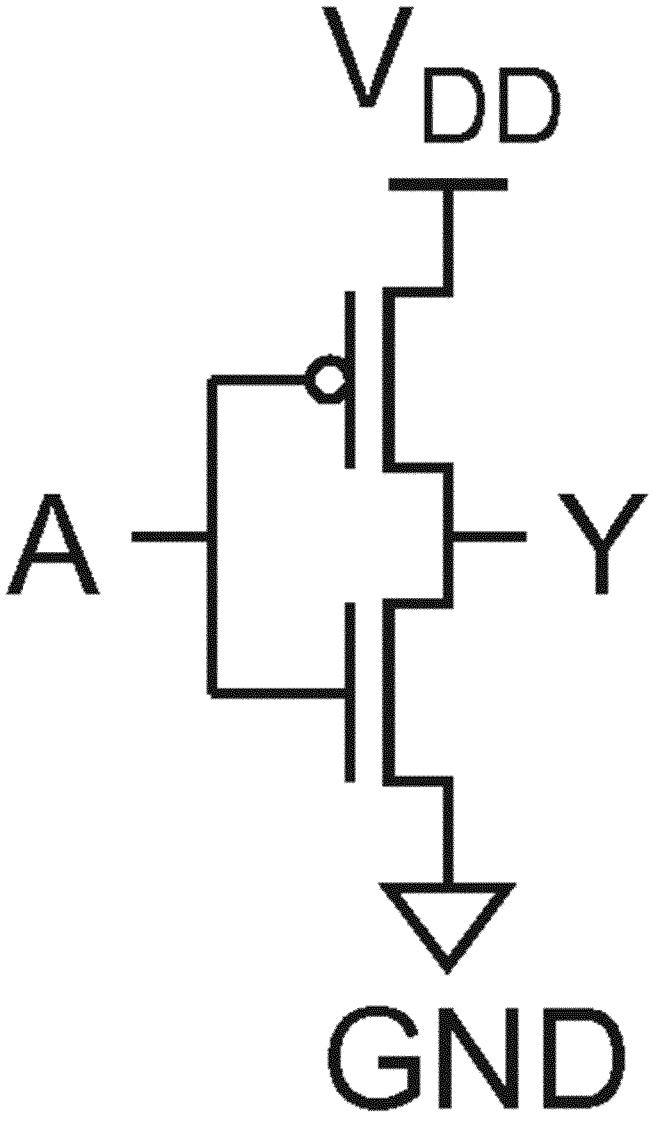


Fig. 117a

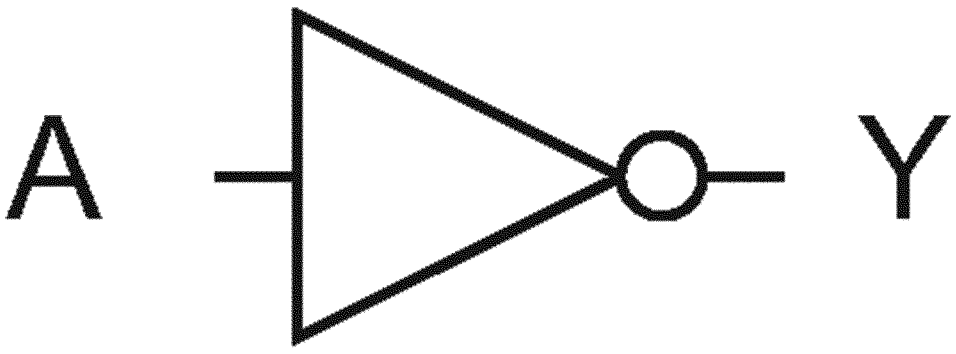


Fig. 117b

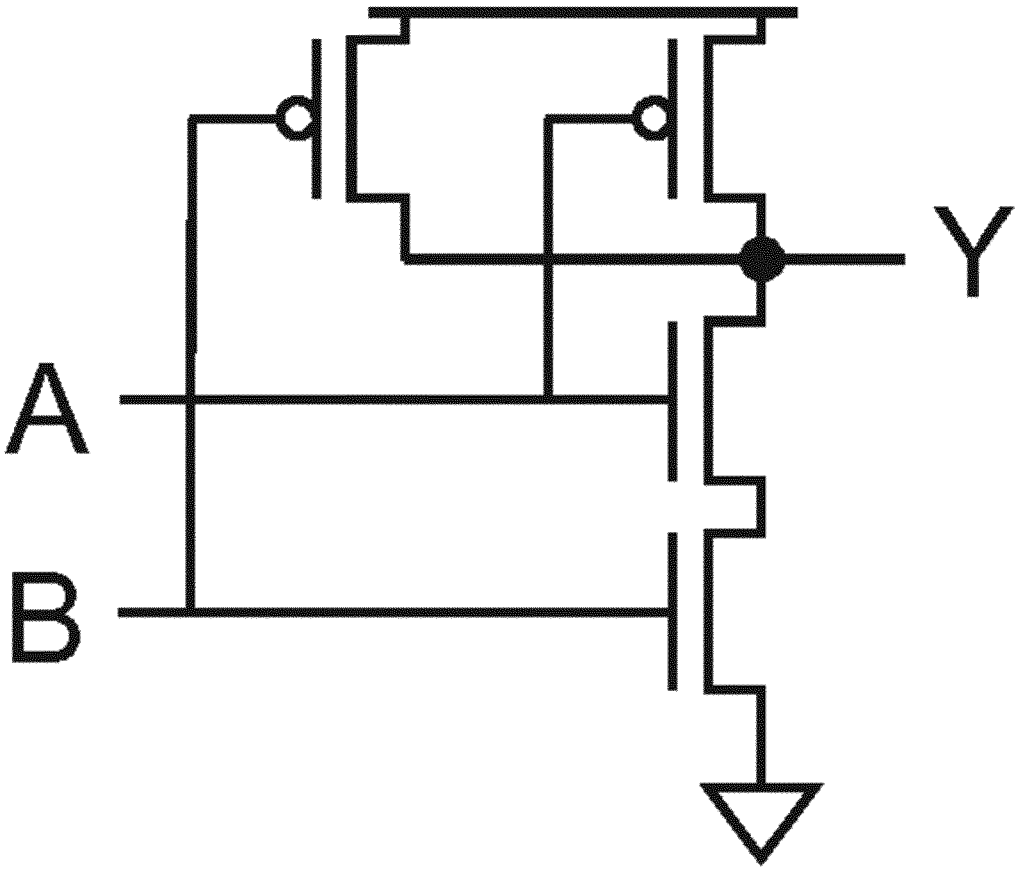


Fig. 117c

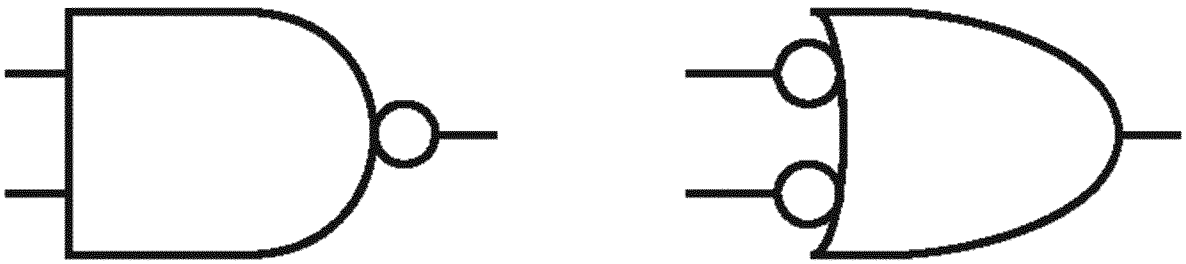


Fig. 117d

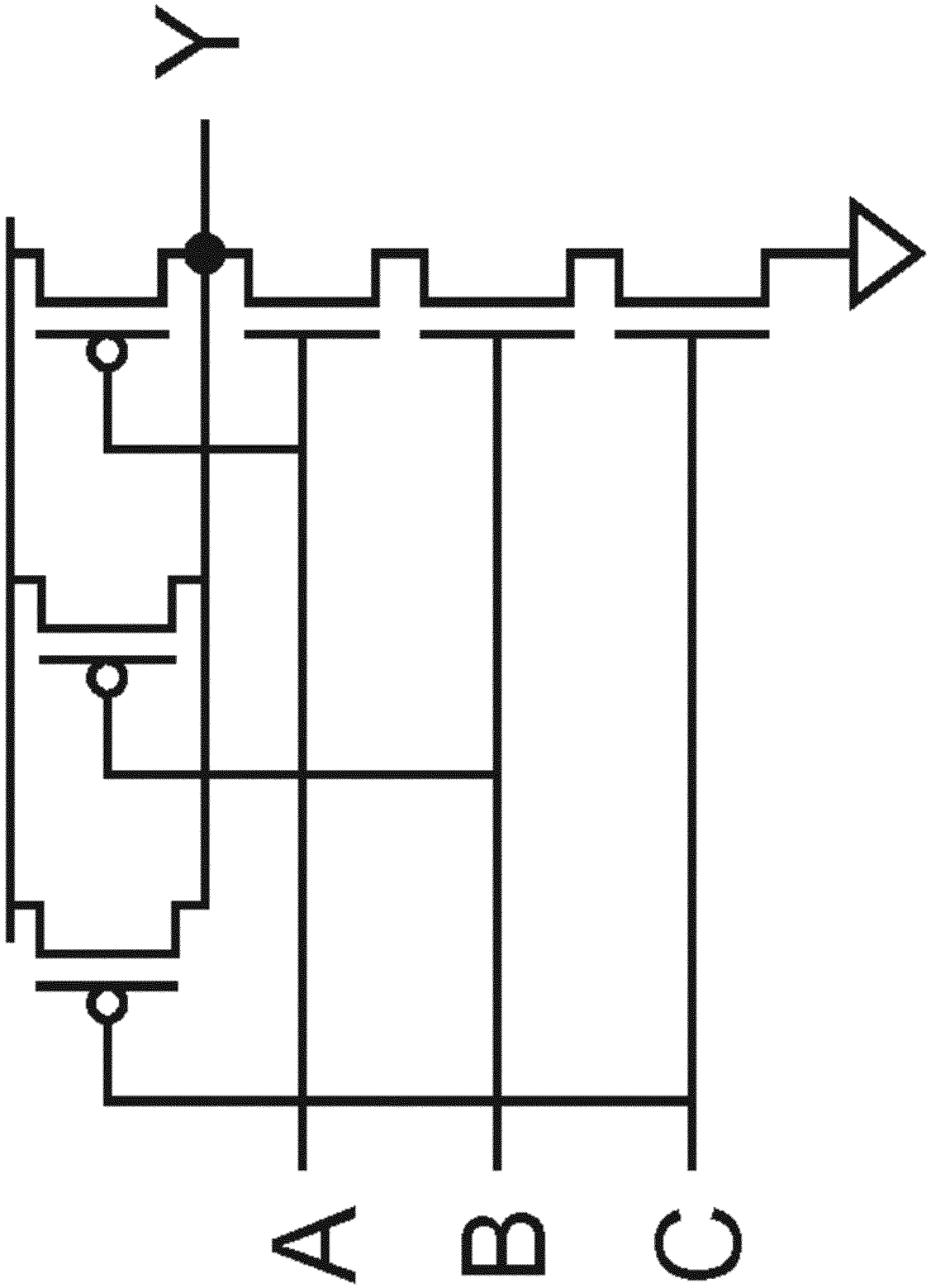


Fig. 117e

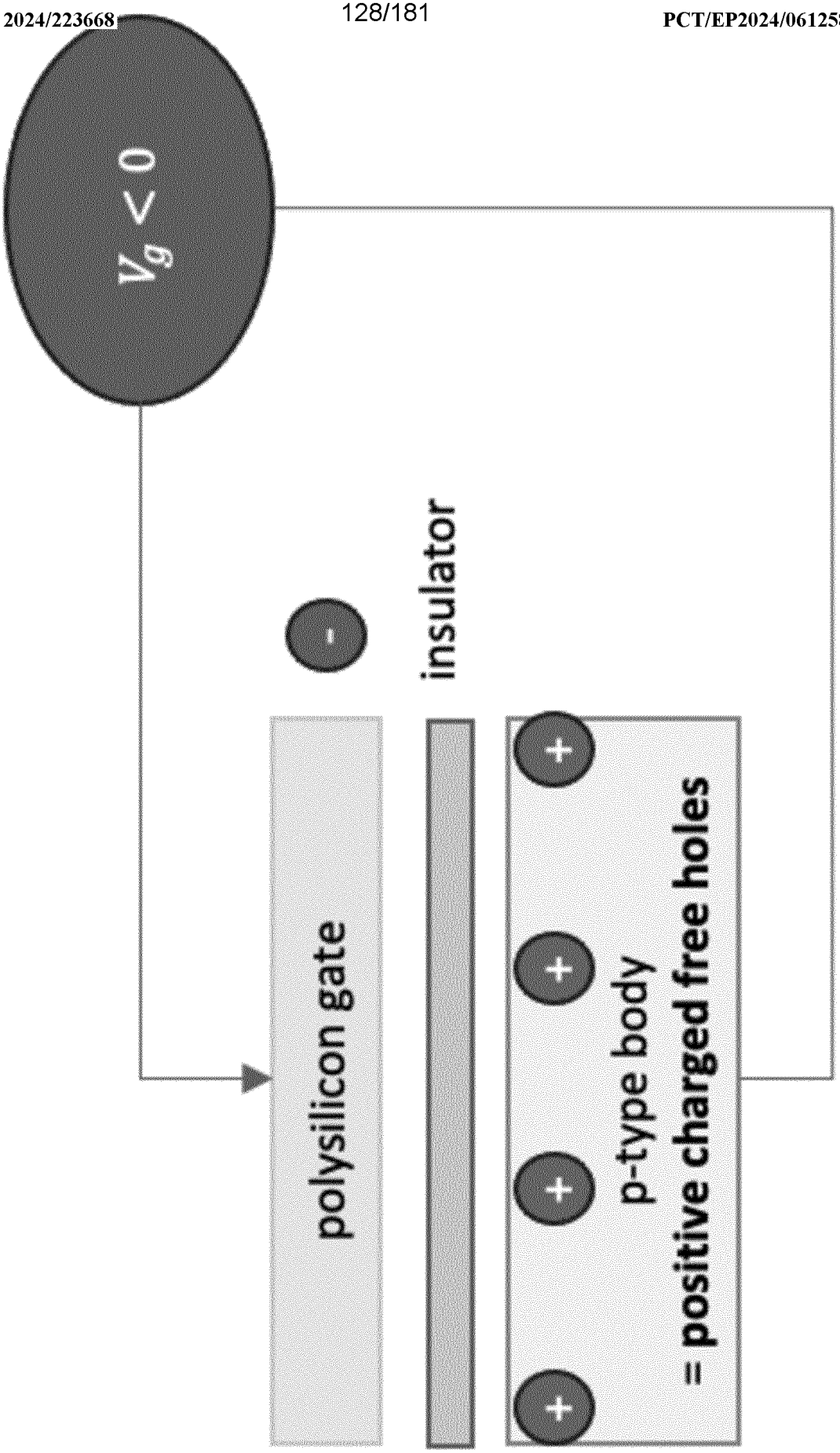

$V_g < 0$
polysilicon gate
insulator
p-type body
= positive charged free holes

Fig. 118a

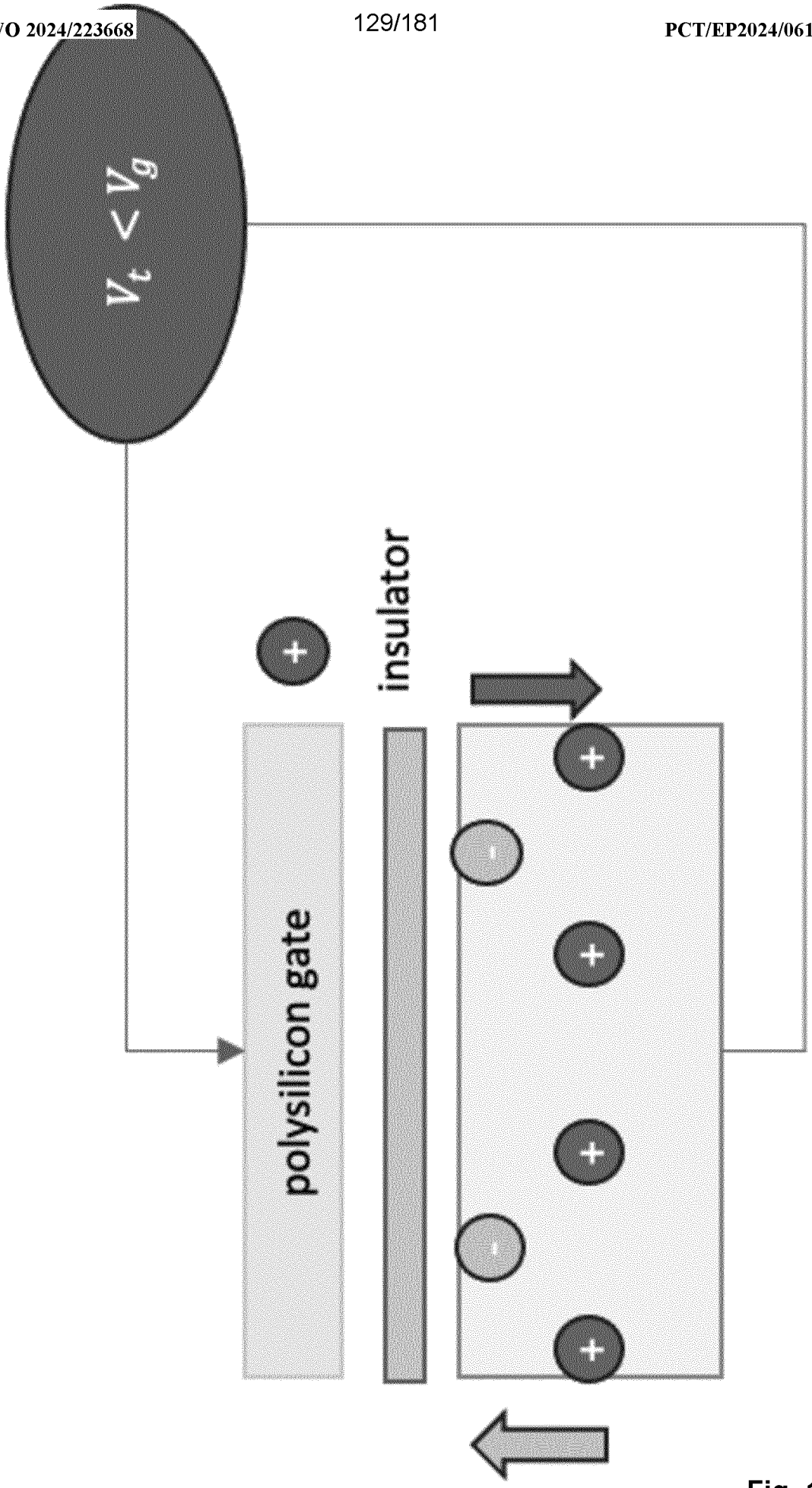


Fig. 118b

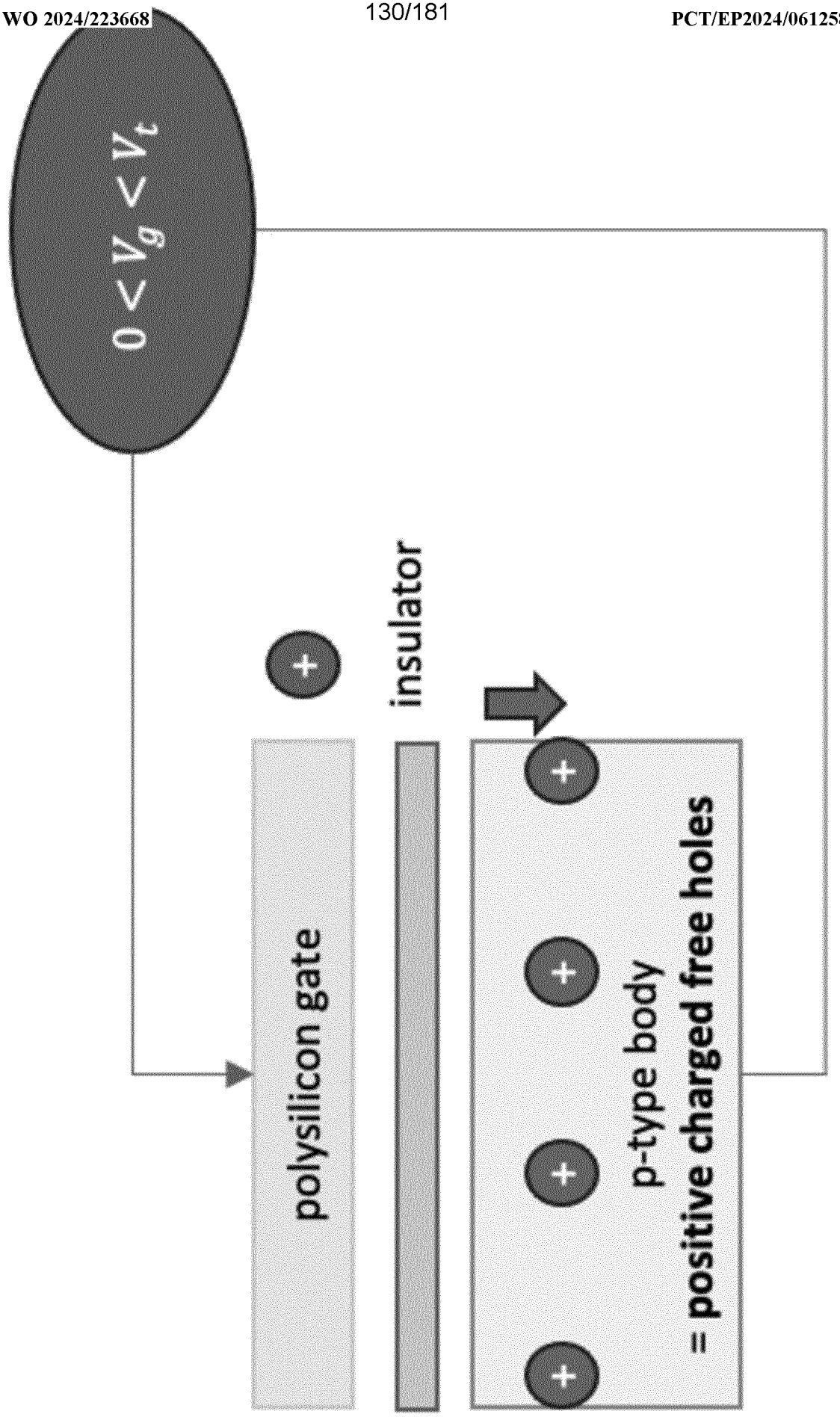

$0 < V_g < V_t$
polysilicon gate
insulator
p-type body
= positive charged free holes

Fig. 118c

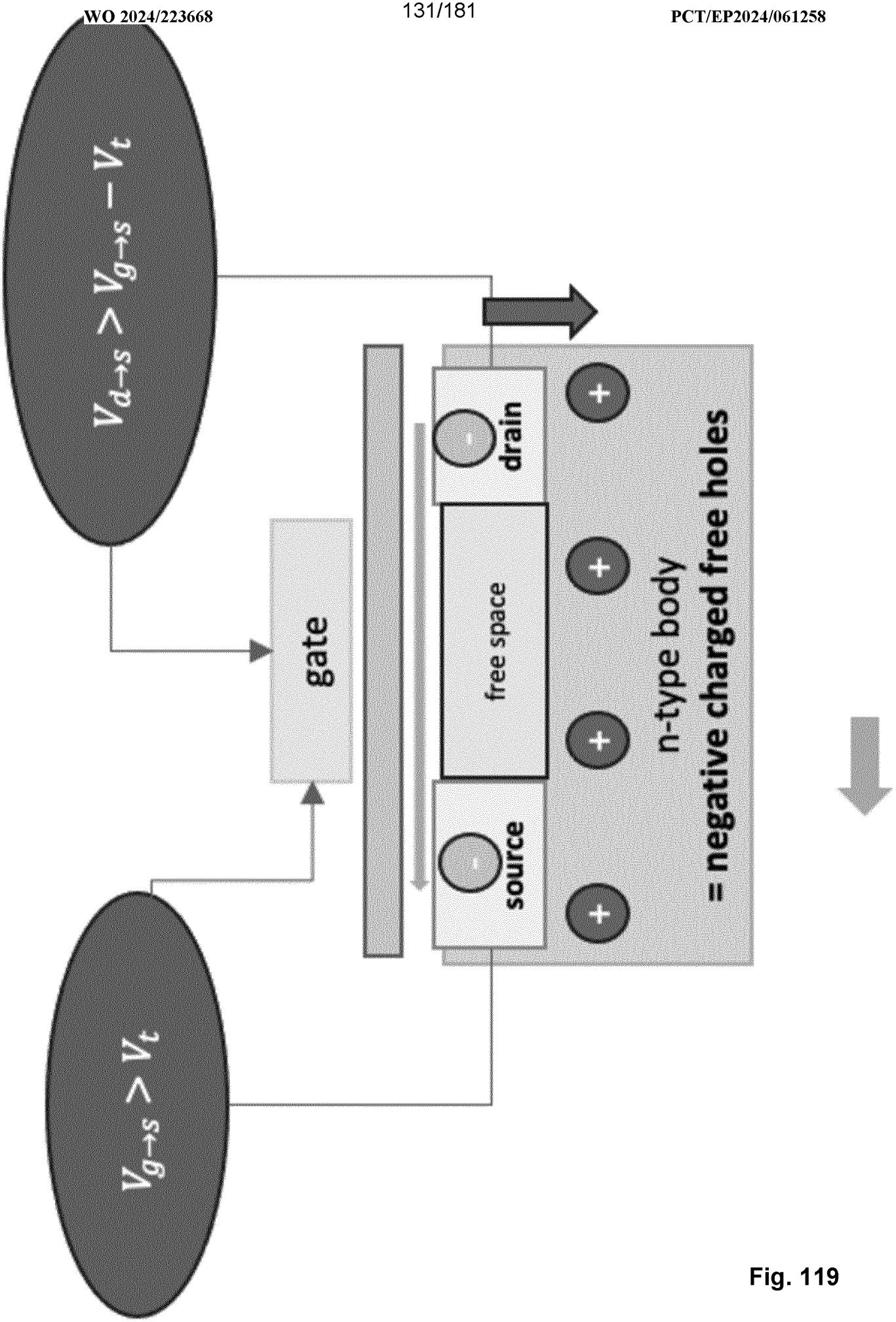

$V_{d\leftrightarrow s} < V_{g\leftrightarrow s} - V_t$
gate
drain
free space
source
n-type body
= negative charged free holes
$V_{g\leftrightarrow s} < V_t$

Fig. 119

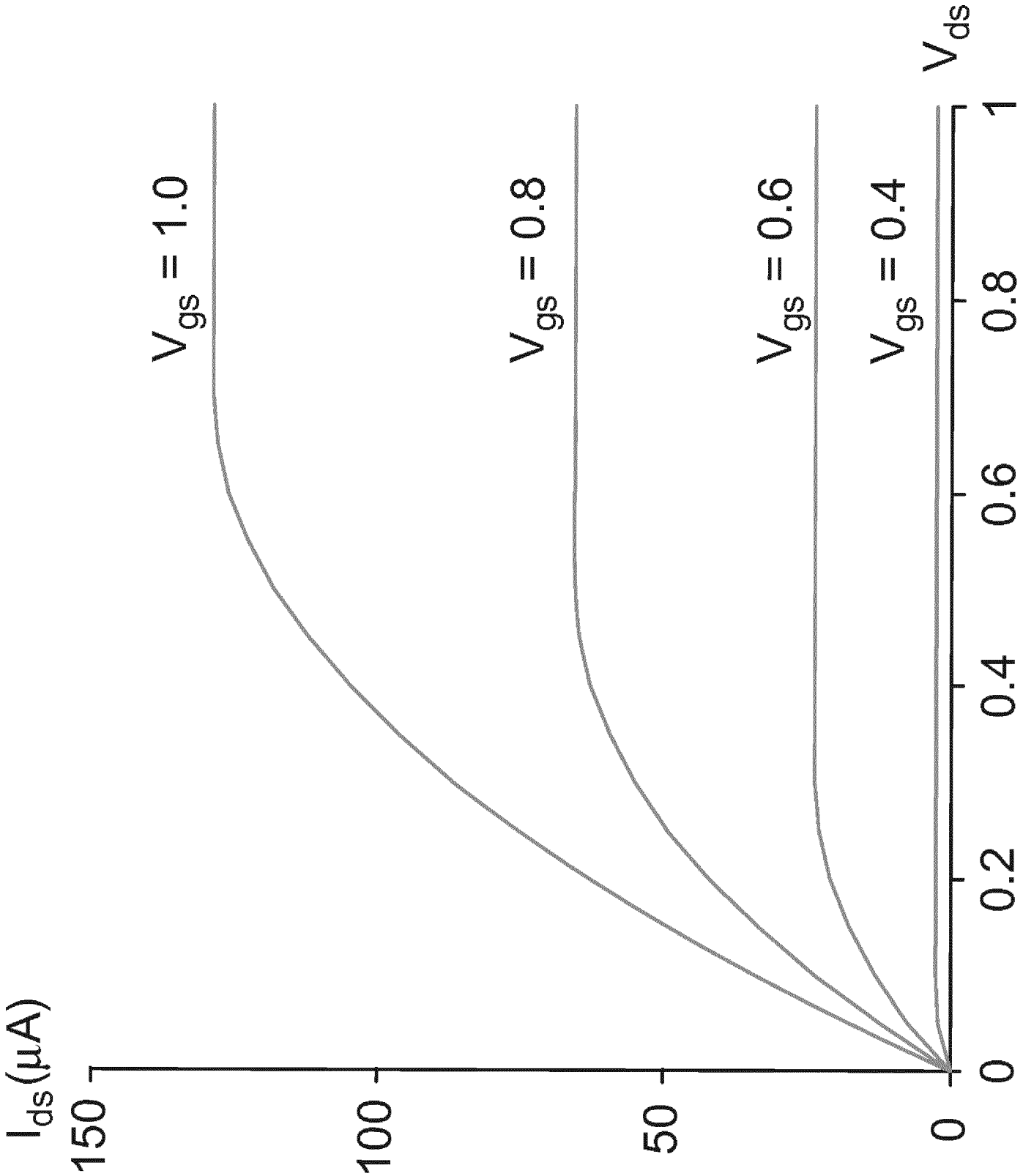


Fig. 120a

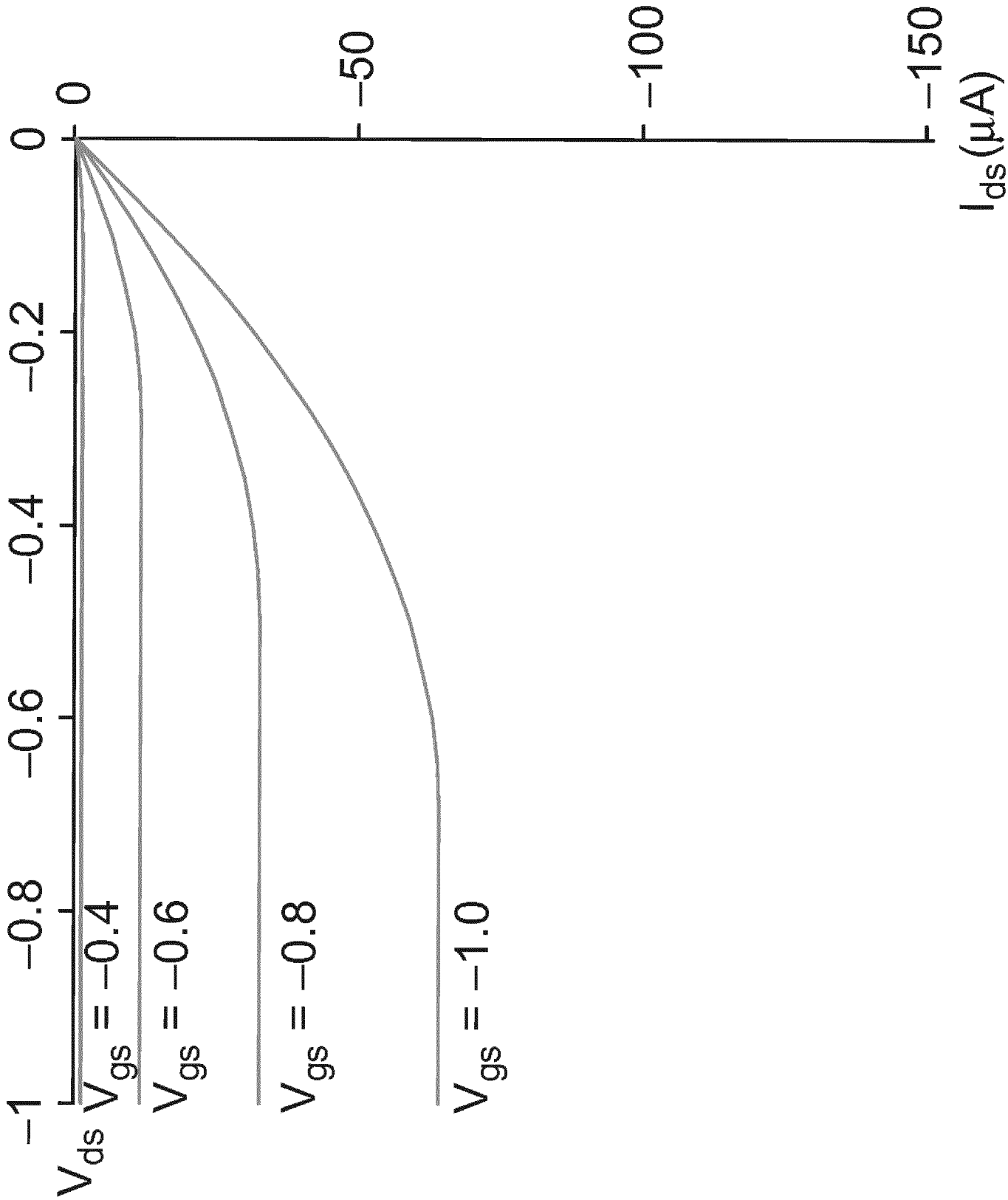


Fig. 120b

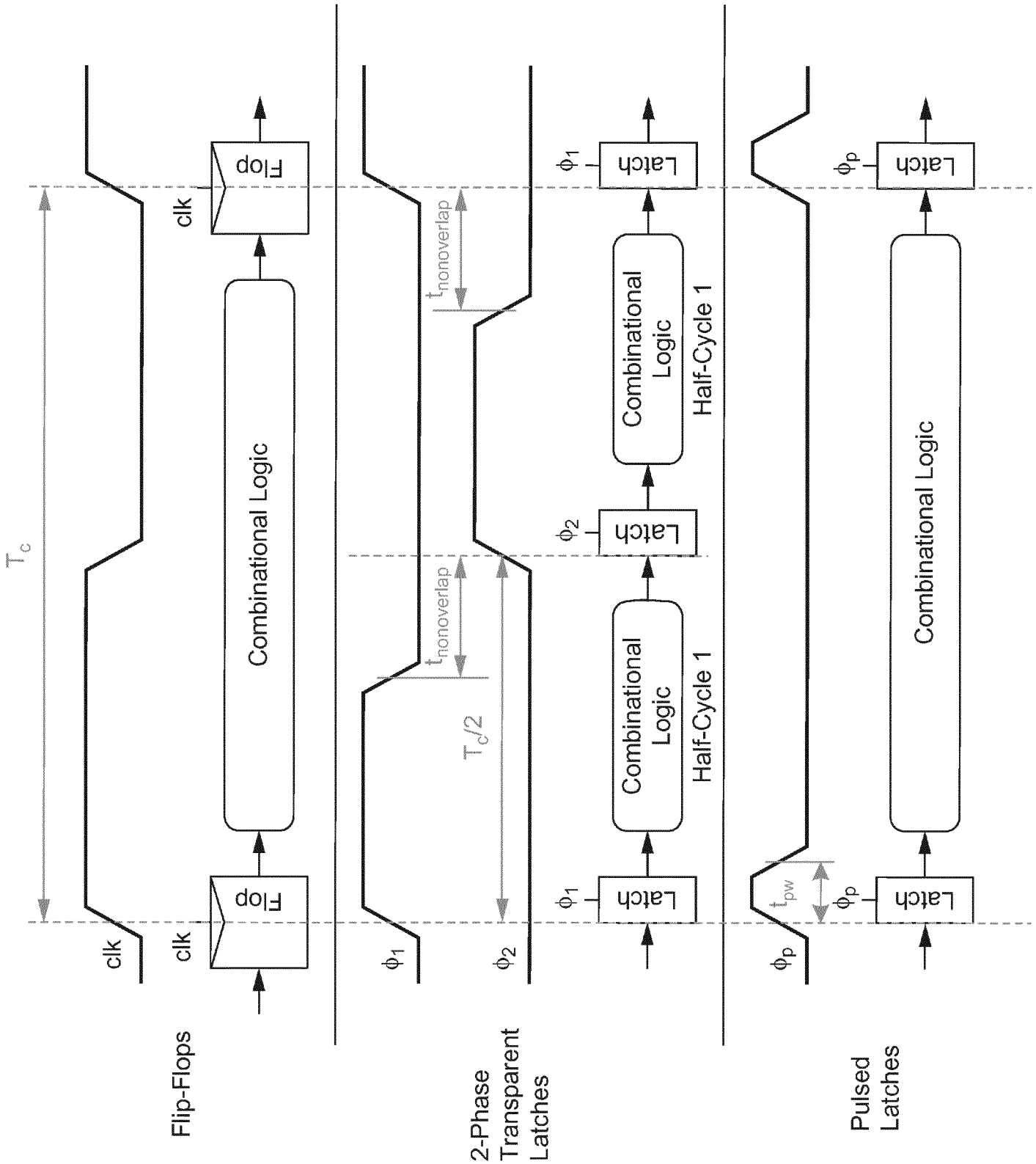

Flip-Flops
2-Phase Transparent Latches
Pulsed Latches
clk
Flop
Combinational Logic
T_c
$T_c/2$
$t_{nonoverlap}$
t_{pw}
ϕ_1
ϕ_2
ϕ_p
Latch
Half-Cycle 1

Fig. 121

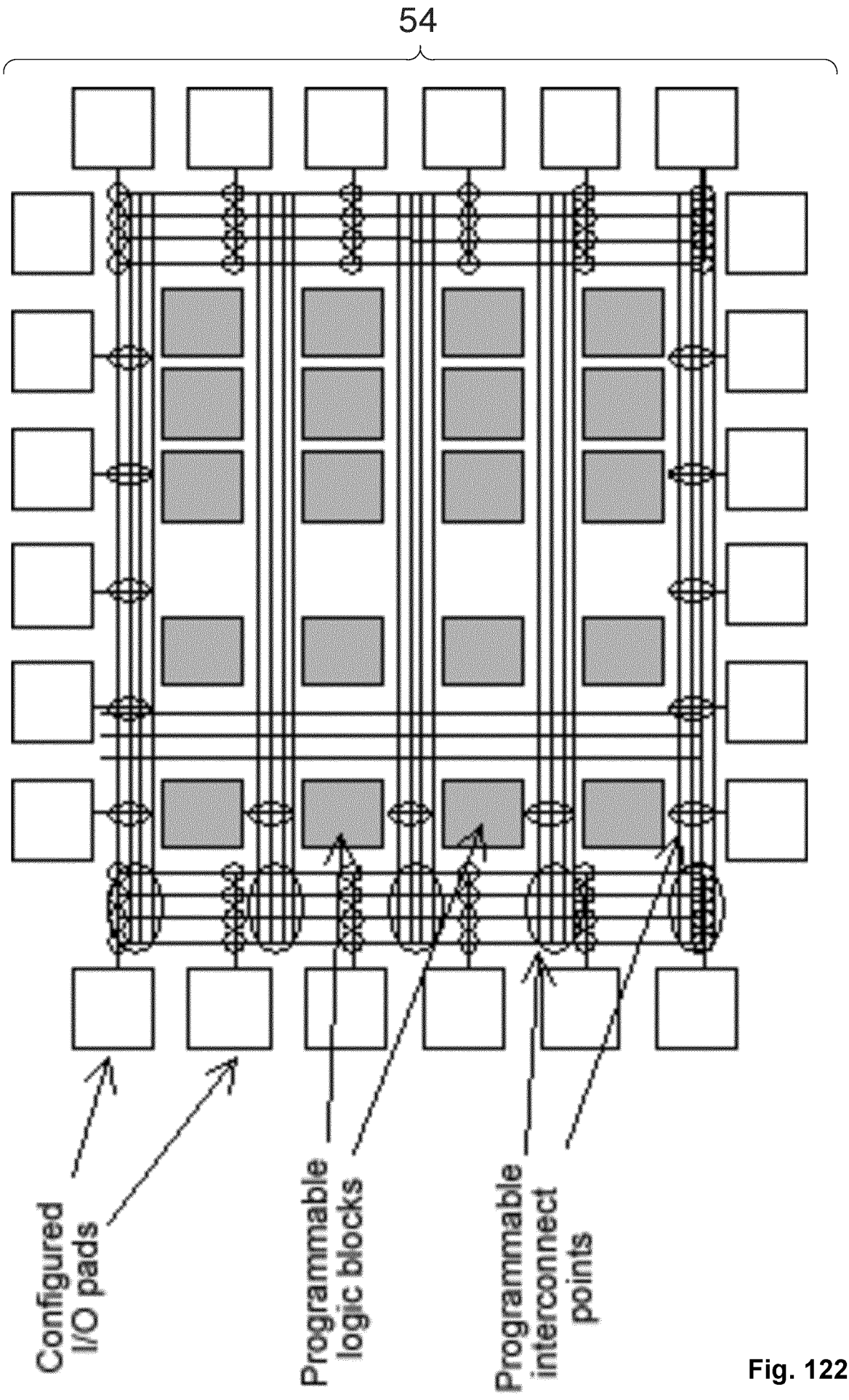

54
Configured I/O pads
Programmable logic blocks
Programmable interconnect points

Fig. 122

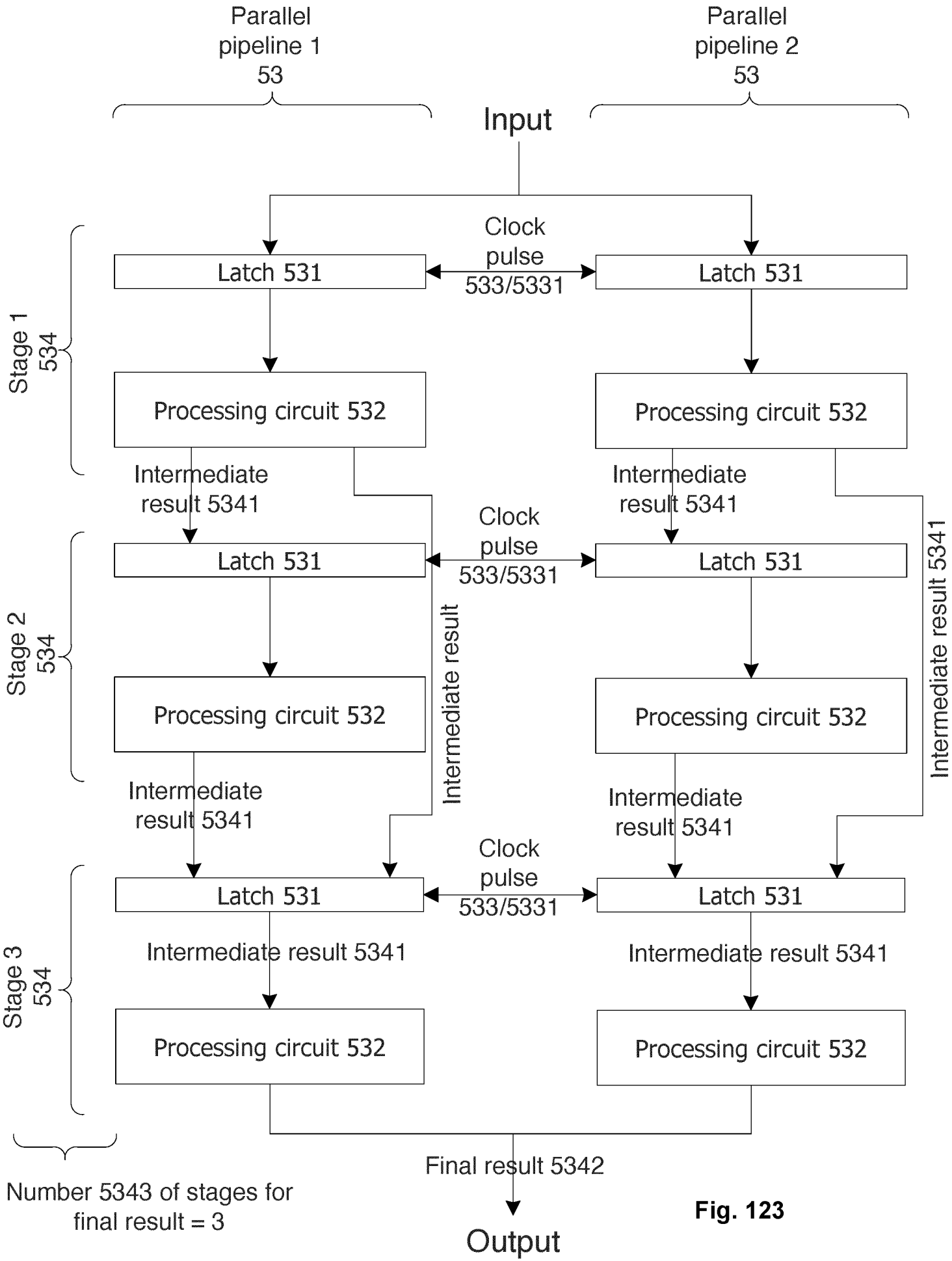

Parallel pipeline 1 53
Parallel pipeline 2 53
Input
Clock pulse 533/5331
Latch 531
Stage 1 534
Processing circuit 532
Intermediate result 5341
Stage 2 534
Intermediate result
Stage 3 534
Final result 5342
Output
Number 5343 of stages for final result = 3

Fig. 123

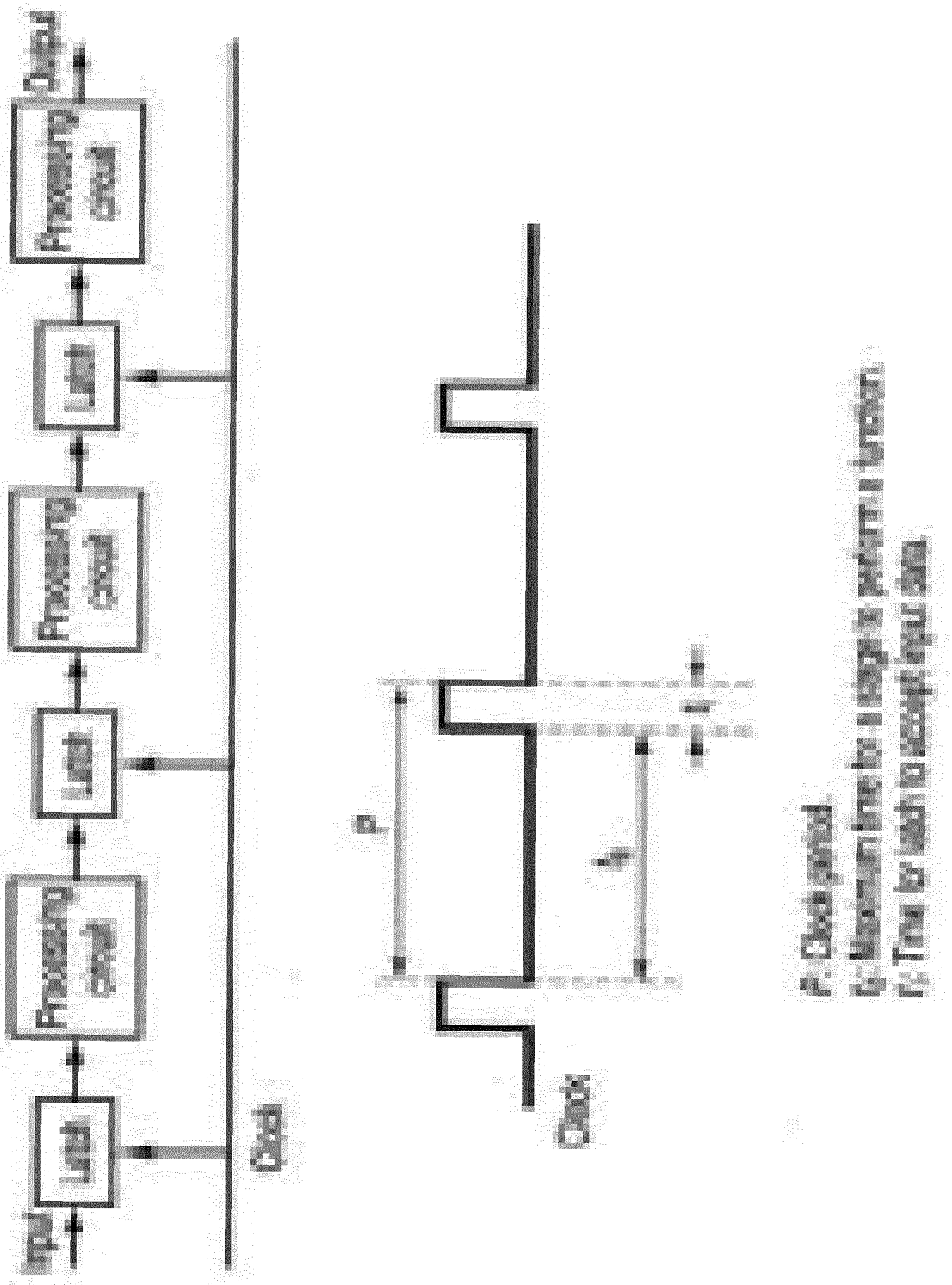


Fig. 124

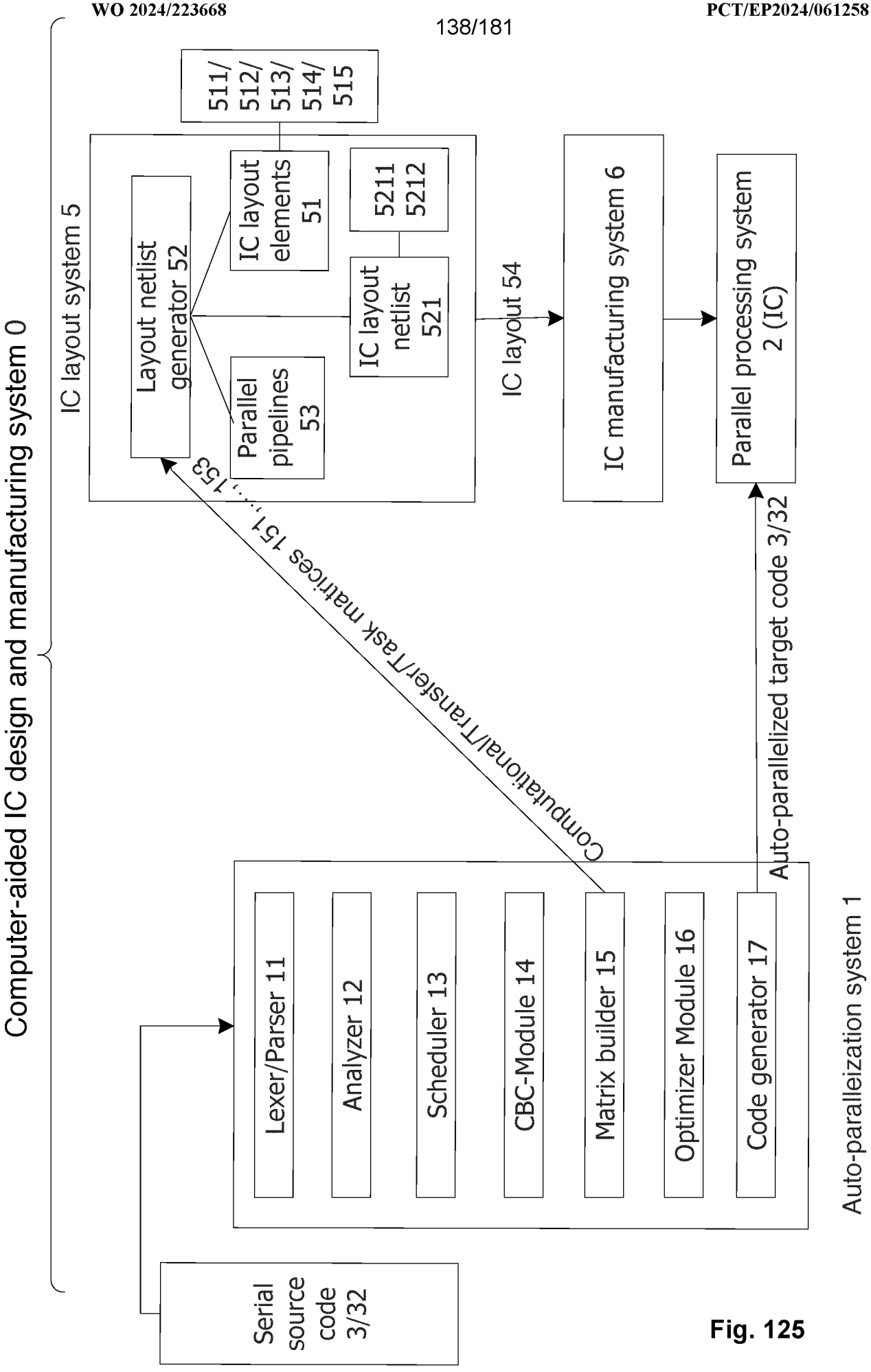

Computer-aided IC design and manufacturing system 0
IC layout system 5
Layout netlist generator 52
IC layout elements 51
511/ 512/ 513/ 514/ 515
5211 5212
IC layout netlist 521
Parallel pipelines 53
IC layout 54
IC manufacturing system 6
Parallel processing system 2 (IC)
Computational/Transfer/Task matrices 151,...,153
Auto-parallelized target code 3/32
Lexer/Parser 11
Analyzer 12
Scheduler 13
CBC-Module 14
Matrix builder 15
Optimizer Module 16
Code generator 17
Auto-parallelization system 1
Serial source code 3/32

Fig. 125

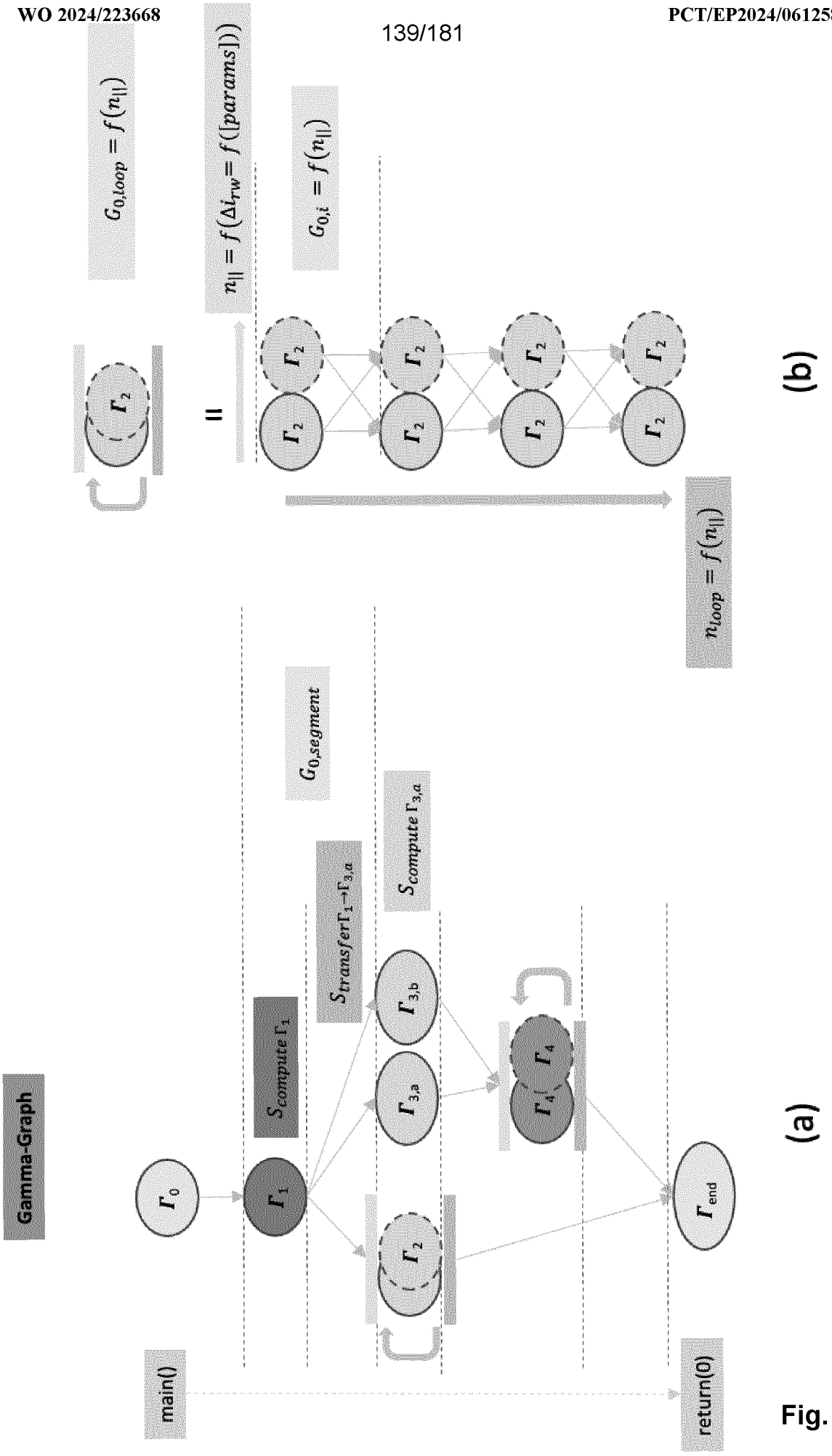

Gamma-Graph
main()
Γ_0
Γ_1
$S_{compute}\Gamma_1$
$G_{0,segment}$
$S_{transfer}\Gamma_1 \rightarrow \Gamma_{3,a}$
$S_{compute}\Gamma_{3,a}$
$\Gamma_{3,b}$
$\Gamma_{3,a}$
Γ_2
Γ_4
Γ_{end}
return(0)
(a)
$G_{0,loop} = f(n_{||})$
$n_{||} = f(\Delta i_{rw} = f([params]))$
$G_{0,i} = f(n_{||})$
$n_{loop} = f(n_{||})$
(b)

Fig. 126

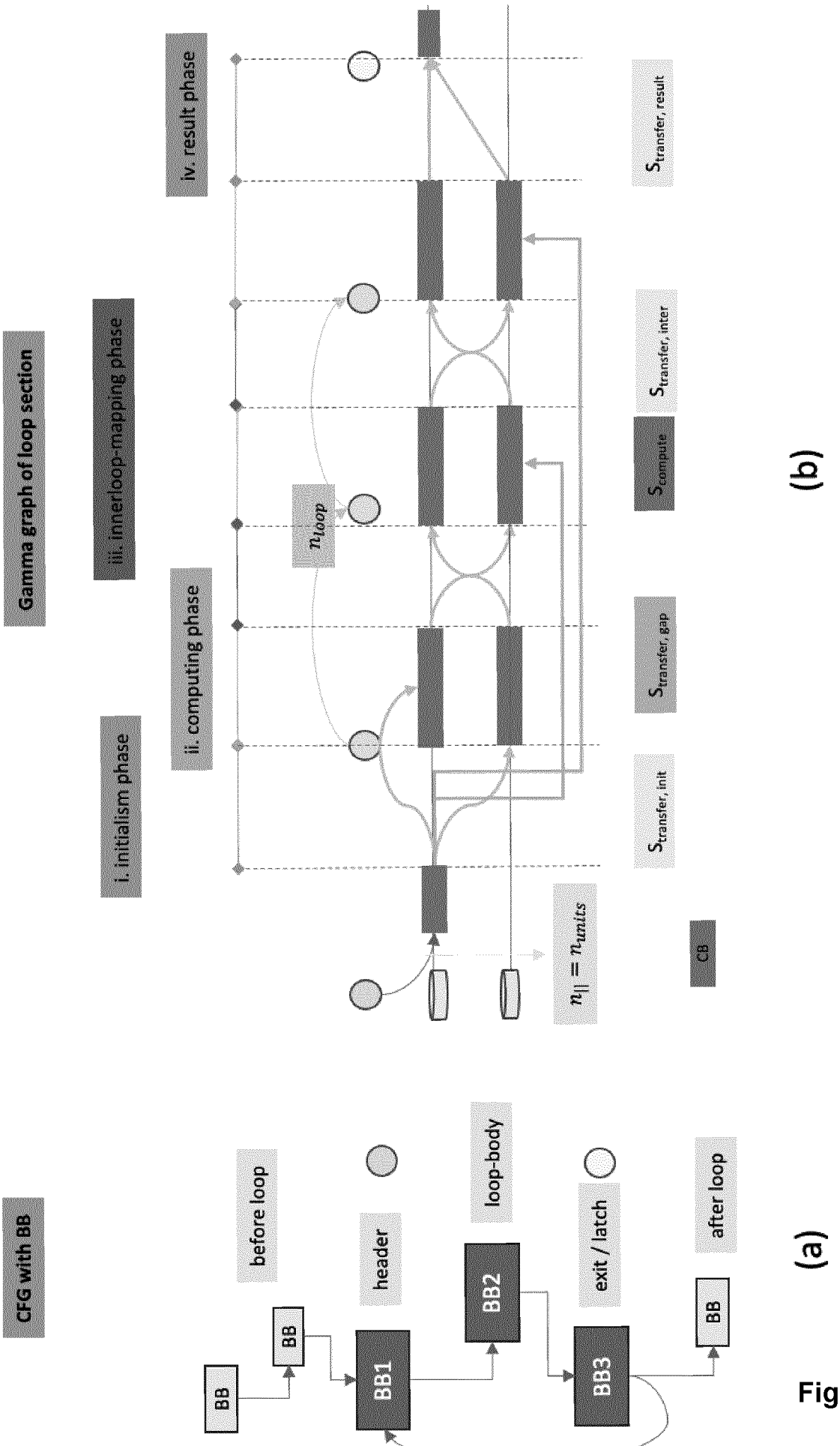

CFG with BB
BB
BB
BB1
BB2
BB3
BB
before loop
header
loop-body
exit / latch
after loop
(a)
Gamma graph of loop section
i. initialism phase
iii. innerloop-mapping phase
ii. computing phase
iv. result phase
n_{loop}
$n_{||} = n_{units}$
$S_{transfer,\ init}$
$S_{transfer,\ gap}$
$S_{compute}$
$S_{transfer,\ inter}$
$S_{transfer,\ result}$
CB
(b)

Fig. 127

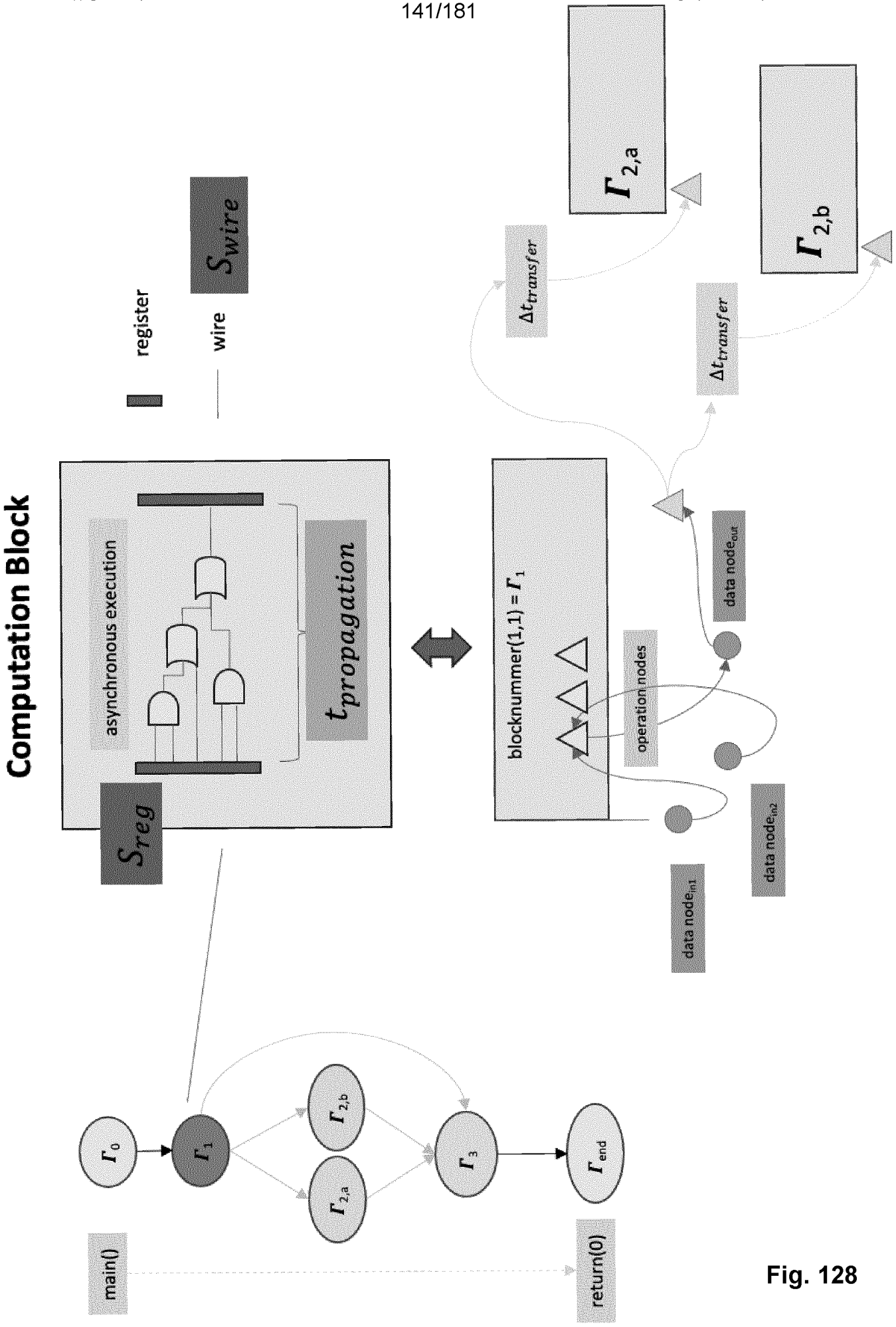

Computation Block
register
wire
S_{wire}
S_{reg}
asynchronous execution
$t_{propagation}$
$\Delta t_{transfer}$
$\Gamma_{2,a}$
$\Gamma_{2,b}$
blocknummer(1,1) = Γ_1
operation nodes
data node$_{out}$
data node$_{in1}$
data node$_{in2}$
Γ_0
Γ_1
$\Gamma_{2,b}$
$\Gamma_{2,a}$
Γ_3
Γ_{end}
main()
return(0)

Fig. 128

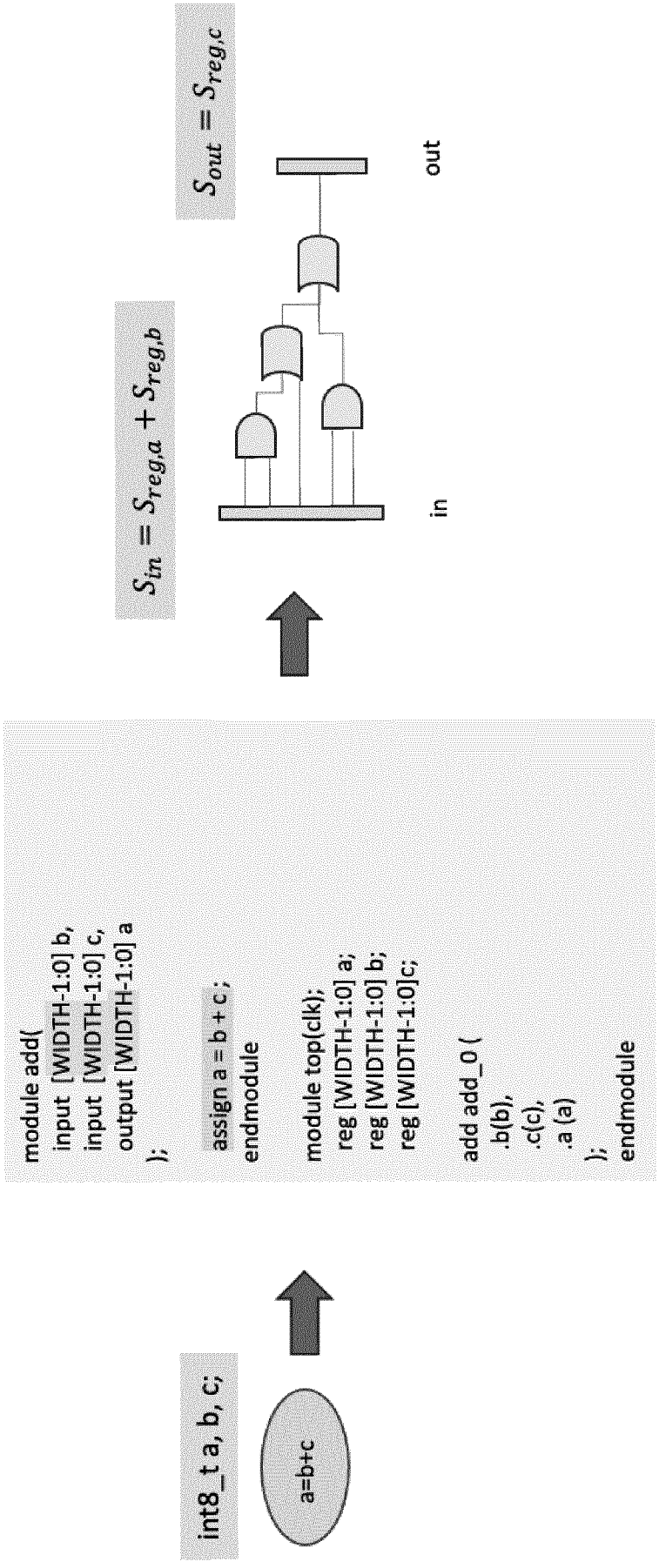

int8_t a, b, c;
a=b+c
module add(
input [WIDTH-1:0] b,
input [WIDTH-1:0] c,
output [WIDTH-1:0] a
);
assign a = b + c ;
endmodule
module top(clk);
reg [WIDTH-1:0] a;
reg [WIDTH-1:0] b;
reg [WIDTH-1:0]c;
add add_0 (
.b(b),
.c(c),
.a (a)
);
endmodule
$S_{in} = S_{reg,a} + S_{reg,b}$
$S_{out} = S_{reg,c}$
in
out

Fig. 129

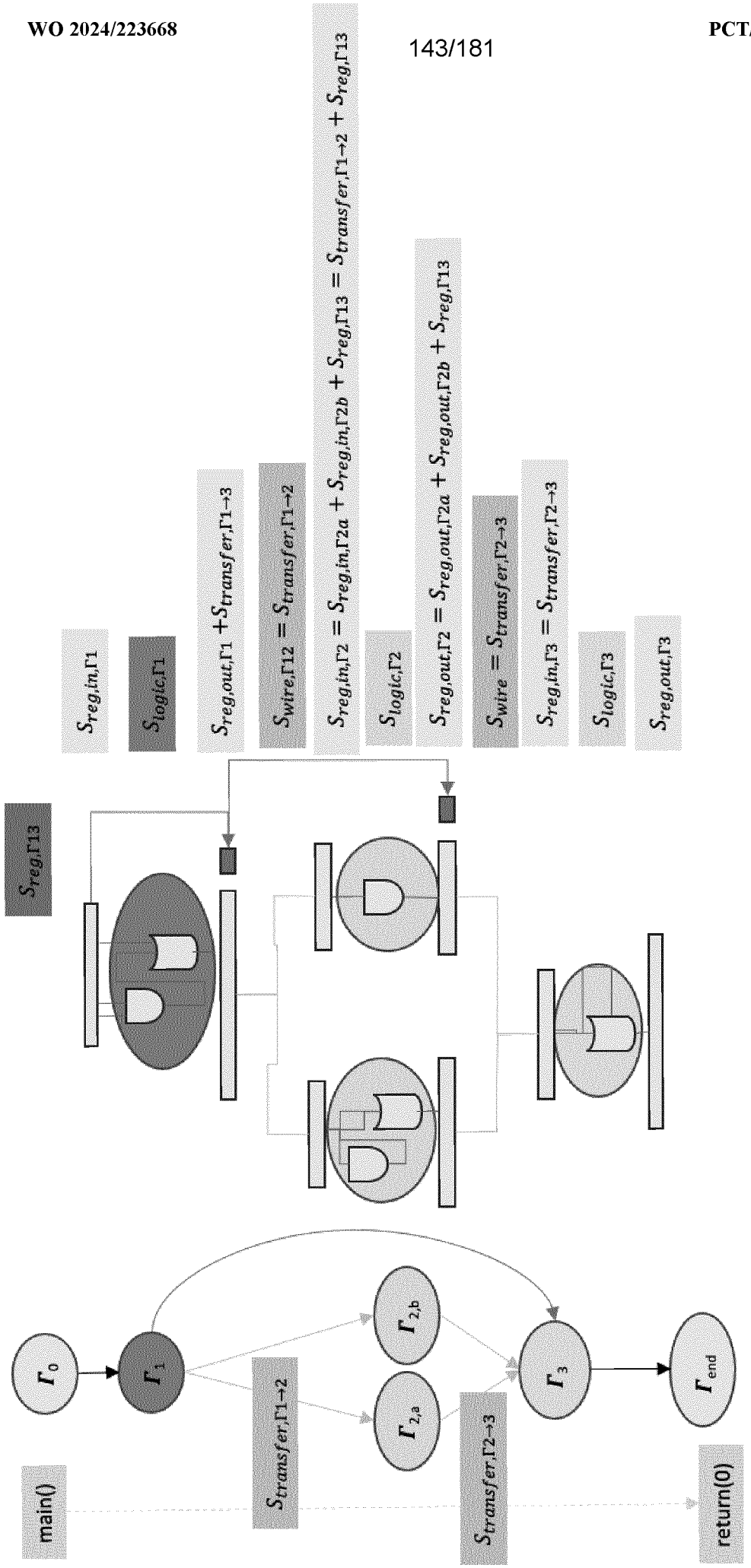

$S_{reg,in,\Gamma1}$
$S_{logic,\Gamma1}$
$S_{reg,out,\Gamma1} + S_{transfer,\Gamma1\to3}$
$S_{wire,\Gamma12} = S_{transfer,\Gamma1\to2}$
$S_{reg,in,\Gamma2} = S_{reg,in,\Gamma2a} + S_{reg,in,\Gamma2b} + S_{reg,\Gamma13} = S_{transfer,\Gamma1\to2} + S_{reg,\Gamma13}$
$S_{logic,\Gamma2}$
$S_{reg,out,\Gamma2} = S_{reg,out,\Gamma2a} + S_{reg,out,\Gamma2b} + S_{reg,\Gamma13}$
$S_{wire} = S_{transfer,\Gamma2\to3}$
$S_{reg,in,\Gamma3} = S_{transfer,\Gamma2\to3}$
$S_{logic,\Gamma3}$
$S_{reg,out,\Gamma3}$
$S_{reg,\Gamma13}$
Γ_0
Γ_1
$\Gamma_{2,b}$
$\Gamma_{2,a}$
Γ_3
Γ_{end}
main()
$S_{transfer,\Gamma1\to2}$
$S_{transfer,\Gamma2\to3}$
return(0)

Fig. 130

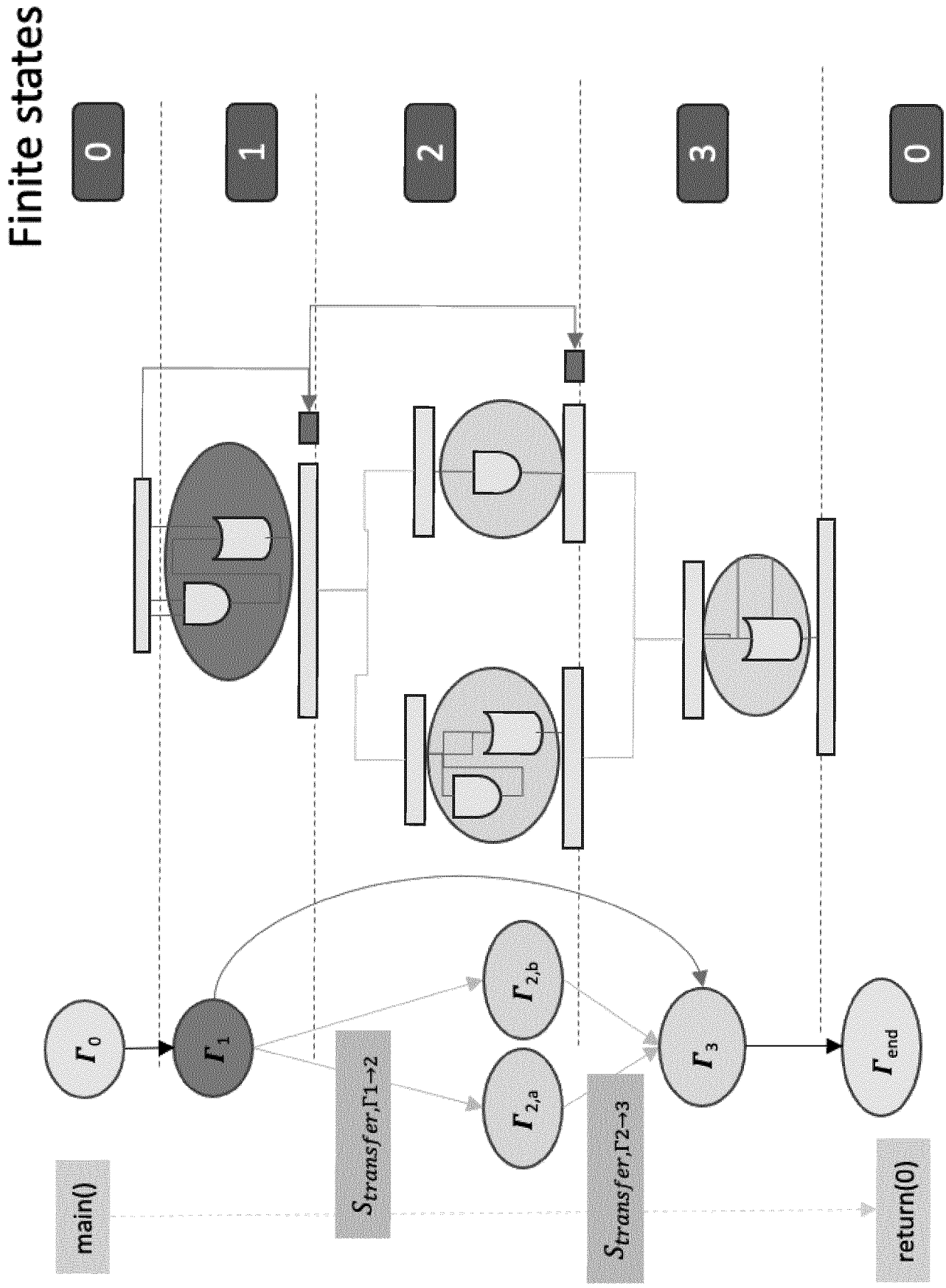

Finite states
0
1
2
3
0
Γ_0
Γ_1
$\Gamma_{2,b}$
$\Gamma_{2,a}$
Γ_3
Γ_{end}
$S_{transfer,\Gamma1\to2}$
$S_{transfer,\Gamma2\to3}$
main()
return(0)

Fig. 131

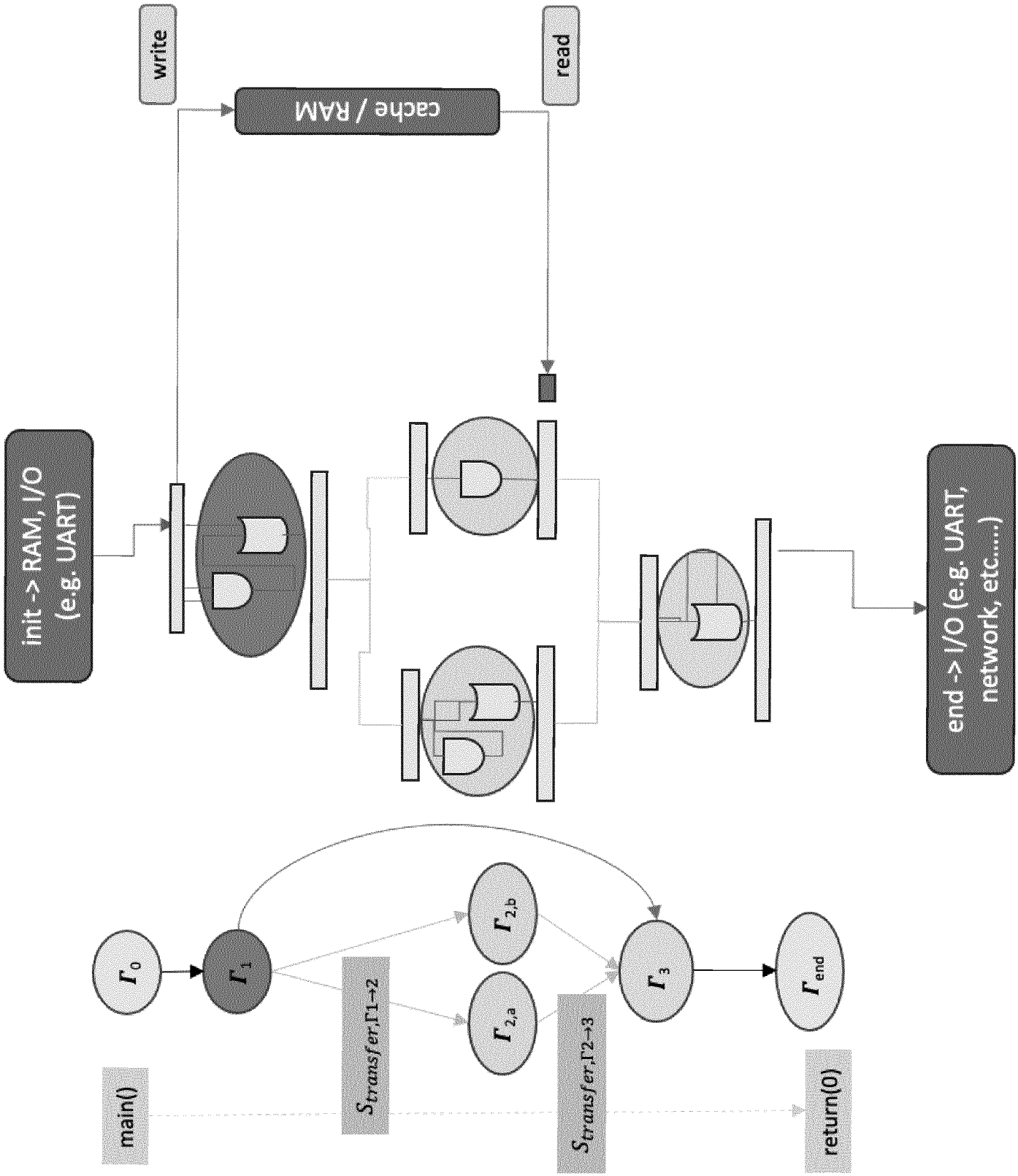

write
read
cache / RAM
init -> RAM, I/O (e.g. UART)
end -> I/O (e.g. UART, network, etc.....)
Γ_0
Γ_1
$\Gamma_{2,b}$
$\Gamma_{2,a}$
Γ_3
Γ_{end}
$S_{transfer,\Gamma1\to2}$
$S_{transfer,\Gamma2\to3}$
main()
return(0)

Fig. 132

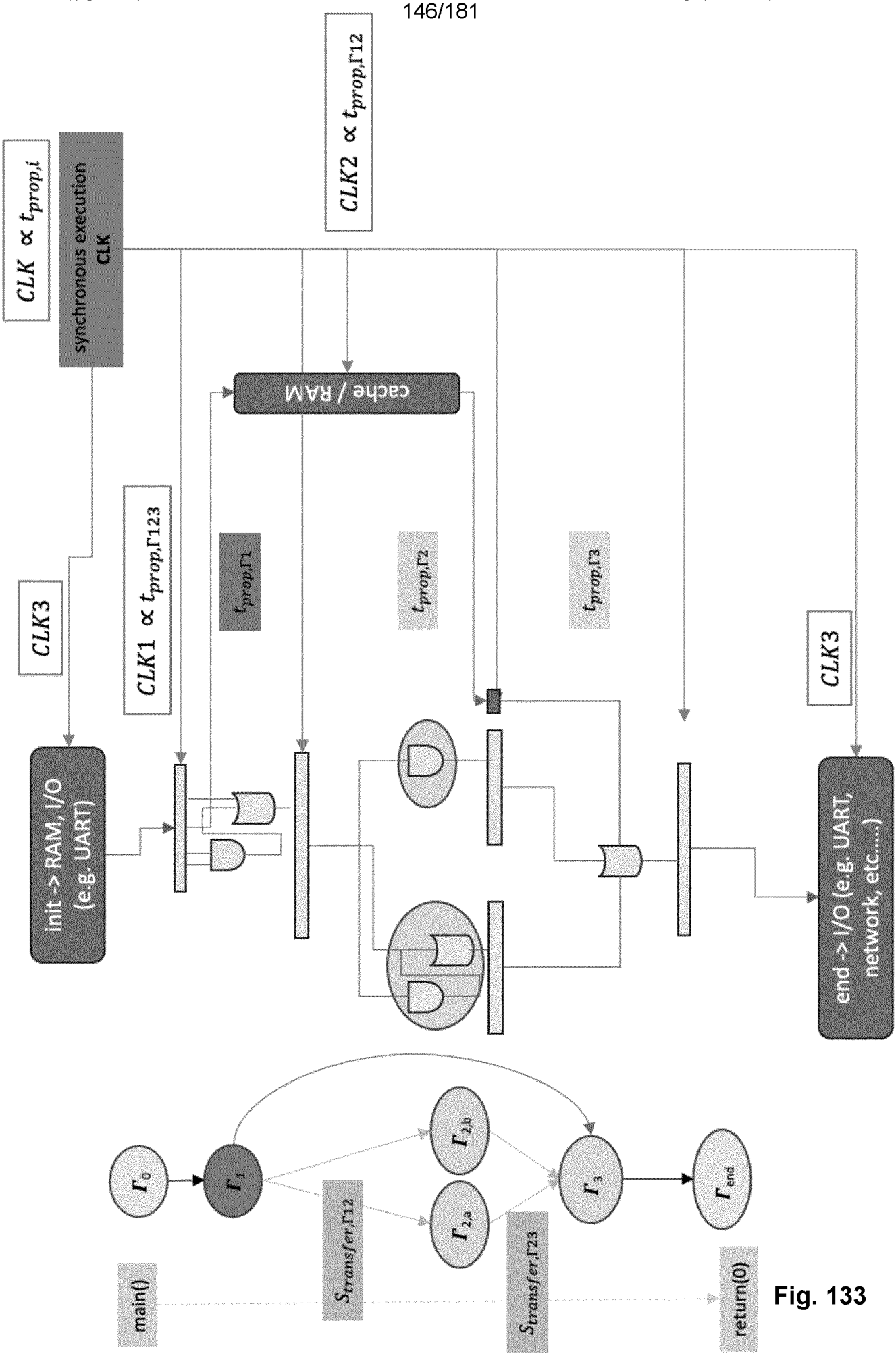

CLK ∝ t_prop,i
synchronous execution
CLK
CLK2 ∝ t_prop,Γ12
cache / RAM
CLK3
CLK1 ∝ t_prop,Γ123
t_prop,Γ1
t_prop,Γ2
t_prop,Γ3
CLK3
init -> RAM, I/O
(e.g. UART)
end -> I/O (e.g. UART,
network, etc.....)
Γ0
Γ1
Γ2,b
Γ2,a
Γ3
Γend
S_transfer,Γ12
S_transfer,Γ23
main()
return(0)

Fig. 133

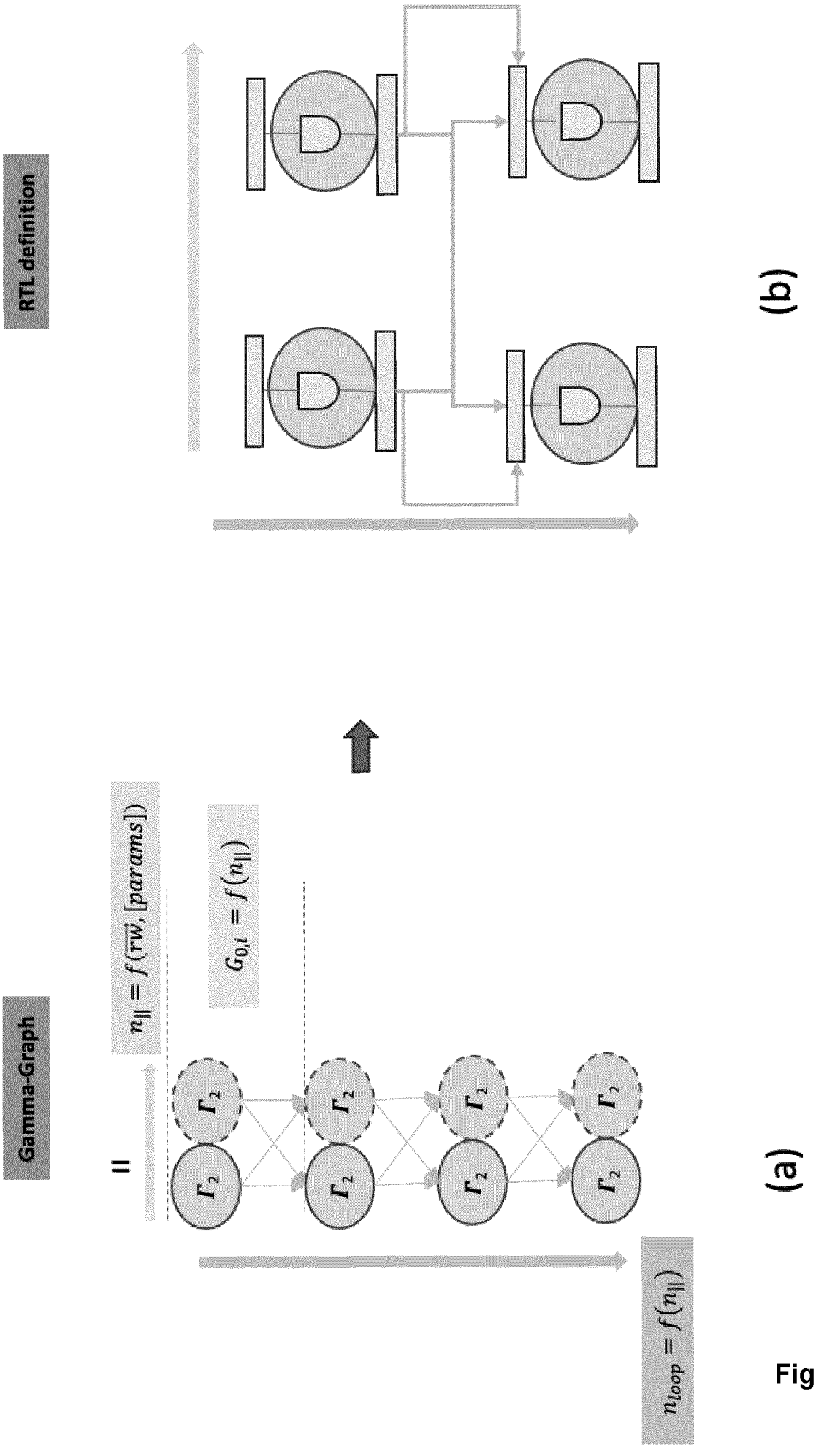
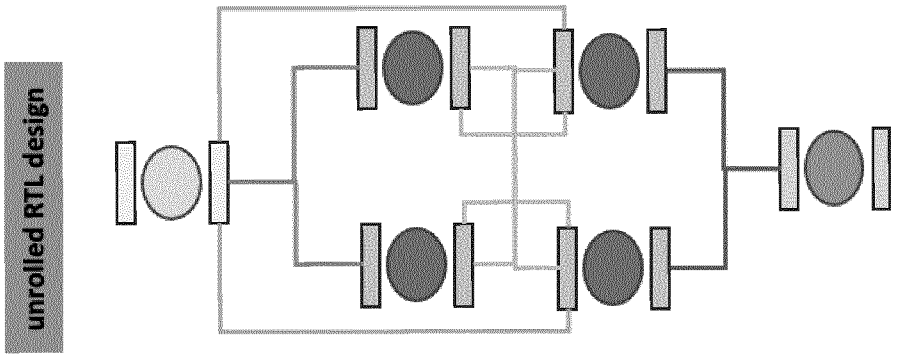
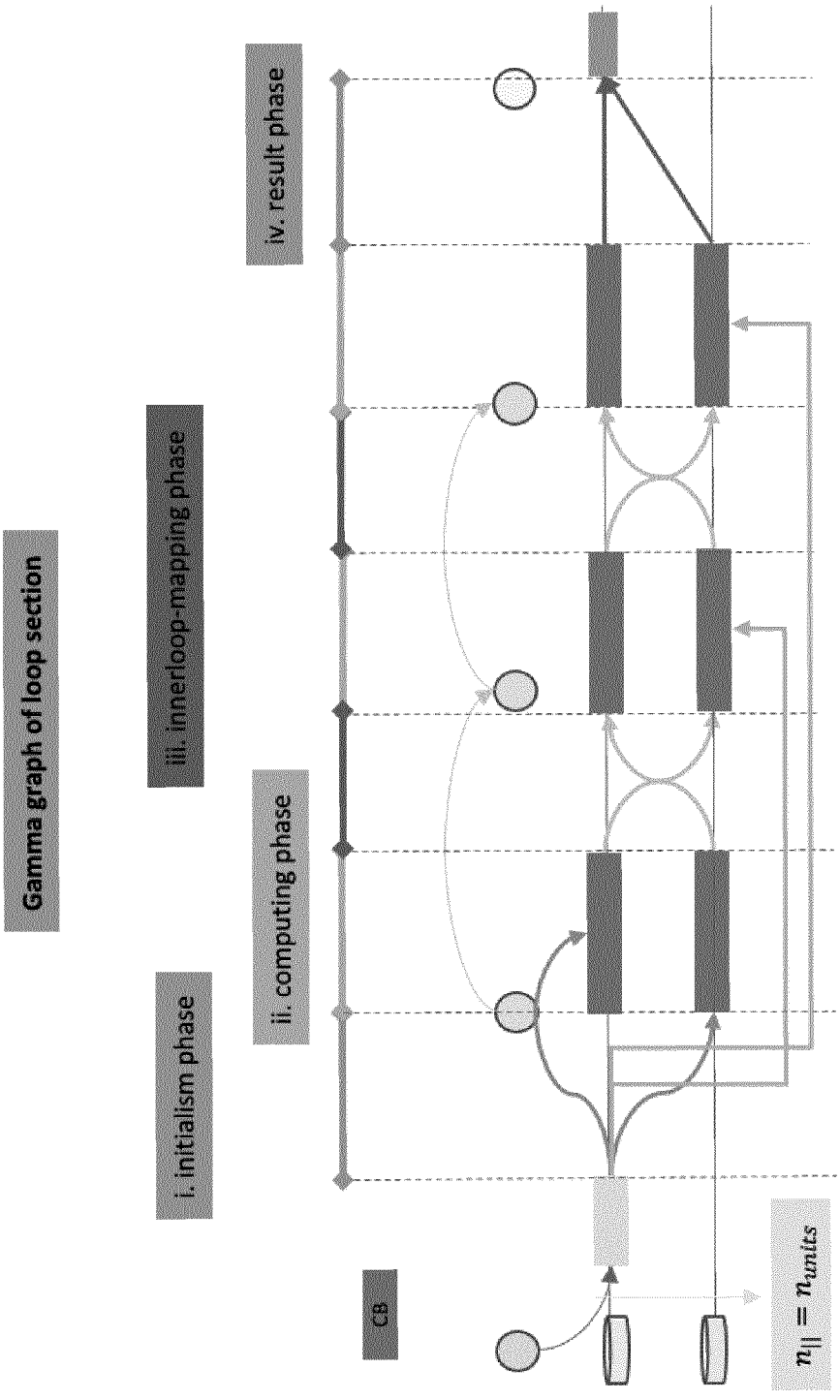

RTL definition
(b)
Gamma-Graph
$n_{||} = f(\overrightarrow{rw}, [params])$
$G_{0,i} = f(n_{||})$
Γ_2
$n_{loop} = f(n_{||})$
(a)

Fig. 134

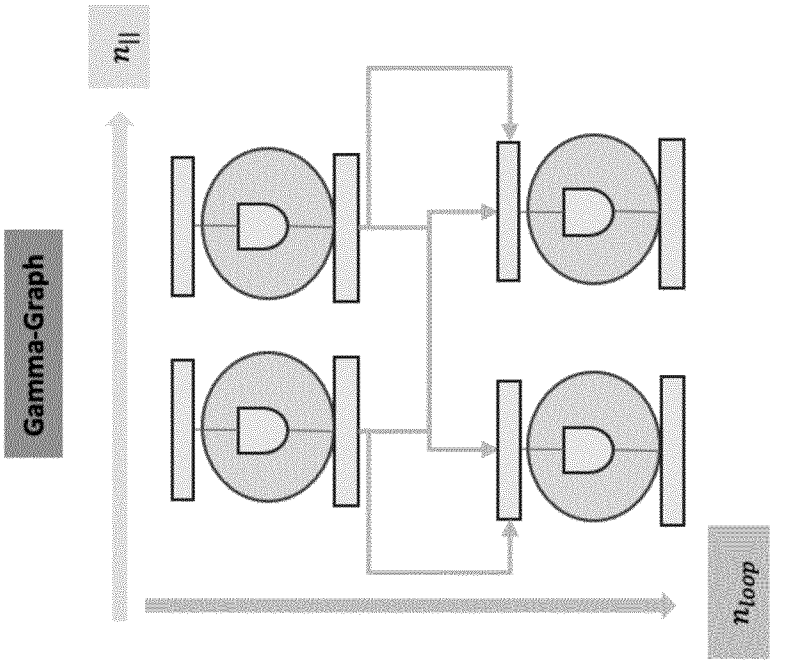

unrolled RTL design

(b)

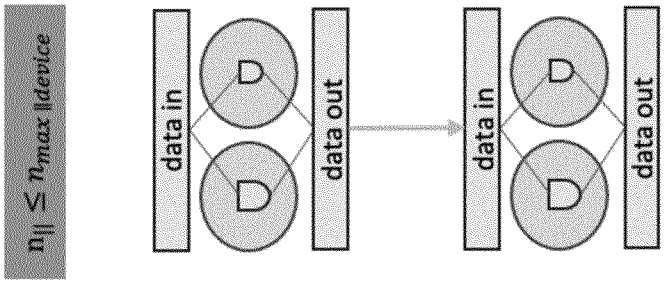

Gamma graph of loop section
i. Initialism phase
ii. computing phase
iii. innerloop-mapping phase
iv. result phase
CB
$n_{||} = n_{units}$

(a)

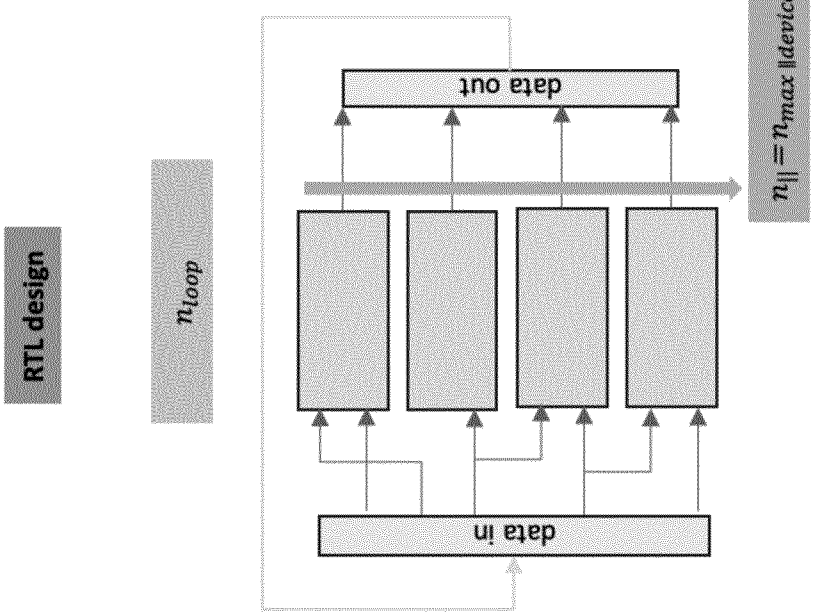
Fig. 135


Gamma-Graph
$n_{||}$
n_{loop}

(a)


$n_{||} \leq n_{max\ ||device}$
data in
data out
data in
data out

(b)


RTL design
n_{loop}
data out
data in
$n_{||} = n_{max\ ||device}$
instance

(c)

Fig. 136

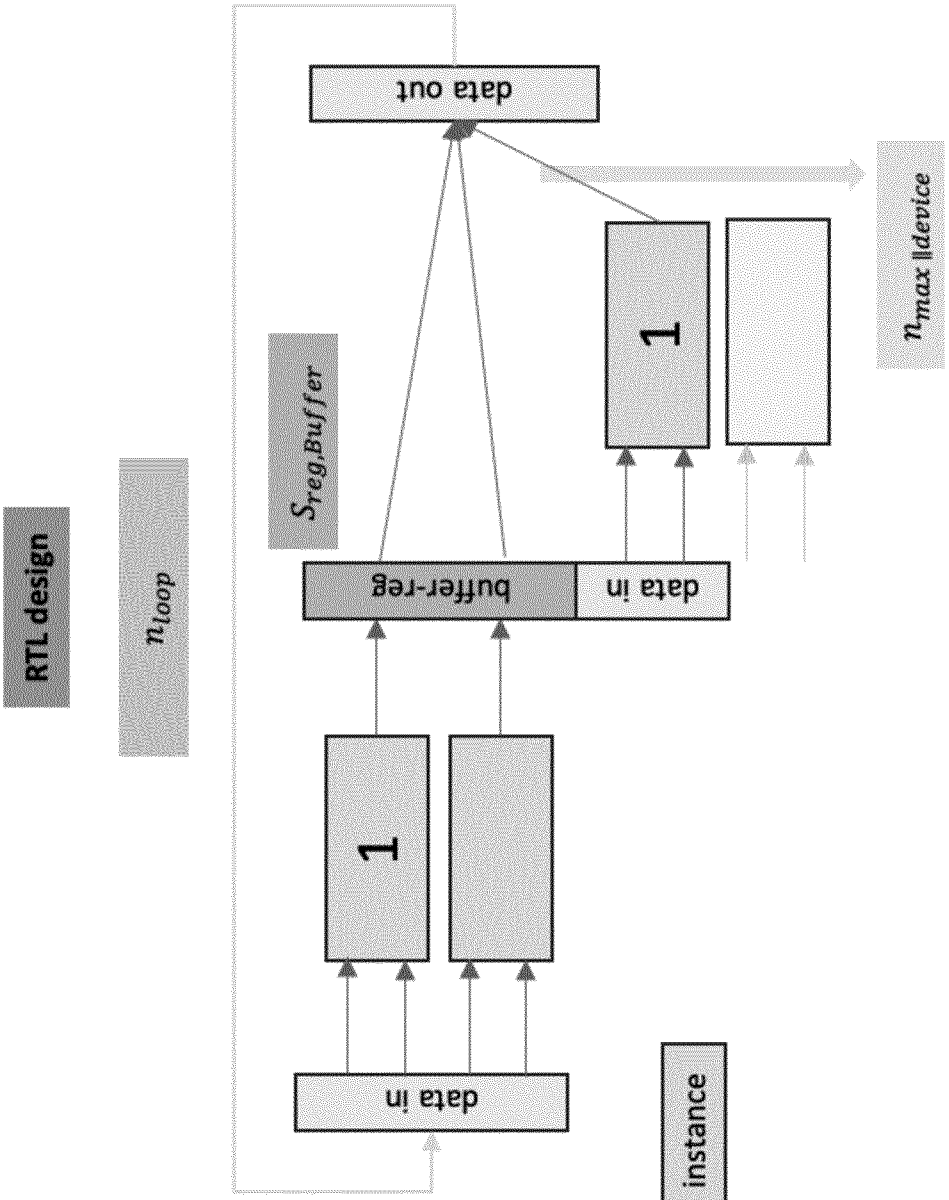

RTL design
n_{loop}
$S_{reg,Buffer}$
data out
$n_{max\ ||device}$
1
buffer-reg
data in
data in
instance
(b)
$n_{||} > n_{max\ ||device}$
$n_{||} = 3$
data in
data out
data in
data out
data in
data out
data in
data out
(a)

Fig. 137

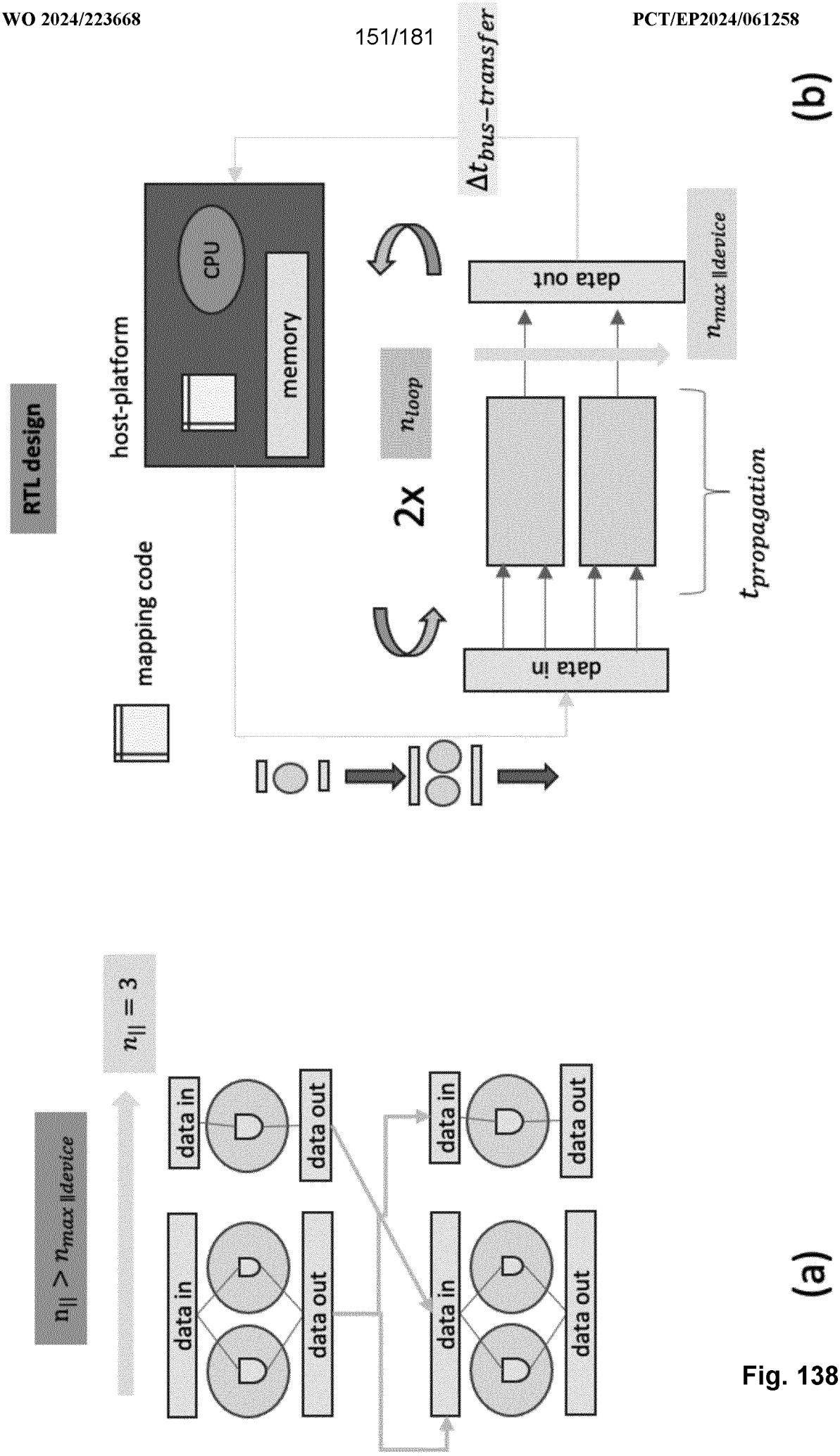

(b)
RTL design
host-platform
CPU
memory
mapping code
$\Delta t_{bus-transfer}$
n_{loop}
2x
data out
data in
$n_{max\ ||device}$
$t_{propagation}$
(a)
$n_{||} > n_{max\ ||device}$
$n_{||} = 3$
data in
data out

Fig. 138

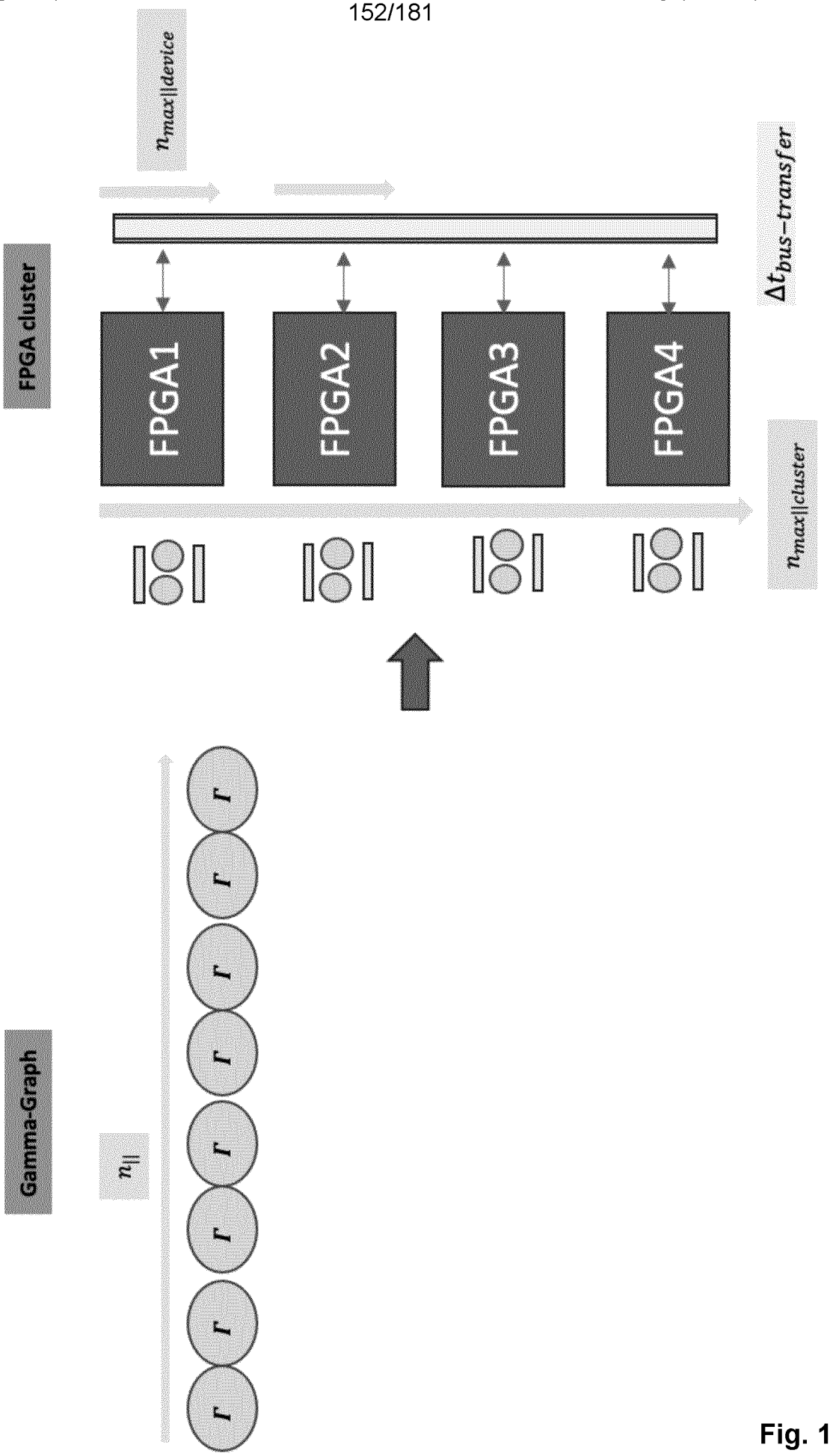

$n_{max||device}$
$\Delta t_{bus-transfer}$
FPGA cluster
FPGA1
FPGA2
FPGA3
FPGA4
$n_{max||cluster}$
Gamma-Graph
$n_{||}$
Γ

Fig. 139

```
flipflop #(NBDN, NIN, NOUT) flipflop0 (
    .clk(CLK),
    .copy_flag(copy1_flag),
    .in_memory({}),
    .ff_in({}),
    .ff_out({}),
    .copy_state(copy1_state)
    );
```

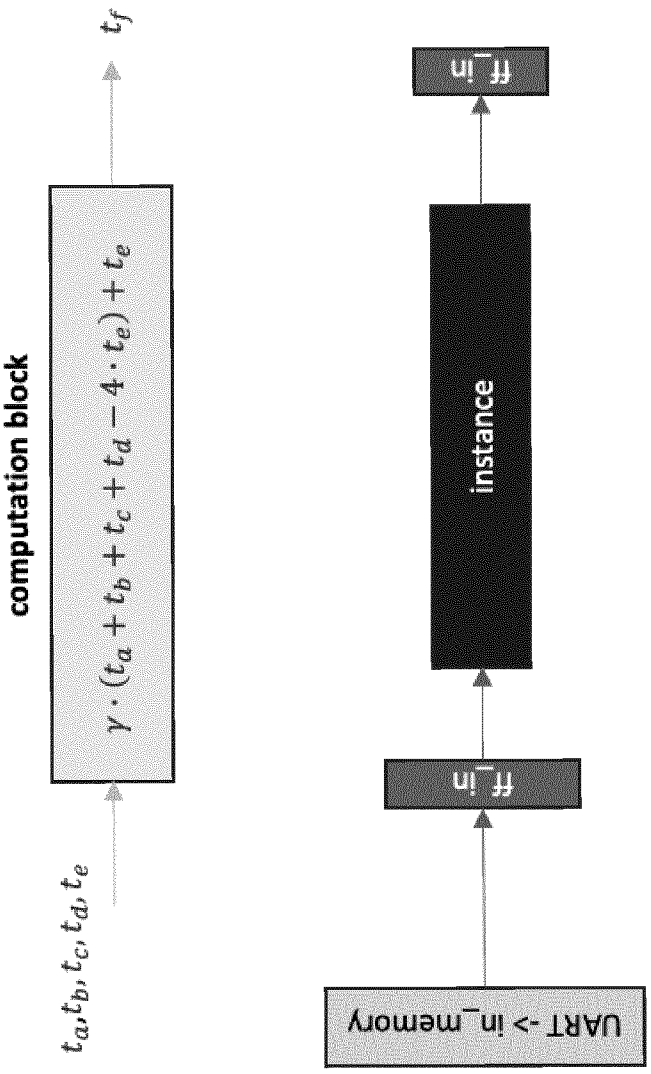
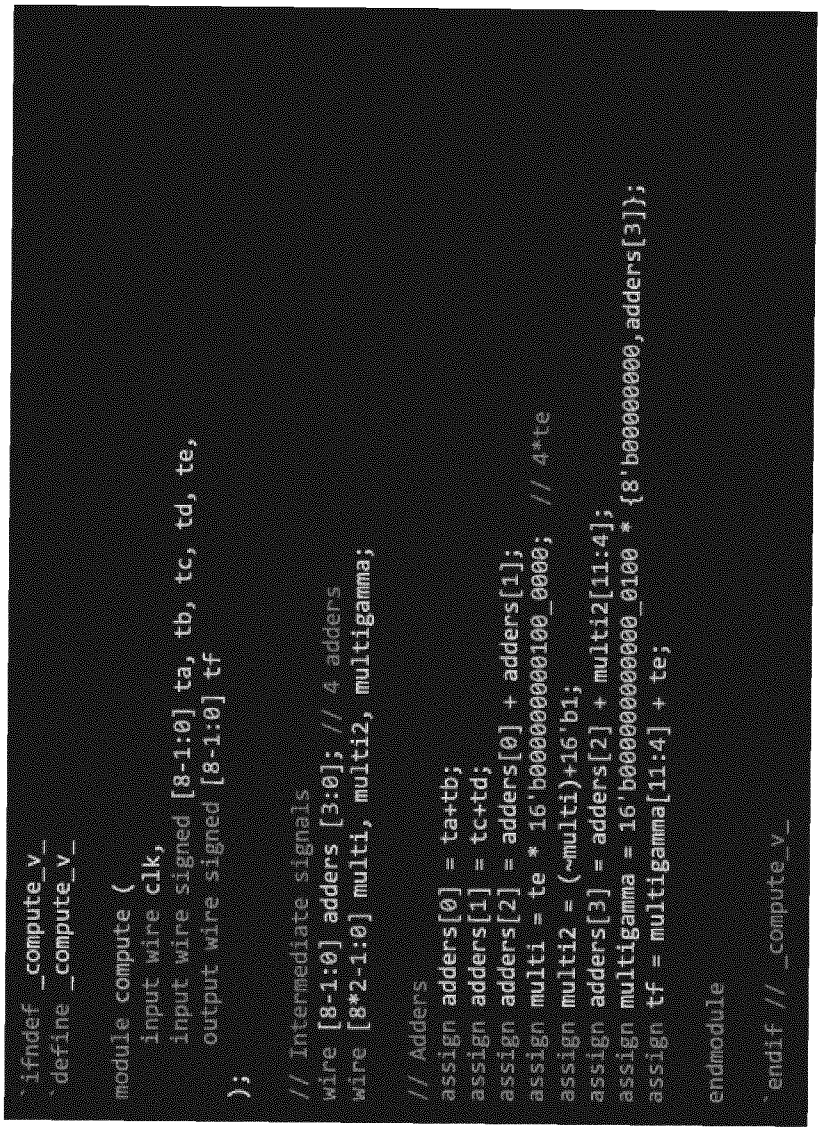


Fig. 140



```
`ifndef _compute_v_
`define _compute_v_
module compute (
    input wire clk,
    input wire signed [8-1:0] ta, tb, tc, td, te,
    output wire signed [8-1:0] tf
);
// Intermediate signals
wire [8-1:0] adders [3:0]; // 4 adders
wire [8*2-1:0] multi, multi2, multigamma;

// Adders
assign adders[0] = ta+tb;
assign adders[1] = tc+td;
assign adders[2] = adders[0] + adders[1];
assign multi = te * 16'b0000000000100_0000; // 4*te
assign multi2 = (~multi)+16'b1;
assign adders[3] = adders[2] + multi2[11:4];
assign multigamma = 16'b0000000000000_0100 * {8'b00000000,adders[3]};
assign tf = multigamma[11:4] + te;

endmodule

`endif // _compute_v_
```

$\gamma \cdot (t_a + t_b + t_c + t_d - 4 \cdot t_e) + t_e$

instance

Fig. 141

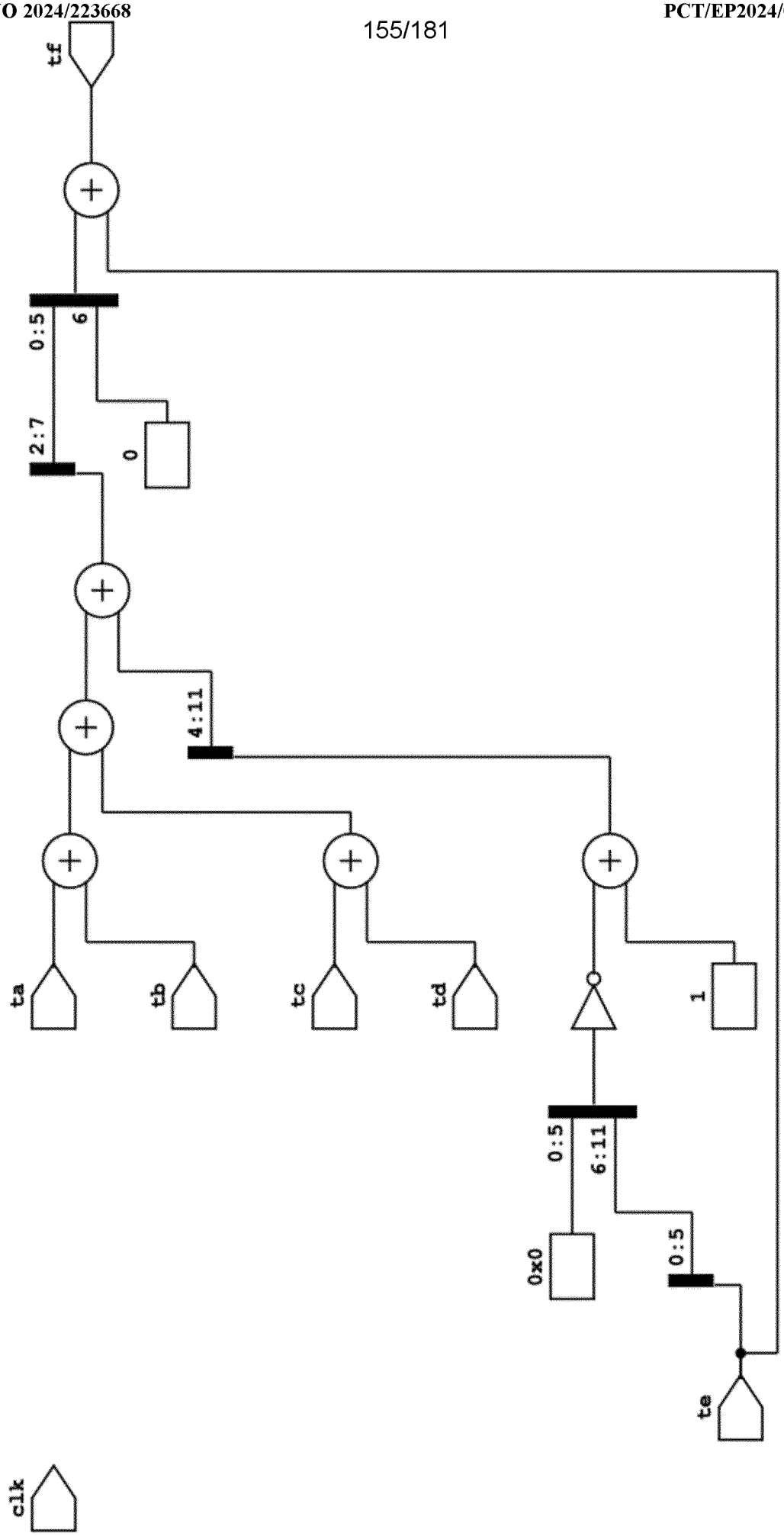


Fig. 142

```
wire signed [NBDN*8-1:0]
tf_comp0;
compute #(NBDN) comp0 (
  .clk(CLK),
  .ta(ff1_out[6]),
  .tb(in_memory[0]),
  .tc(ff1_out[1]),
  .td(in_memory[6]),
  .te(ff1_out[0]),
  .tf(tf_comp0)
  );
```

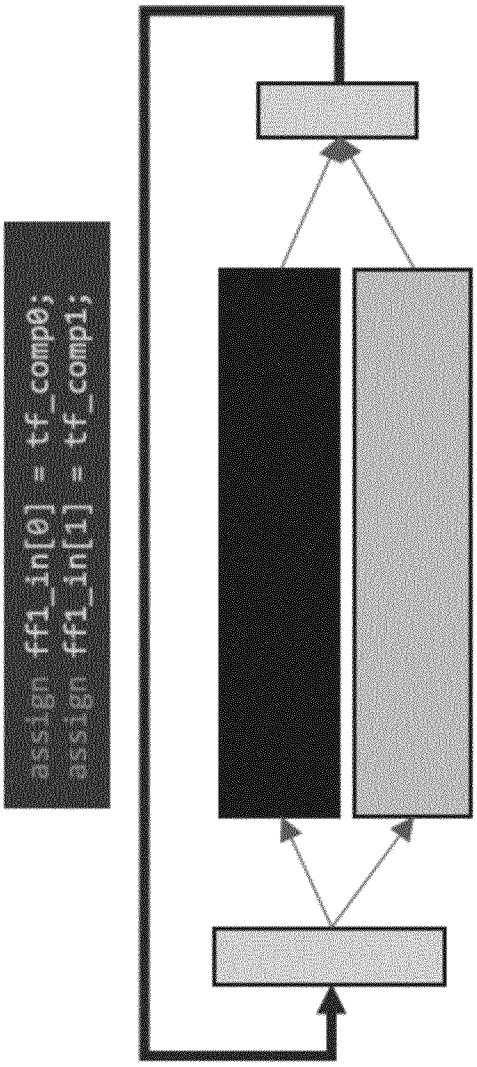


Fig. 143

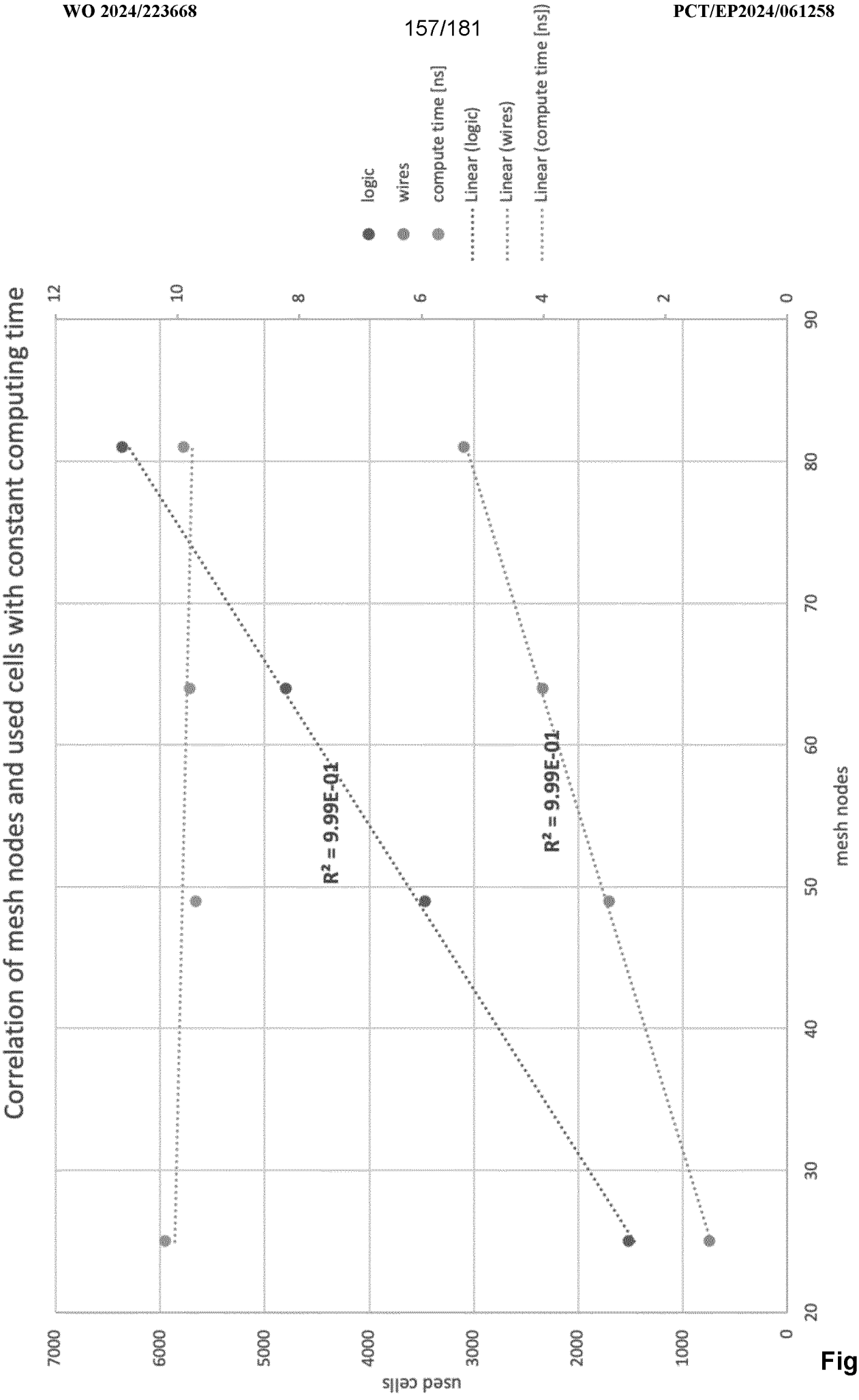

Correlation of mesh nodes and used cells with constant computing time
logic
wires
compute time [ns]
Linear (logic)
Linear (wires)
Linear (compute time [ns])
R² = 9.99E-01
R² = 9.99E-01
used cells
mesh nodes
0
1000
2000
3000
4000
5000
6000
7000
20
30
40
50
60
70
80
90
0
2
4
6
8
10
12

Fig. 144

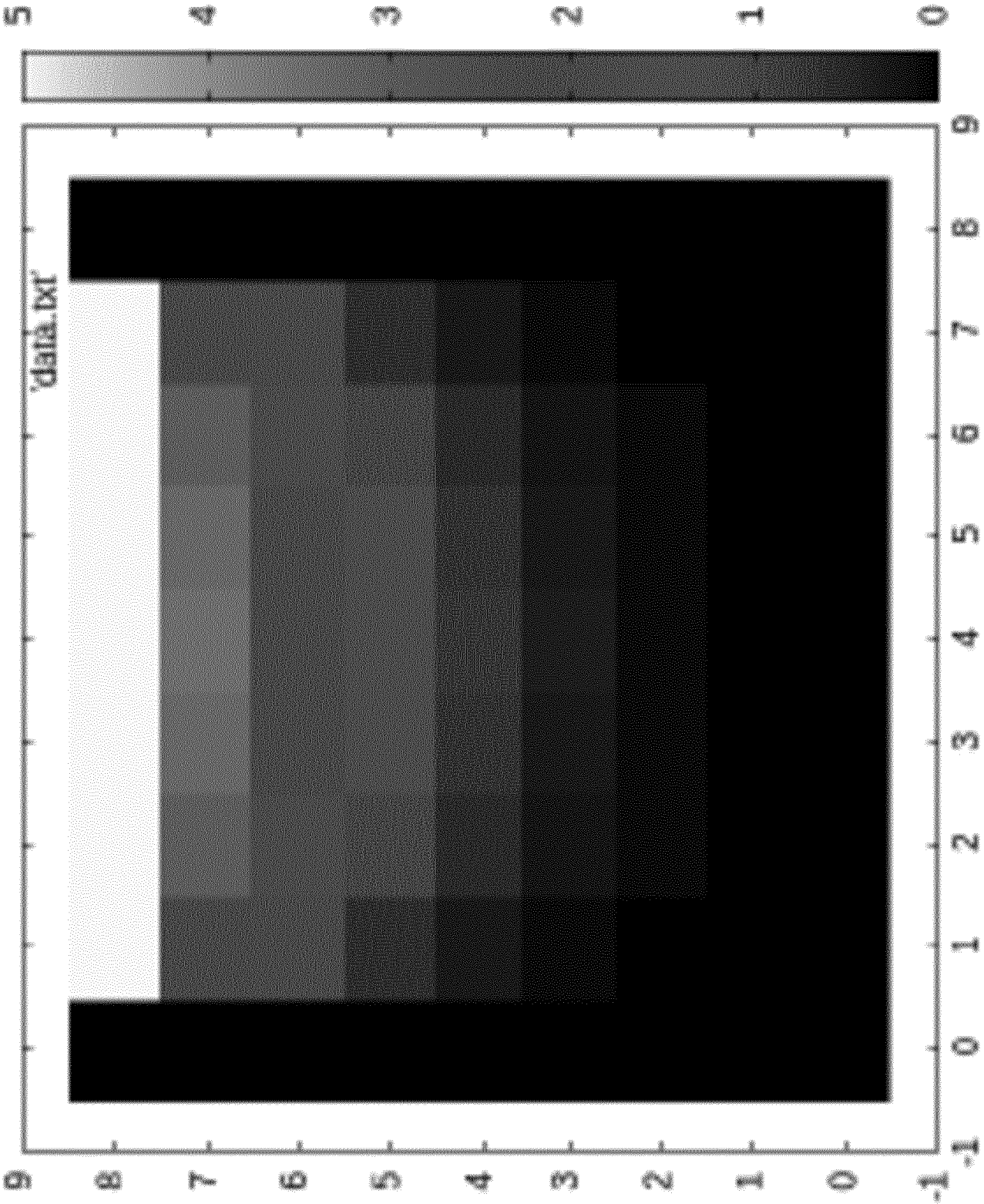


Fig. 145

Field	35 or 43	rs	Rt	Address
Bit-position	31:26	25:21	20:16	15:0
		$t2	$t1	Offset

(a)

lw, $t1, offset($t2)

RegWrite -> register number

address

data memory

registers

Write data

control

Instruction [20-16]
$t1

Instruction [31-26]

Instruction [25-21]
$t2

Instruction [15-0]

Instruction

Signed-extend

5-step pipeline

pipeline

IF ID EX MEM WB

2 cycles

(b)

IF/ID ID/EX EX/MEM MEM/WB

PC IM registers S_{reg} mem

Fig. 146

Field	0	rs	rt	rd	shamt	Funct
Bit-position	31:26	25:21	20:16	15:11	10:6	5:0
		$t2	$t3	$t1, dest		

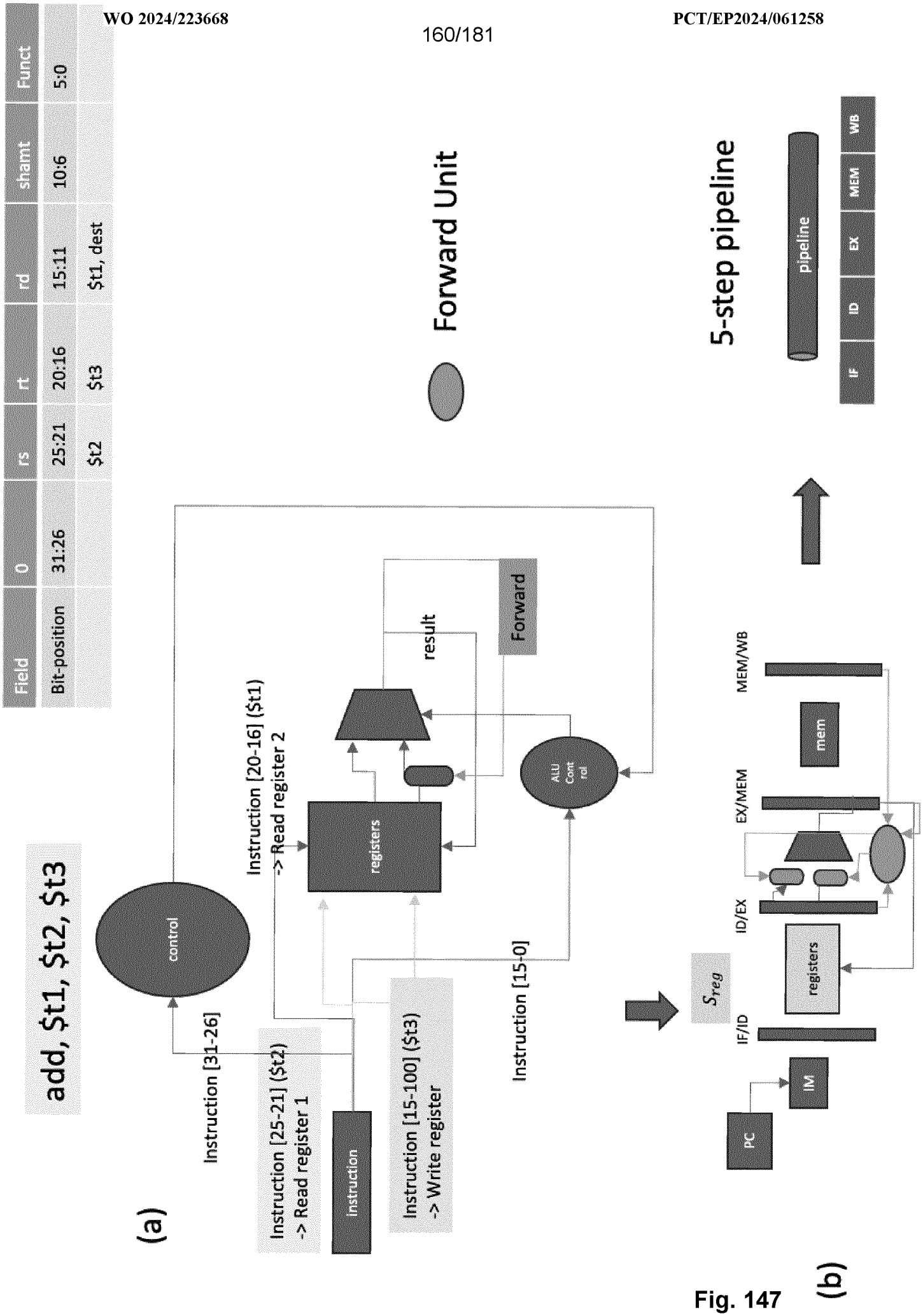


Fig. 147

$$S_{reg,stage,in,\Gamma 2} = \max_{\Gamma 2a,2b} n_{datanode,CB} \cdot S_{data\ type}$$

$$S_{reg,stage,out,\Gamma 2} = S_{data\ type,2a} + S_{data\ type,2b}$$

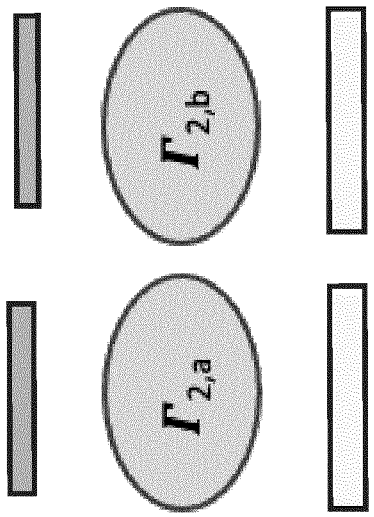


Fig. 148

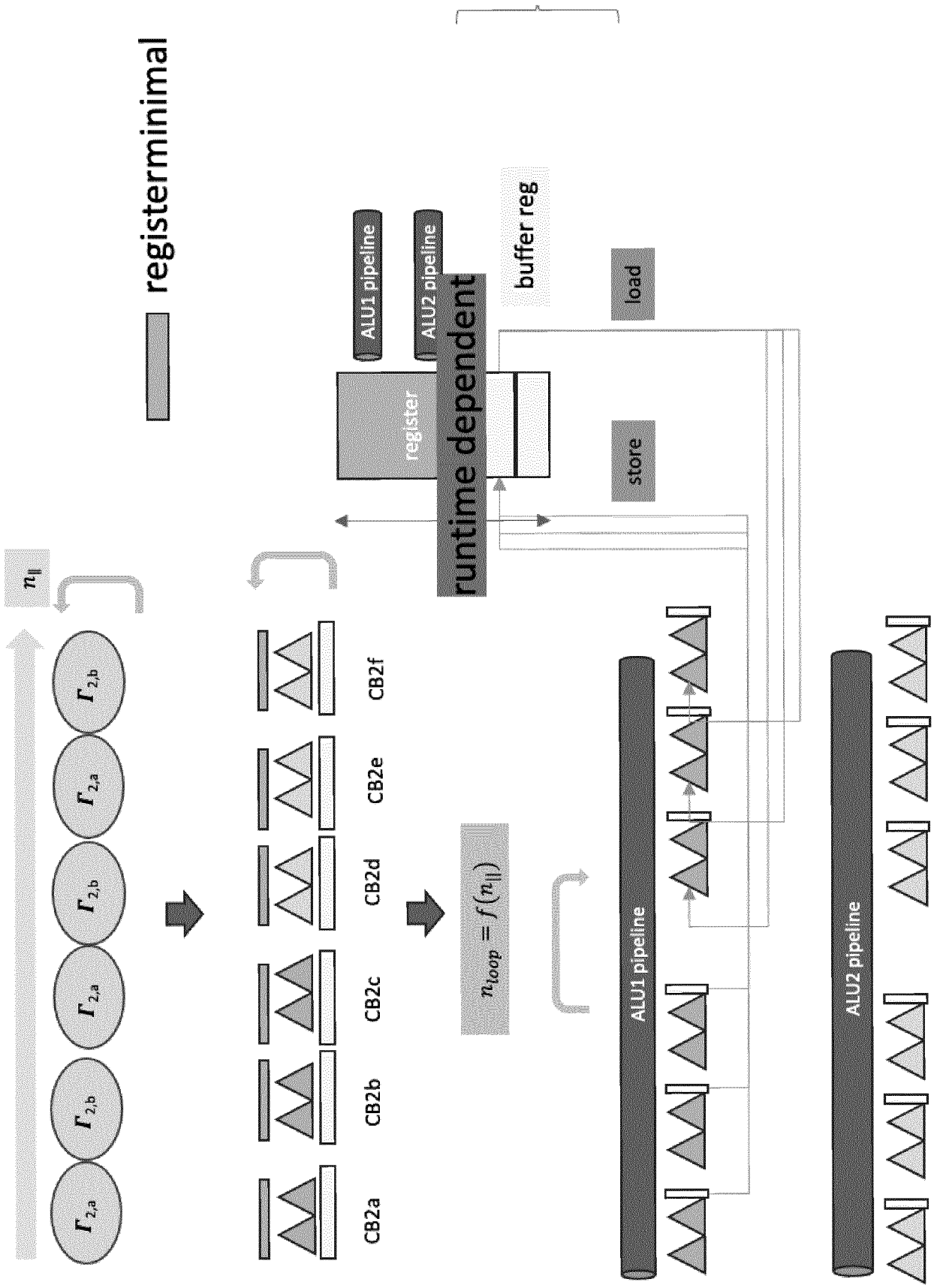

registerminimal
$n_{\|}$
$\Gamma_{2,a}$
$\Gamma_{2,b}$
$\Gamma_{2,b}$
$\Gamma_{2,a}$
$\Gamma_{2,b}$
$\Gamma_{2,a}$
CB2a
CB2b
CB2c
CB2d
CB2e
CB2f
$n_{loop} = f(n_{\|})$
ALU1 pipeline
ALU2 pipeline
register
runtime dependent
buffer reg
load
store
ALU1 pipeline
ALU2 pipeline

Fig. 149

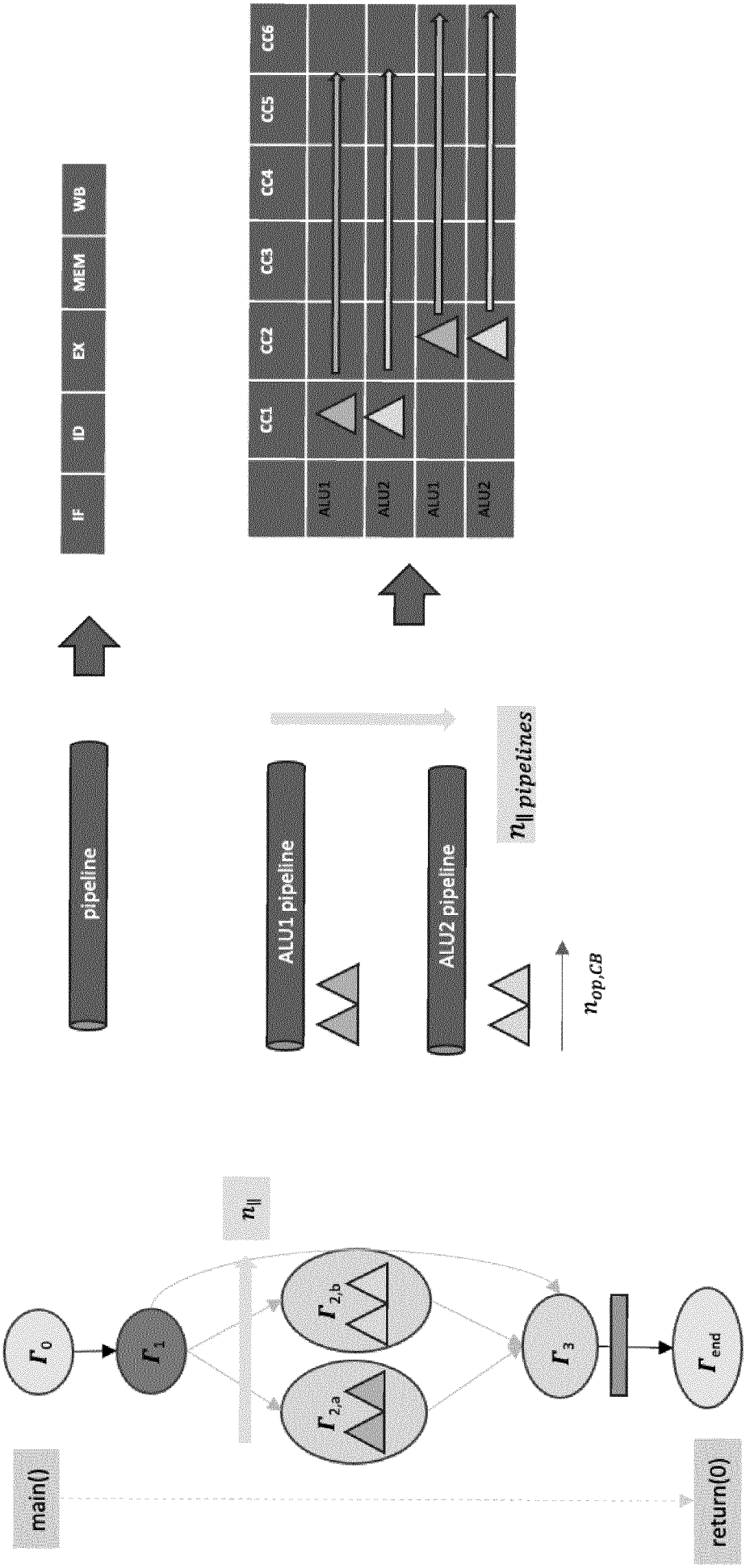

IF
ID
EX
MEM
WB
CC1
CC2
CC3
CC4
CC5
CC6
ALU1
ALU2
ALU1
ALU2
pipeline
ALU1 pipeline
ALU2 pipeline
$n_{\parallel}$ pipelines
$n_{op,CB}$
$n_{\parallel}$
Γ_0
Γ_1
$\Gamma_{2,b}$
$\Gamma_{2,a}$
Γ_3
Γ_{end}
main()
return(0)

Fig. 150

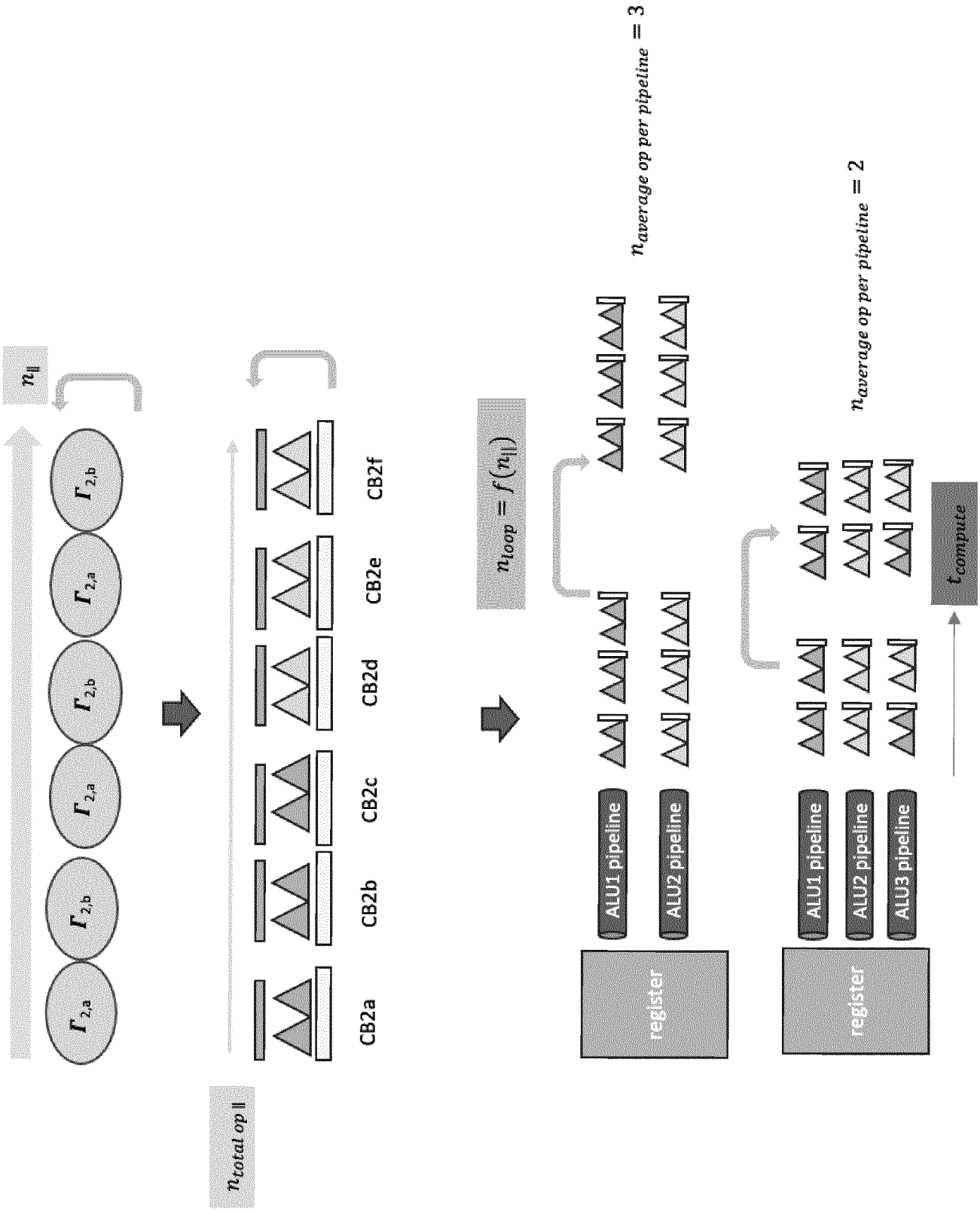
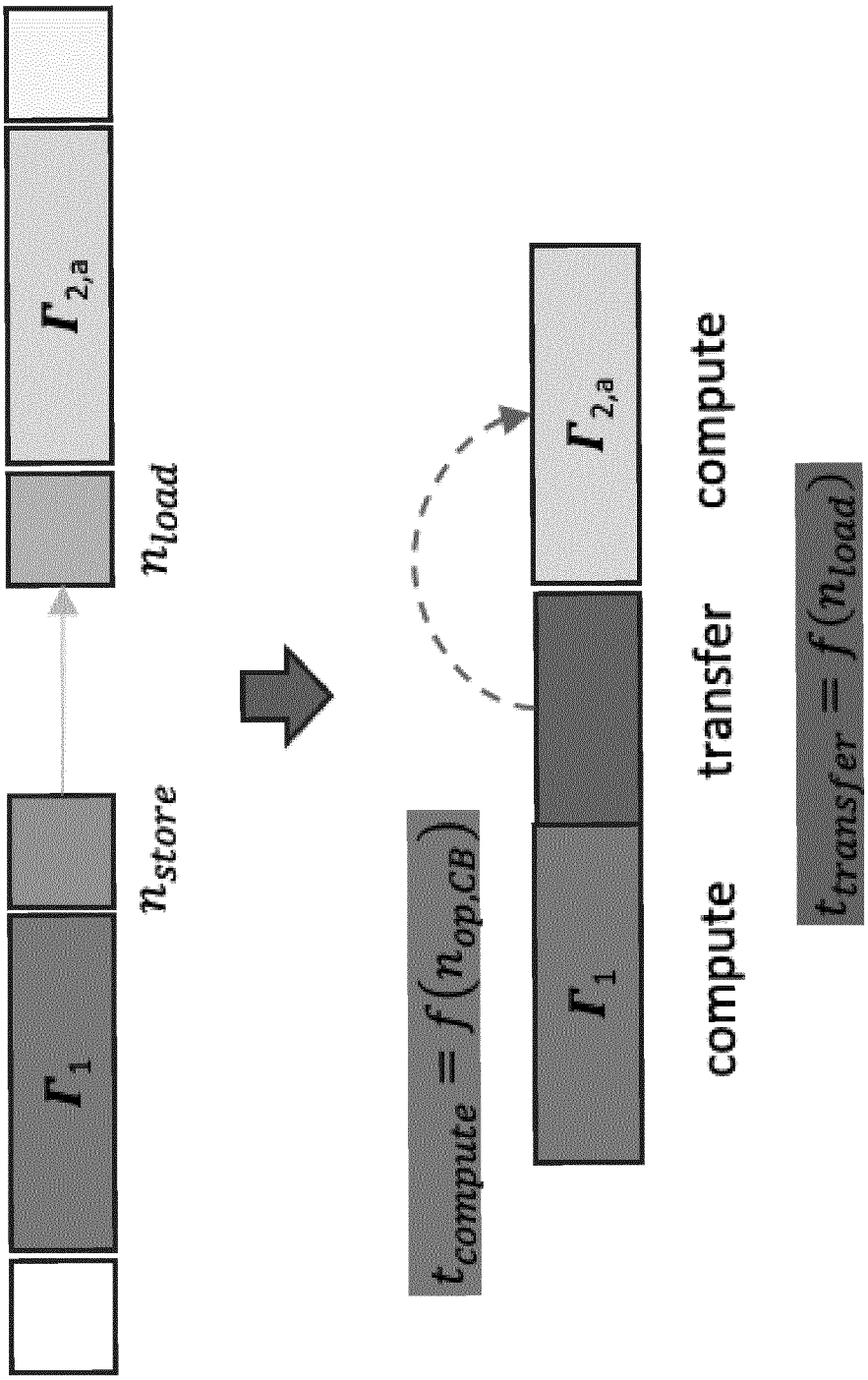

$n_{||}$
$\Gamma_{2,a}$
$\Gamma_{2,b}$
$\Gamma_{2,b}$
$\Gamma_{2,a}$
$\Gamma_{2,b}$
$\Gamma_{2,a}$
$n_{total\ op\ ||}$
CB2a
CB2b
CB2c
CB2d
CB2e
CB2f
$n_{loop} = f(n_{||})$
$n_{average\ op\ per\ pipeline} = 3$
$n_{average\ op\ per\ pipeline} = 2$
ALU1 pipeline
ALU2 pipeline
ALU3 pipeline
register
$t_{compute}$

Fig. 151


$\Gamma_{2,a}$
n_{load}
n_{store}
Γ_1
$t_{compute} = f(n_{op,CB})$
$\Gamma_{2,a}$
Γ_1
compute
transfer
compute
$t_{transfer} = f(n_{load})$

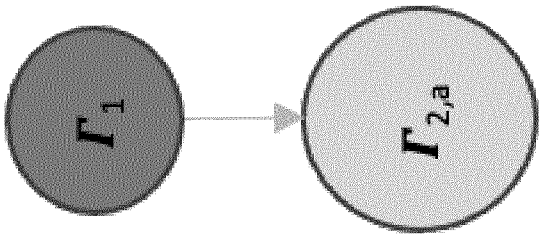

Γ_1
$\Gamma_{2,a}$

Fig. 152

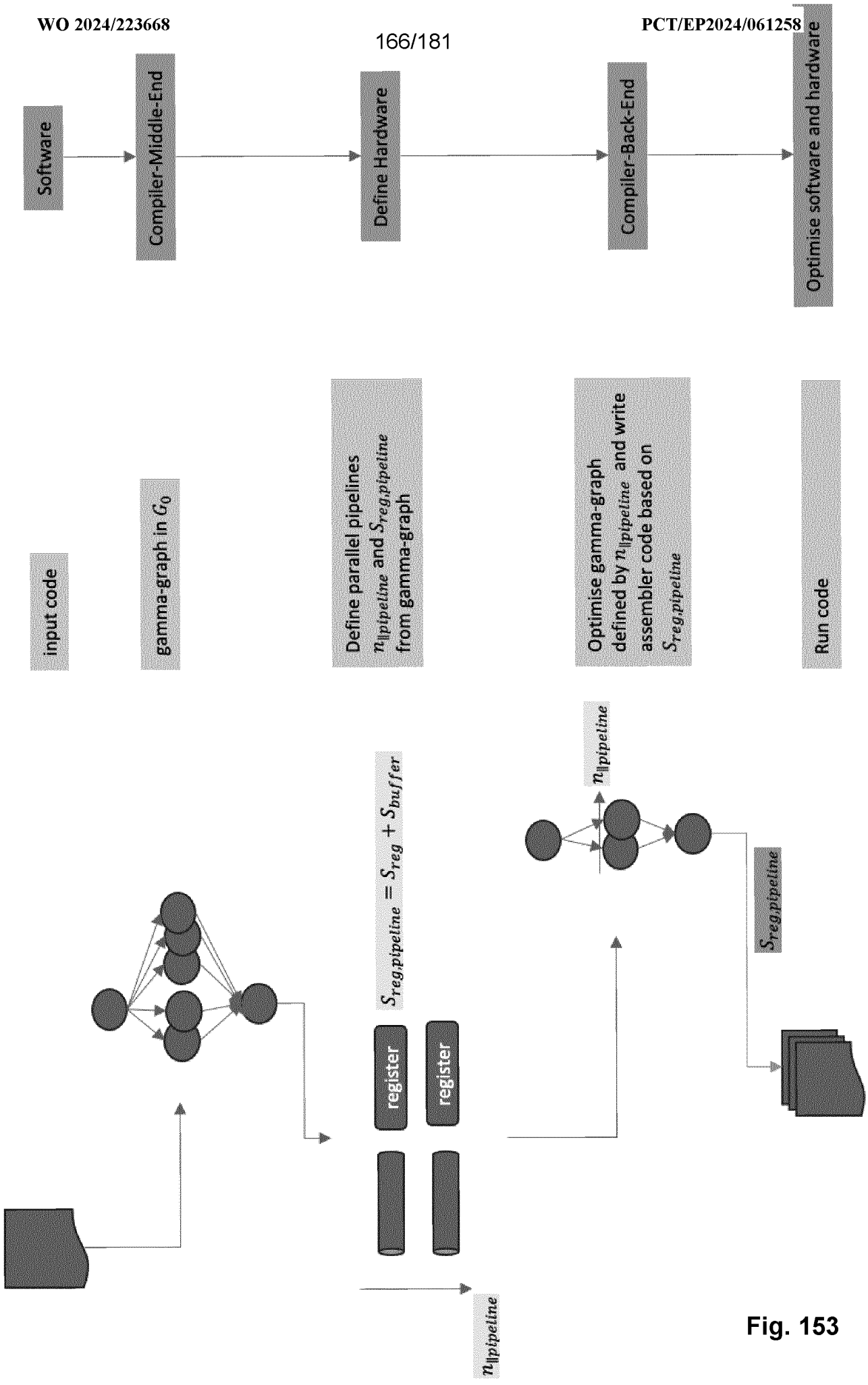

Software
Compiler-Middle-End
Define Hardware
Compiler-Back-End
Optimise software and hardware
input code
gamma-graph in G_0
Define parallel pipelines $n_{\|pipeline}$ and $S_{reg,pipeline}$ from gamma-graph
Optimise gamma-graph defined by $n_{\|pipeline}$ and write assembler code based on $S_{reg,pipeline}$
Run code
$S_{reg,pipeline} = S_{reg} + S_{buffer}$
register
register
$n_{\|pipeline}$
$n_{\|pipeline}$
$S_{reg,pipeline}$

Fig. 153

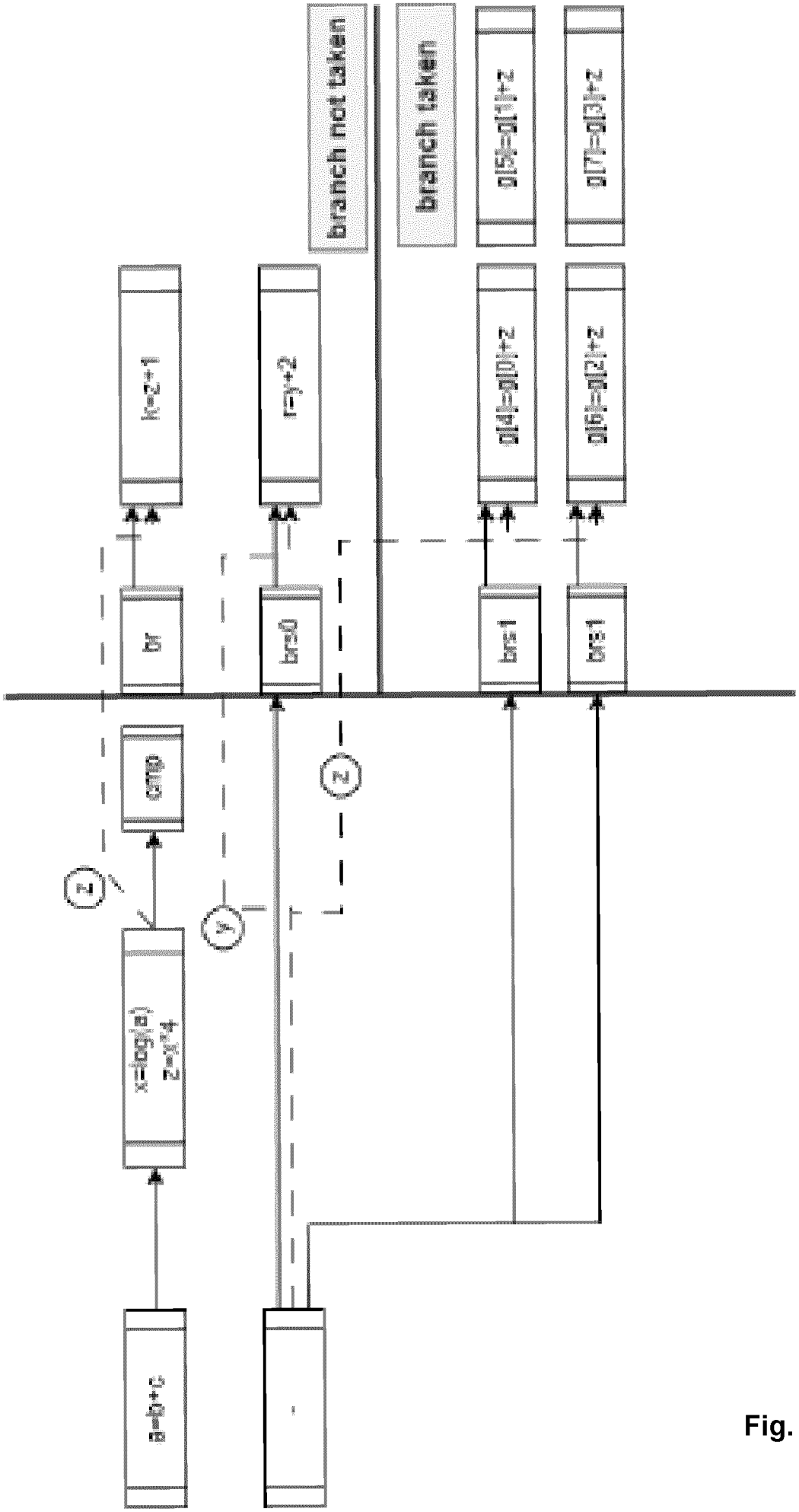

branch not taken
branch taken
p15=p11+2
p17=p13+2
k=2-x
n=x-2
p14=p10+2
p16=p12+2
br0
br1
br1
gto
x=log(a)/log(2)
a+b=c
a
b
c
2
1

Fig. 154

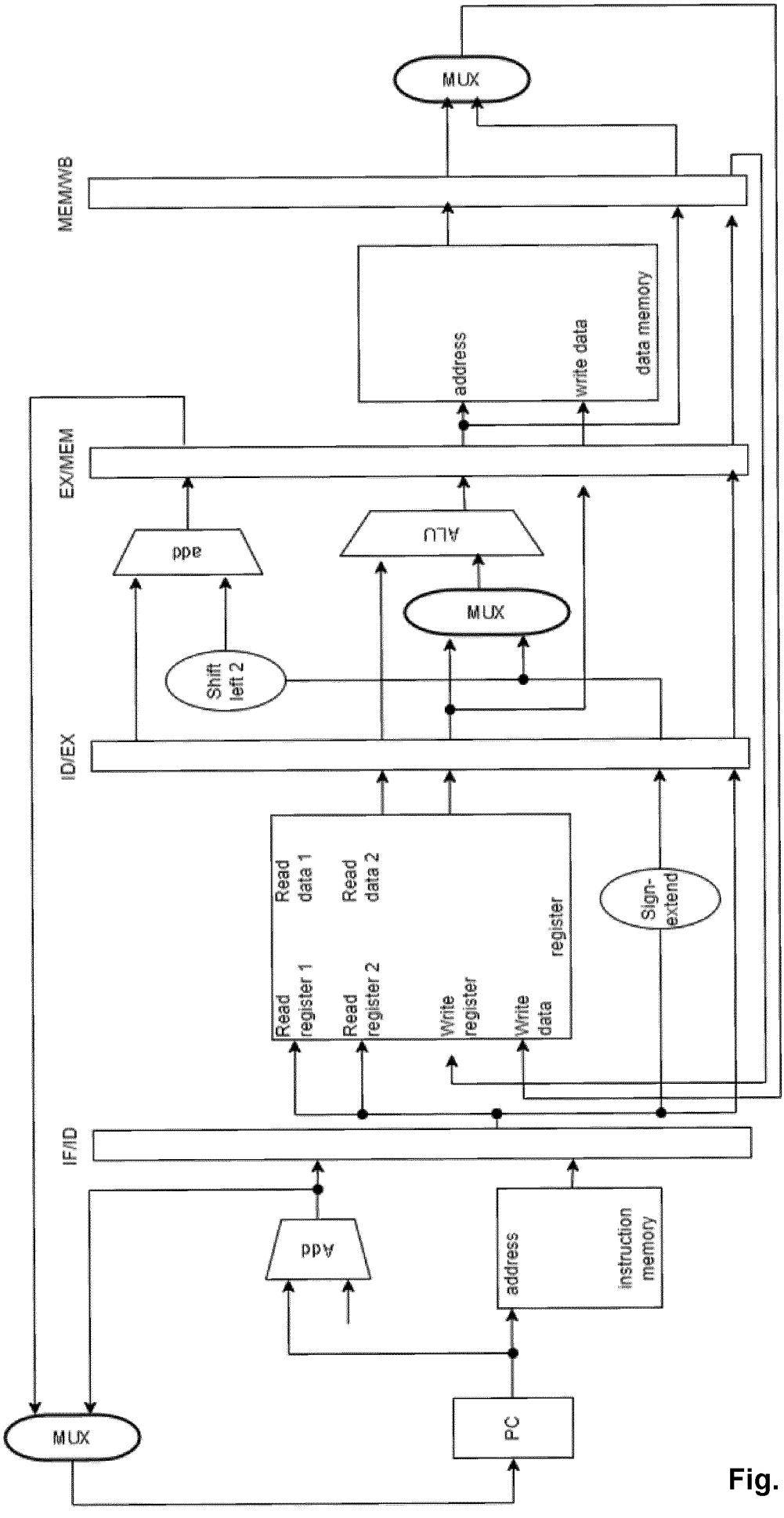

MUX
MEM/WB
address
write data
data memory
EX/MEM
add
ALU
MUX
Shift left 2
ID/EX
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
register
Sign-extend
IF/ID
Add
address
instruction memory
PC
MUX

Fig. 155

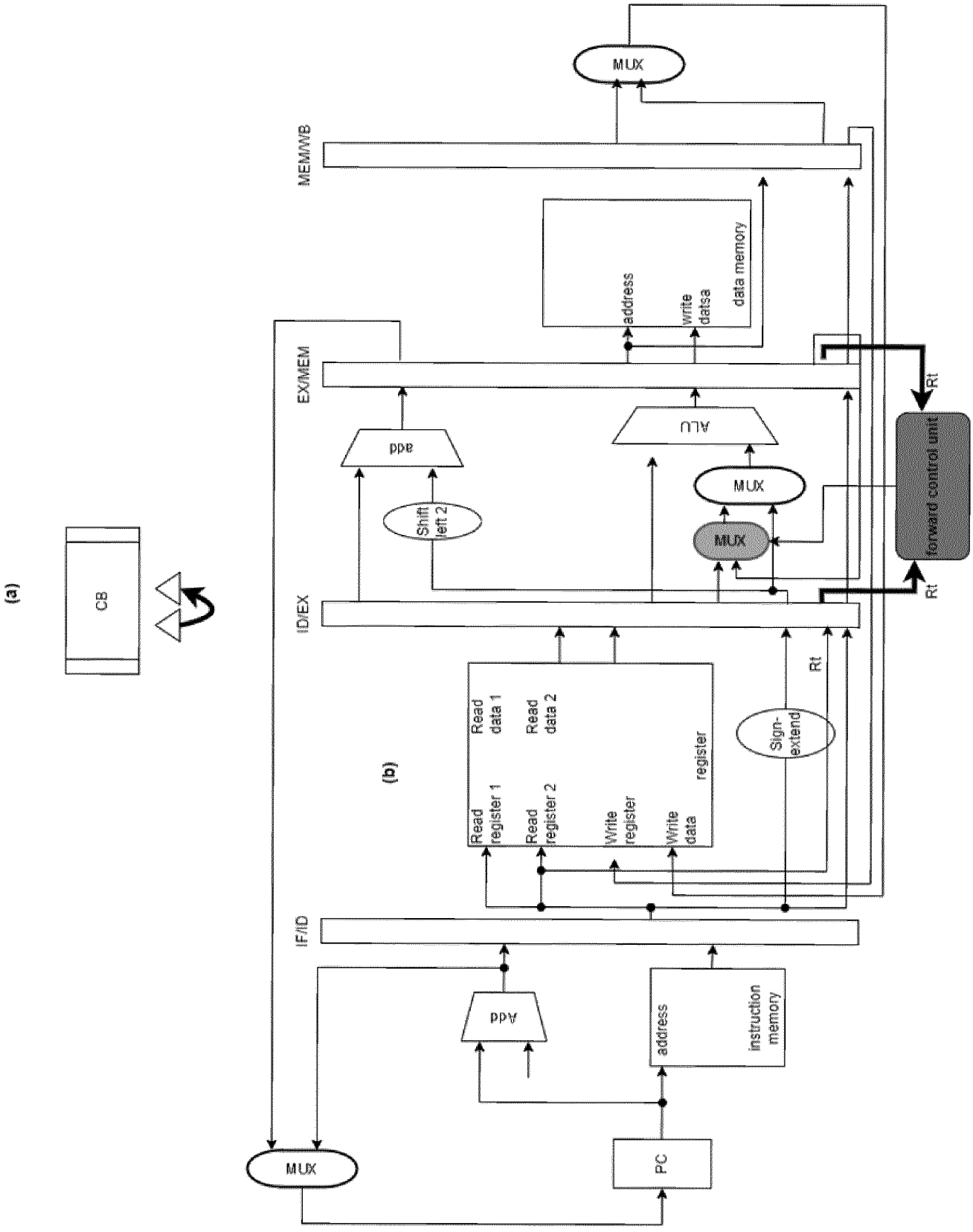

(a)
CB
(b)
MUX
MEM/WB
address
write datsa
data memory
EX/MEM
ALU
add
Shift left 2
MUX
MUX
forward control unit
Rt
Rt
ID/EX
Rt
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
register
Sign-extend
IF/ID
add
address
instruction memory
PC
MUX

Fig. 156

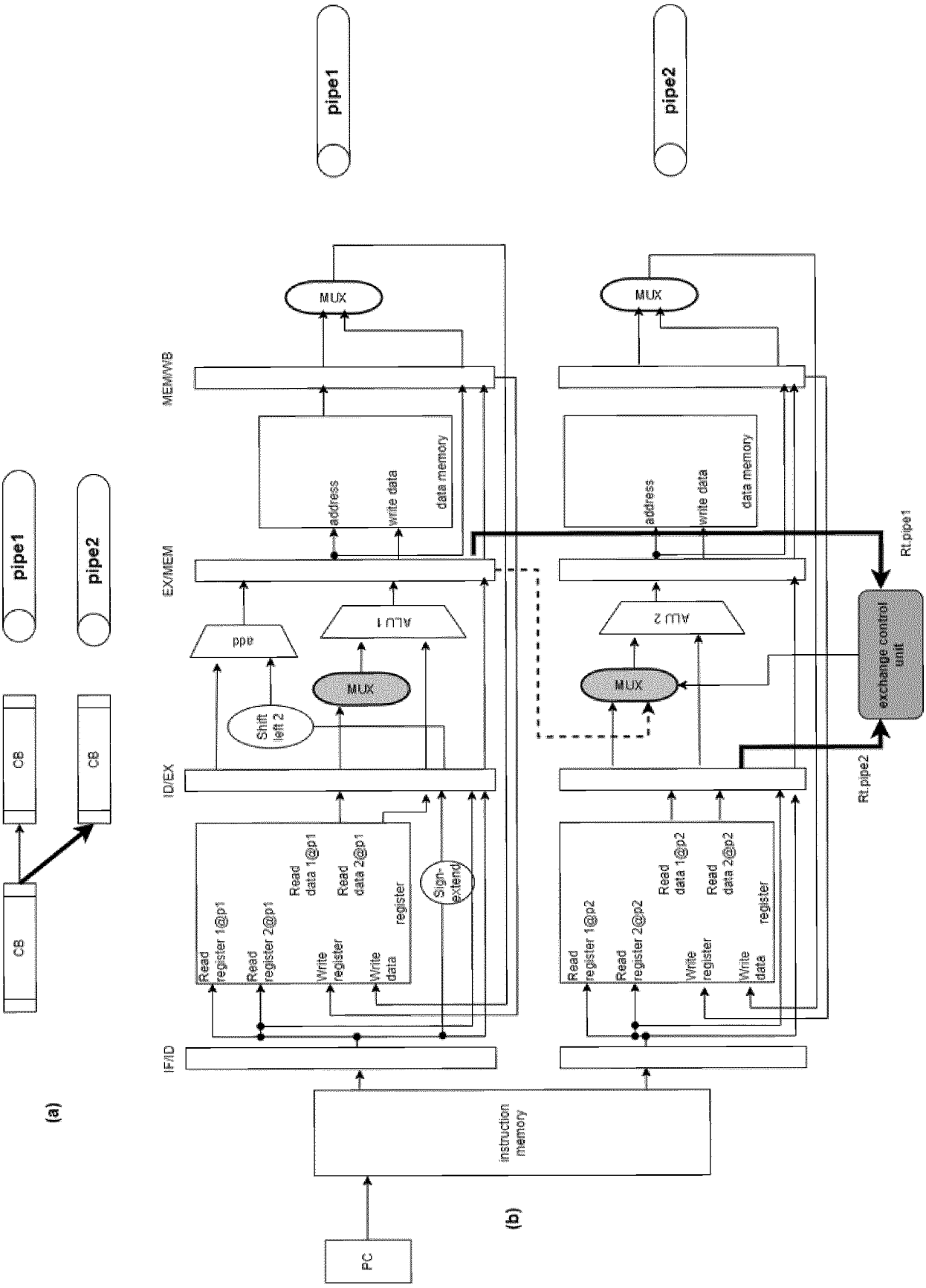

pipe1
pipe2
CB
CB
CB
(a)
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
instruction memory
Read register 1@p1
Read register 2@p1
Write register
Write data
Read data 1@p1
Read data 2@p1
register
Sign-extend
Shift left 2
add
MUX
ALU 1
address
write data
data memory
Read register 1@p2
Read register 2@p2
Write register
Write data
Read data 1@p2
Read data 2@p2
register
ALU 2
exchange control unit
Rt.pipe1
Rt.pipe2
(b)

Fig. 157

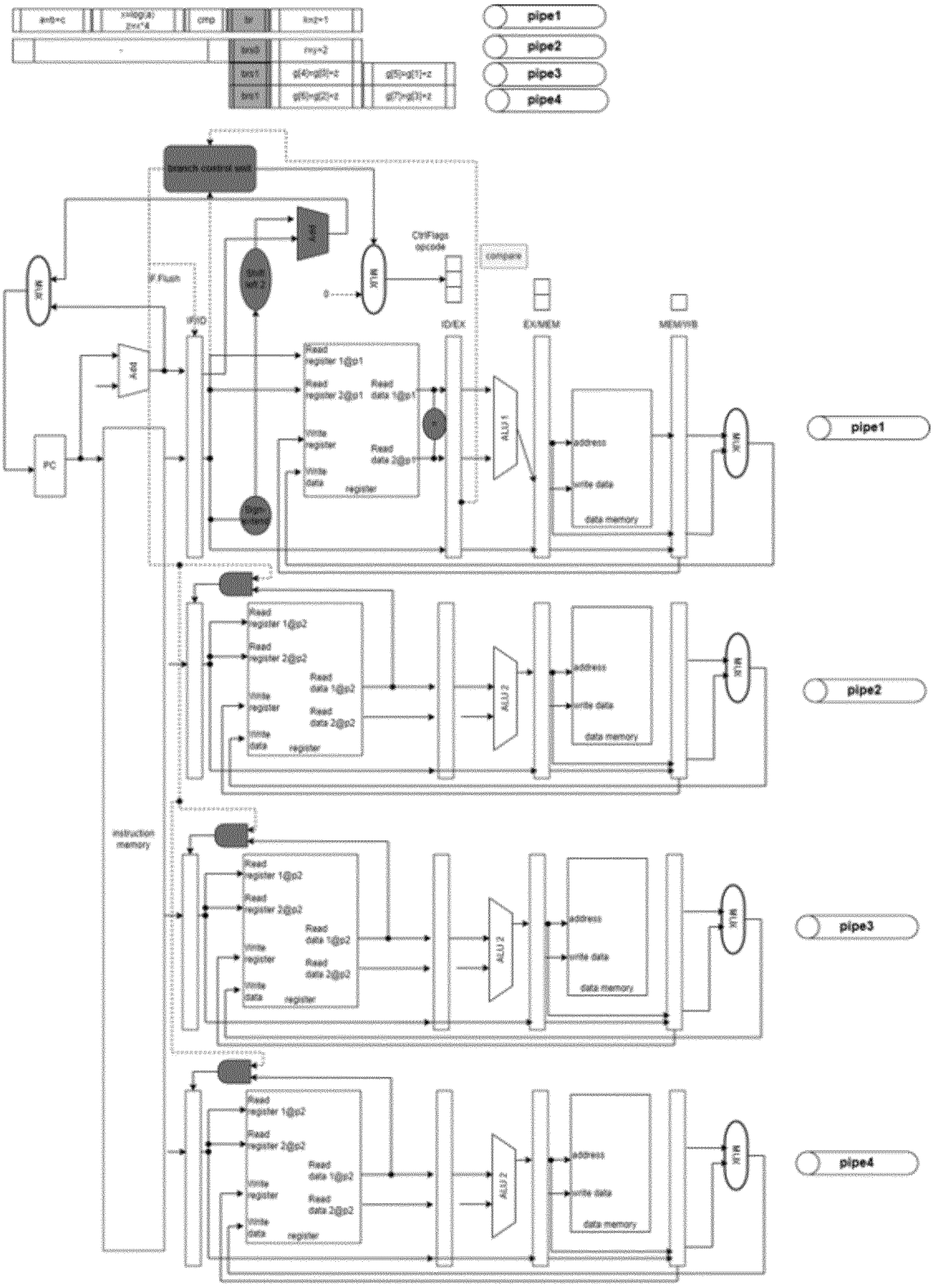

pipe1
pipe2
pipe3
pipe4
Branch control unit
IF Flush
IF/ID
ID/EX
EX/MEM
MEM/WB
ALU 1
ALU 2
PC
Instruction memory
Read register 1@p1
Read register 2@p1
Read data 1@p1
Read data 2@p1
Write register
Write data
register
data memory
address
write data
Read register 1@p2
Read register 2@p2
Read data 1@p2
Read data 2@p2

Fig. 158

$n_{CBpergamman} = \frac{n_{||}}{n_{units}} = \frac{16}{3} = (5,1)$
Gamma Graph with increased granularity
Γ_0
$\Gamma_{1,a}$
$\Gamma_{1,b}$
$\Gamma_{1,c}$
n_{loop}
$n_{||}$
$n_i = 4$
$n_j = 2$
$n_k = 2$
k->j->i
k=0
k=1
j=0
j=1
i=0
p1
p2
p3
p4
divmod(5,4)=(1,1)
divmod(10,4)=(2,2)
divmod(15,4)=(3,3)
divmod(5,8)=(0,5)
divmod(10,8)=(1,2)
divmod(15,8)=(1,7)
i=0-3
i=2-3
i=0-1
i=0
i=2-3
i=0-3
j=0
j=1
k=0
k=1
i
j
k
P1
P2
P3
P4
gbpindx
divmod(pn,i)=d1,d2
i,j = d2, j@d1
divmod(pn,j)=d3,d4
k
0
5
10
15
(1,1)
(2,2)
(3,3)
0,0
1,1
2,0
3,1
(0,5)
(1,2)
(1,7)
k=0
k=0
k=1
k=1

Fig. 159

```
#include "stdio.h"
#include "stdint.h"
#include "math.h"

#define N 8
int main(int argc, char *argv[]) {
    uint8_t a, b, c, x, y, z, r, k;
    int g[N];
    a = b + c;
    x = log(a);
    y = a*a;
    z = x*4;
    if (z < 5) {
        for (int i=0; i<N-4; i++) {
            g[i+4] = g[i]+z;
        }
        k = g[N-1]*g[N-2];
        r = y+4;
    } else {
        r = y+2;
        k = z+1;
    }
    printf("%d %d", r, k);
return(1);
}
```

Fig. 160

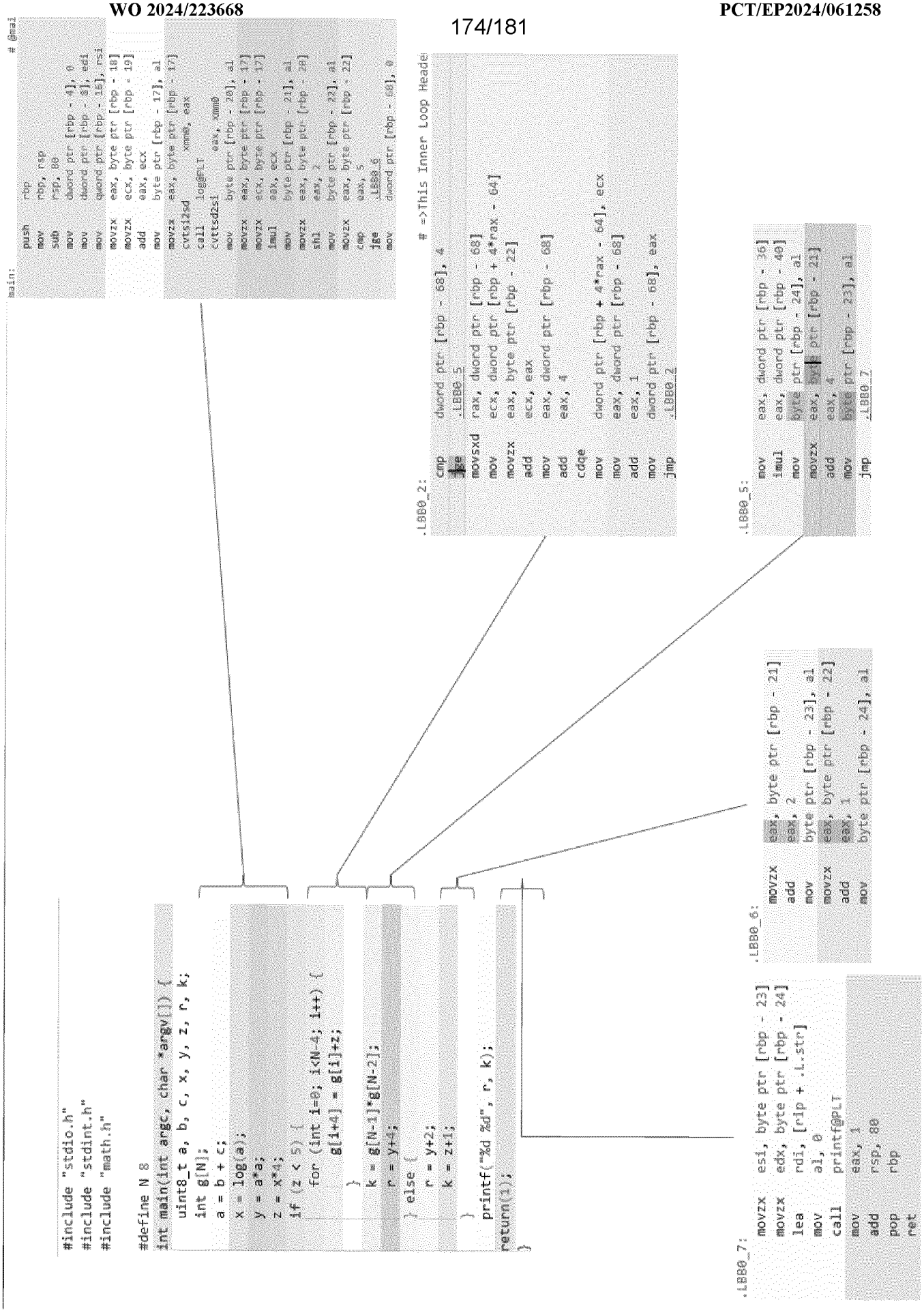

#include "stdio.h"
#include "stdint.h"
#include "math.h"
#define N 8
int main(int argc, char *argv[]) {
uint8_t a, b, c, x, y, z, r, k;
int g[N];
a = b + c;
x = log(a);
y = a*a;
z = x*4;
if (z < 5) {
for (int i=0; i<N-4; i++) {
g[i+4] = g[i]+z;
}
k = g[N-1]*g[N-2];
r = y+4;
} else {
r = y+2;
k = z+1;
}
printf("%d %d", r, k);
return(1);
}
main:
push rbp
mov rbp, rsp
sub rsp, 80
mov dword ptr [rbp - 4], 0
mov dword ptr [rbp - 8], edi
mov qword ptr [rbp - 16], rsi
movzx eax, byte ptr [rbp - 18]
movzx ecx, byte ptr [rbp - 19]
add eax, ecx
mov byte ptr [rbp - 17], al
movzx eax, byte ptr [rbp - 17]
cvtsi2sd xmm0, eax
call log@PLT
cvttsd2si eax, xmm0
mov byte ptr [rbp - 20], al
movzx eax, byte ptr [rbp - 17]
movzx ecx, byte ptr [rbp - 17]
imul eax, ecx
mov byte ptr [rbp - 21], al
movzx eax, byte ptr [rbp - 20]
shl eax, 2
mov byte ptr [rbp - 22], al
movzx eax, byte ptr [rbp - 22]
cmp eax, 5
jge .LBB0_6
mov dword ptr [rbp - 68], 0
=>This Inner Loop Heade
.LBB0_2:
cmp dword ptr [rbp - 68], 4
jle .LBB0_5
movsxd rax, dword ptr [rbp - 68]
mov ecx, dword ptr [rbp + 4*rax - 64]
movzx eax, byte ptr [rbp - 22]
add ecx, eax
mov eax, dword ptr [rbp - 68]
add eax, 4
cdqe
mov dword ptr [rbp + 4*rax - 64], ecx
mov eax, dword ptr [rbp - 68]
add eax, 1
mov dword ptr [rbp - 68], eax
jmp .LBB0_2
.LBB0_5:
mov eax, dword ptr [rbp - 36]
imul eax, dword ptr [rbp - 40]
mov byte ptr [rbp - 24], al
movzx eax, byte ptr [rbp - 21]
add eax, 4
mov byte ptr [rbp - 23], al
jmp .LBB0_7
.LBB0_6:
movzx eax, byte ptr [rbp - 21]
add eax, 2
mov byte ptr [rbp - 23], al
movzx eax, byte ptr [rbp - 22]
add eax, 1
mov byte ptr [rbp - 24], al
.LBB0_7:
movzx esi, byte ptr [rbp - 23]
movzx edx, byte ptr [rbp - 24]
lea rdi, [rip + .L.str]
mov al, 0
call printf@PLT
mov eax, 1
add rsp, 80
pop rbp
ret

Fig. 161

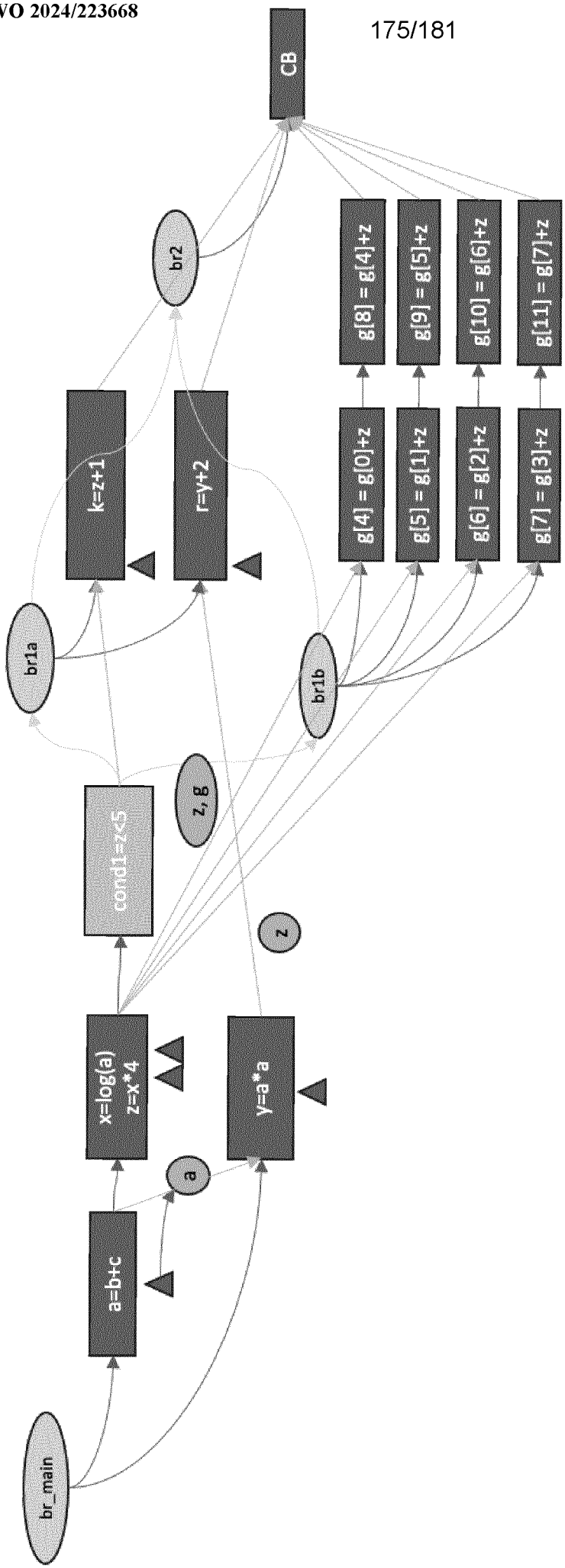

CB
br2
g[8] = g[4]+z
g[9] = g[5]+z
g[10] = g[6]+z
g[11] = g[7]+z
k=z+1
r=y+2
g[4] = g[0]+z
g[5] = g[1]+z
g[6] = g[2]+z
g[7] = g[3]+z
br1a
br1b
z, g
cond1=z<5
z
x=log(a)
z=x*4
y=a*a
a
a=b+c
br_main

Fig. 162

load, %r1, 5 := can be scheduled at least 1 CB ahead
store, %2 := can be scheduled after the R-instr -> pipelined step
not to overwrite register in next stage -> needed based on transfer
a=b+c
x= log(a)
z=x*4
y=a*a
cond1=z<5
a
z
Sreg = 3
load, %r1, %id2
load, %r2, %id3
add, %r3, %r1, %r2
store, %id1, %3
%r1 %r2 %r3 %r4
%id1 %id2 (a)
Sreg = 4
(load, %r3, %r3)
load, %r4, 4
log, %r2, %r3
store, %id4, %r2
mul, %1, %r2, %r4
store, %id6, %r1
%r1 %r2 %r3 %r4
(z) (x) (a) C=4
Sreg = 2
(load, %r1, %r1)
load, %r1, 5
cmp, %r2, %r1, %r4
%r1 %r2 %r3 %r4
(z) (cond1) (a) C=4
Sreg = 2
(load,<>, %r3)
mul, %r6, %r3, %r3
store, %id5, %2
%r5 %r6 %r7 %r8
(y)
ALU1 pipeline
ALU2 pipeline
a b c x y z r k g[0] g[11]
%id1 %id2 %id3 %id4 %id5 %id6 %id7 %id8 %id9

Fig. 163

g[4] = g[0]+z
g[5] = g[1]+z
g[6] = g[2]+z
g[7] = g[3]+z
g[8] = g[4]+z
g[9] = g[5]+z
g[10] = g[6]+z
g[11] = g[7]+z
ALU1 pipeline
ALU2 pipeline
g[0]
z
g[1]
g[4] = g[0]+z
g[5] = g[1]+z
z, g[4]
g[8] = g[4]+z
z, g[5]
g[9] = g[5]+z
Sreg = 2
(load, %r1, %r1)
load, %r2, g[0]
add, %r3, %r1, %r2
store, %id9+4, %r3
Sreg = 2+1
(load, %r1, %r1)
load, %r4, g[1]
add, B%1, %r1, %r4
store, %id9+5, B%1
Sreg = 2+1
(load, %r1, %r1)
(load,%r3, %r3)
add, %r2, %r1, %r3
store, %id9+8, %r2
Sreg = 2+1
(load, %r1, %r1)
(load, B%1, B%1)
add, %r3, %r1, B%1
store, %id9+9, %r3
mark parallel transfers
%r1 %r2 %r3 %r4 B%1
(z) (g[0]) (g[4])
%r1 %r2 %r3 %r4 B%1
(z) (g[0]) (g[4]) (g[1]) (g[5])
%r1 %r2 %r3 %r4 B%1
(z) (g[8]) (g[4]) (g[1]) (g[5])
%r1 %r2 %r3 %r4 B%1
(z) (g[8]) (g[9]) (g[1]) (g[5])

Fig. 164

g[4] = g[0]+z
g[5] = g[1]+z
g[6] = g[2]+z
g[7] = g[3]+z
g[8] = g[4]+z
g[9] = g[5]+z
g[10] = g[6]+z
g[11] = g[7]+z
ALU1 pipeline
ALU2 pipeline
g[2]
z
g[6] = g[2]+z
g[3]
g[7] = g[3]+z
z, g[6]
g[10] = g[6]+z
z, g[7]
g[11] = g[7]+z
Sreg = 2
(load, %r5, %r5)
load, %r2, g[2]
add, %r7, %r5, %r6
store, %id9+6, %r7
Sreg = 2+1
(load, %r5, %r5)
load, %r8, g[3]
add, B%2, %r5, %r8
store, %id9+7, B%2
Sreg = 2+1
(load, %r5, %r5)
(load, %r7, %r7)
add, %r6, %r5, %r7
store, %id9+10, %r6
Sreg = 2+1
(load, %r5, %r5)
(load, B%2, B%2)
add, %r7, B%2, %r5
store, %id9+11, %r7
%r5 %r6 %r7 %r8 B%2
(z) (g[2]) (g[6])
%r5 %r6 %r7 %r8 B%2
(z) (g[2]) (g[6]) (g[3]) (g[7])
%r5 %r6 %r7 %r8 B%2
(z) (g[10]) (g[6]) (g[3]) (g[7])
%r5 %r6 %r7 %r8 B%2
(z) (g[10]) (g[11]) (g[1]) (g[7])

Fig. 165

pipeline 1		pipeline 2
load, %r1, %id2		
load, %r2, %id3		
load, %r4, 4		
add, %r3, %r1, %r2		
store, %id1, %3		
load, %r1, 5		
log, %r2, %r3		mul, %r6, %r3, %r3
store, %id4, %r2		store, %id5, %2
mul, %1, %r2, %r4		
store, %id6, %r1		
cmp, %r2, %r1, %r4		
nop		
nop		
load, %r2, g[0]		load, %r2, g[2]
load, %r4, g[1]		load, %r8, g[3]
add, %r3, %r1, %r2		add, %r7, %r5, %r6
store, %id9+4, %r3		store, %id9+6, %r7
add, B%1, %r1, %r4		add, B%2, %r5, %r8
store, %id9+5, B%1		store, %id9+7, B%2
add, %r2, %r1, %r3		add, %r6, %r5, %r7
store, %id9+8, %r2		store, %id9+10, %r6
add, %r3, %r1, B%1		add, %r7, B%2, %r5
store, %id9+9, %r3		store, %id9+11, %r7

Fig. 166

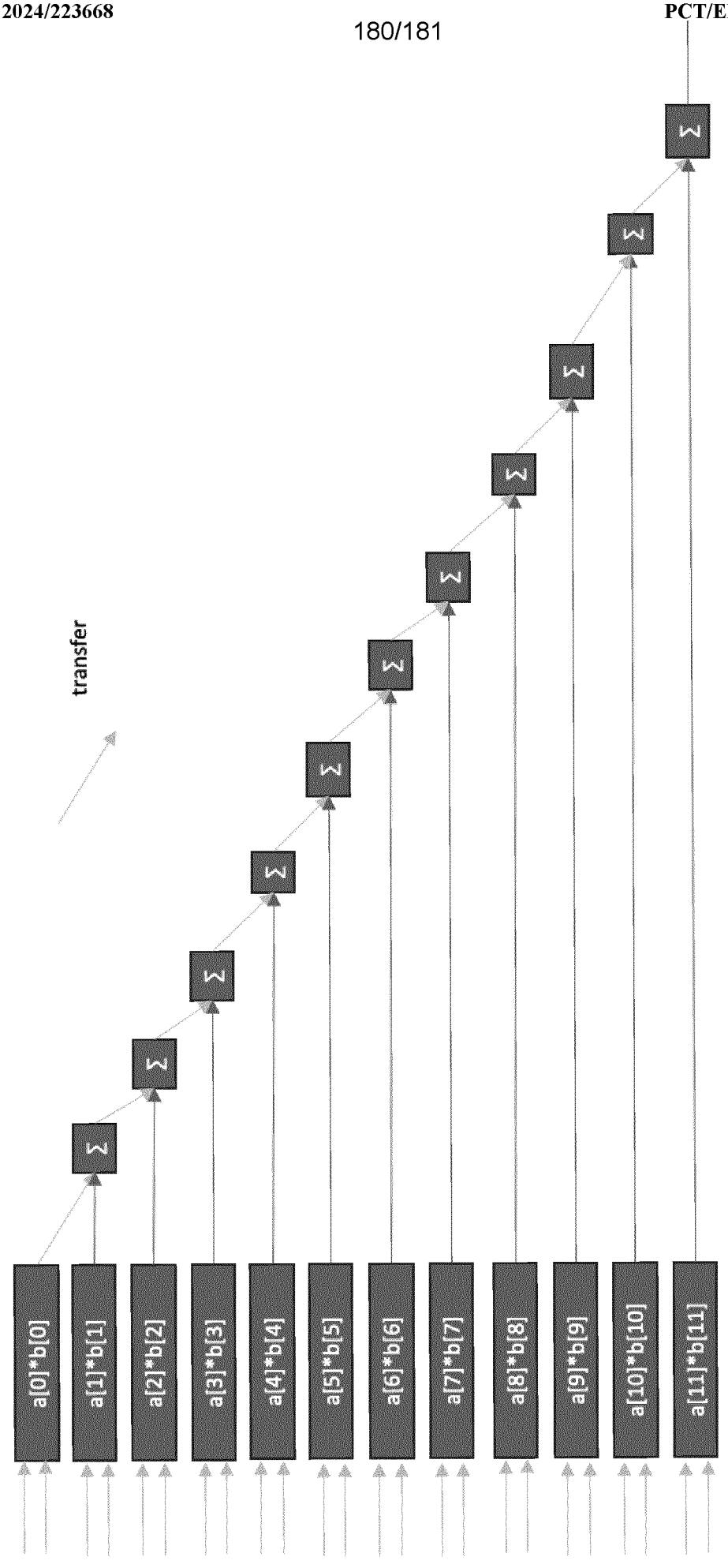

a[0]*b[0]
a[1]*b[1]
a[2]*b[2]
a[3]*b[3]
a[4]*b[4]
a[5]*b[5]
a[6]*b[6]
a[7]*b[7]
a[8]*b[8]
a[9]*b[9]
a[10]*b[10]
a[11]*b[11]
Σ
transfer

Fig. 167

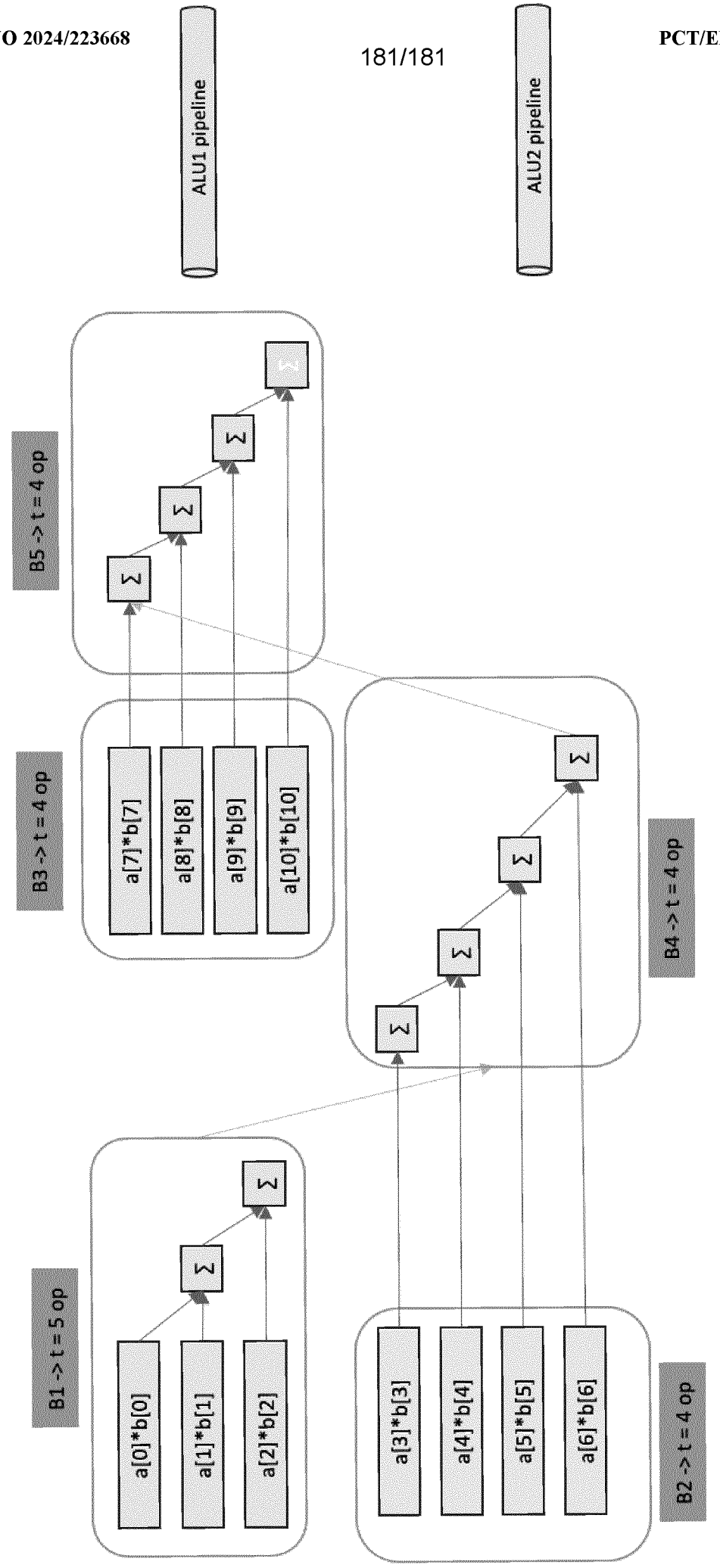

ALU1 pipeline
ALU2 pipeline
B5 -> t = 4 op
B3 -> t = 4 op
B4 -> t = 4 op
B1 -> t = 5 op
B2 -> t = 4 op
a[0]*b[0]
a[1]*b[1]
a[2]*b[2]
a[3]*b[3]
a[4]*b[4]
a[5]*b[5]
a[6]*b[6]
a[7]*b[7]
a[8]*b[8]
a[9]*b[9]
a[10]*b[10]

Fig. 168