

## 1422 A APPENDIX

### 1423 A.1 Array Index Notation Grammar

1424 The full syntax of array index notation can be found in Figure 32.  
1425

```

1426 <array_stmt> ::= <access> '=' <expr>
1427 <access>      ::= <tensor> {<index>}
1428 <index>       ::= <index_var> [ <index_slice> ]
1429 <index_slice> ::= '(' <lo> ':' <hi> [ ':' <st> ] ')'
1430 <expr>        ::= <literal> | <access> | <call_expr> | <reduce_expr>
1431               | <binary_expr> | <unary_expr> | '(' <expr> ')'
1432 <call_expr>   ::= <func> '(' <expr> {',' <expr>} ')'
1433 <reduce_expr> ::= <func> <expr>
1434               (index_var)
1435 <binary_expr> ::= <expr> <op> <expr>

```

1436 Fig. 32. The syntax of array index notation. Expressions within braces may be repeated any number of times.  
1437  
1438 <func> and <op> both represent arbitrary (user-defined or predefined) functions and are implemented in the  
1439 same way; they differ only in how they are invoked.  
1440  
1441

### 1442 1443 A.2 PyData/Sparse API

1444 An example of performing the xor operation on two sparse tensors using PyData/Sparse is found  
1445 below.  
1446

```

1447 1 import numpy
1448 2 import sparse
1449 3
1450 4 # Create some tensors.
1451 5 dim = 1000
1452 6 A = sparse.random((dim, dim, dim))
1453 7 B = sparse.random((dim, dim, dim))
1454 8 # Perform the XOR computation.
1455 9 C = numpy.logical_xor(A, B)

```

1456 An example performing the GCD operation can be found below:

```

1457 1 import numpy
1458 2 import
1459 3
1460 4 def gcd(x, y):
1461 5     return ... # Compute the GCD between x and y.
1462 6 # Register the gcd function as a ufunc.
1463 7 gcd = np.frompyfunc(gcd, 2, 1)
1464 8
1465 9 # Create some tensors.
1466 10 dim = 1000
1467 11 A = sparse.random((dim, dim, dim))
1468 12 B = sparse.random((dim, dim, dim))
1469 13 # Perform the XOR computation.
1470 14 C = gcd(A, B)

```

1471 While this code is simpler than the code to use our sparse array compiler, users do not have  
1472 control over many factors, such as the formats of the tensors, and are restricted to the predefined  
1473 set of NumPy functions.

### 1474 **A.3 Iteration Lattice Construction Algorithm**

1476 As described in Section 6, the presented iteration lattice construction algorithm (Algorithm 1)  
1477 supports only array index notation expressions that do not contain repeat tensors. Fig. 18 illustrates  
1478 an example of when iteration sub-spaces do not overlap when the index notation contains a repeated  
1479 tensor. This example motivates our implementation of a filtered Cartesian Product.

1480 We include the full algorithm that does support repeated tensors in Algorithm 2.

1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519

**Algorithm 2** Full iteration lattice construction algorithm

---

```

1520 procedure CONSTRUCTLATTICE (FunctionAlgebra A, FunctionArguments args)
1521 // Preprocessing steps
1522 Algebra A = DEMORGAN(A) ▷ Apply De Morgan's Law
1523 Algebra A = AUGMENT(A, args) ▷ Augmentation pass
1524 return BUILDLATTICE(A)
1525 end procedure
1526
1527 // let  $\mathcal{L}$  represent an iteration lattice and  $p$  represent an iteration lattice point
1528 procedure BUILDLATTICE (Algebra A)
1529 if A is Tensor(t) then ▷ Segment Rule
1530 return  $\mathcal{L}(p(\{t\}, \text{producer}=\text{true}))$ 
1531 else if A is  $\sim$ Tensor(t) then ▷ Complement Rule
1532  $p_o = p(\{t, \cup\}, \text{producer}=\text{false})$ 
1533  $p_p = p(\{\cup\}, \text{producer}=\text{true})$ 
1534 return  $\mathcal{L}(\{p_o, p_p\})$ 
1535 else if A is (left  $\cap$  right) then ▷ Intersection Rule
1536  $\mathcal{L}_l, \mathcal{L}_r = \text{BUILDLATTICE}(\text{left}), \text{BUILDLATTICE}(\text{right})$ 
1537  $\text{cp} = \text{FILTEREDCARTESIANPRODUCT}(\mathcal{L}_l.\text{points}(), \mathcal{L}_r.\text{points}())$ 
1538  $\text{mergedPoints} = \{p(\{p_l + p_r\}, \text{producer}=p_l.\text{producer} \wedge p_l.\text{producer}) : \forall (p_l, p_r) \in \text{cp}\}$ 
1539  $\text{mergedPoints} = \text{REMOVEDUPLICATES}(\text{mergedPoints}, \text{ommitterPrecedence})$ 
1540 return  $\mathcal{L}(\text{mergedPoints})$ 
1541 else if A is (left  $\cup$  right) then ▷ Union Rule
1542  $\mathcal{L}_l, \mathcal{L}_r = \text{BUILDLATTICE}(\text{left}), \text{BUILDLATTICE}(\text{right})$ 
1543  $\text{cp} = \text{FILTEREDCARTESIANPRODUCT}(\mathcal{L}_l.\text{points}(), \mathcal{L}_r.\text{points}())$ 
1544  $\text{mergedPoints} = \{p(\{p_l + p_r\}, \text{producer}=p_l.\text{producer} \vee p_l.\text{producer}) : \forall (p_l, p_r) \in \text{cp}\}$ 
1545  $\text{mergedPoints} = \text{mergedPoints} + \mathcal{L}_l.\text{points}() + \mathcal{L}_r.\text{points}()$ 
1546  $\text{mergedPoints} = \text{REMOVEDUPLICATES}(\text{mergedPoints}, \text{producerPrecedence})$ 
1547 return  $\mathcal{L}(\text{mergedPoints})$ 
1548 end procedure
1549
1550 procedure FILTEREDCARTESIANPRODUCT (LatticePoints left, LatticePoints right)
1551  $p_{l,\text{root}}, p_{r,\text{root}} = \text{left.root}, \text{right.root}$ 
1552 for ( $p_l$  in left)  $\times$  ( $p_r$  in right) do  $\text{overlap} = \text{true}$ 
1553 for tensor in  $p_l$  do
1554 if (tensor in  $p_{r,\text{root}} \wedge$  (tensor not in  $p_l$ )) then  $\text{overlap} = \text{false}$ 
1555 end for
1556 for tensor in  $p_r$  do
1557 if (tensor in  $p_{l,\text{root}} \wedge$  (tensor not in  $p_l$ )) then  $\text{overlap} = \text{false}$ 
1558 end for
1559 if  $\text{overlap}$  then  $\text{cp} += \{(p_l, p_r)\}$ 
1560 end for
1561 return cp
1562 end procedure

```

---

1569 **A.4 Medical Imaging Edge Detection**

1570

1571

1572

1573

1574

1575

1576

1577

1578

1579

1580

1581

1582

1583

1584

1585

1586

1587

1588

1589

1590

1591

1592

1593

1594

1595

1596

1597

1598

1599

1600

1601

1602

1603

1604

1605

1606

1607

1608

1609

1610

1611

1612

1613

1614

1615

1616

1617

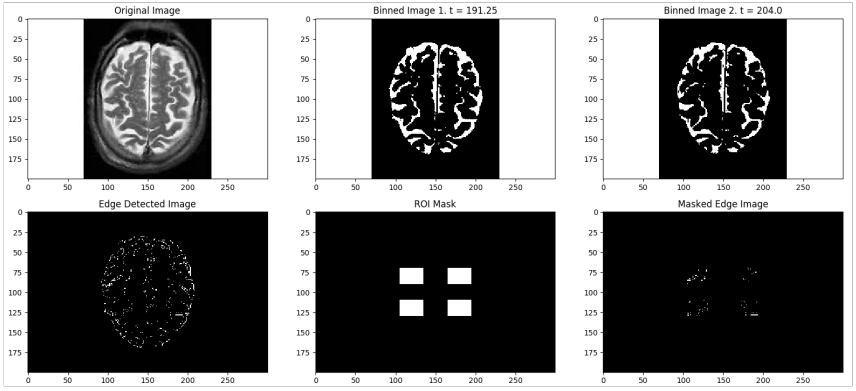


Fig. 33. Example MRI image, thresholding, ROI mask, and output