Scatter 层[TODO]

- mode（要求 TensorRT>=8.2）
  - Scatter ELEMENT 模式
  - Scatter ND 模式

---

mode（要求 TensorRT>=8.2）

ELEMENT 模式

```python
import numpy as np
from cuda import cudart
import tensorrt as trt

np.random.seed(97)
nIn, cIn, hIn, wIn = 1, 3, 4, 5  # 输入张量 NCHW
data0 = np.arange(nIn * cIn * hIn * wIn, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)  # 输入数据
data1 = np.tile(np.arange(hIn), [nIn, cIn, 1, wIn]).astype(np.int32).reshape(nIn, cIn, hIn, wIn)
data2 = -data0

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

# 仅适用于 axis = 2
def scatterCPU(data0,data1,data2):
    nIn,cIn,hIn,wIn = data0.shape
    output = data0
    for n in range(nIn):
        for c in range(cIn):
            for h in range(hIn):
                for w in range(wIn):
                    output[n,c,data1[n,c,h,w],w] = data2[n,c,h,w]
                    #print("<%d,%d,%d,%d>[%d,%d,%d,%d],%f>"%
(n,c,h,w,n,c,data1[n,c,h,w],w,data2[n,c,h,w]))
                    #print(output)
    return output

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (nIn, cIn, hIn, wIn))
inputT2 = network.add_input('inputT2', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#----------------------------------------------------------- --------------------# 替换部分
scatterLayer = network.add_scatter(inputT0, inputT1, inputT2, trt.ScatterMode.ELEMENT)
scatterLayer.axis = 2
#----------------------------------------------------------- --------------------# 替换部分
network.mark_output(scatterLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()
```

```python
inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
inputH2 = np.ascontiguousarray(data2.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(3), dtype=trt.nptype(engine.get_binding_dtype(3)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, inputD2 = cudart.cudaMallocAsync(inputH2.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD2, inputH2.ctypes.data, inputH2.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(inputD2), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH1 :", data1.shape)
print(data1)
print("inputH2 :", data2.shape)
print(data2)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)

resCPU = scatterCPU(data0,data1,data2)
print("diff:\n",outputH0 - resCPU)
```

- 输入张量 0 形状 (1, 3, 4, 5)，输入张量 2 形状与输入张量 0 相同，值为其相反数

$$\left[\left[\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 5. & 6. & 7. & 8. & 9. \\ 10. & 11. & 12. & 13. & 14. \\ 15. & 16. & 17. & 18. & 19. \end{bmatrix} \begin{bmatrix} 20. & 21. & 22. & 23. & 24. \\ 25. & 26. & 27. & 28. & 29. \\ 30. & 31. & 32. & 33. & 34. \\ 35. & 36. & 37. & 38. & 39. \end{bmatrix} \begin{bmatrix} 40. & 41. & 42. & 43. & 44. \\ 45. & 46. & 47. & 48. & 49. \\ 50. & 51. & 52. & 53. & 54. \\ 55. & 56. & 57. & 58. & 59. \end{bmatrix}\right]\right]$$

- 输入张量 1 形状 (1, 3, 4, 5)

$$\left[\left[\begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 & 2 \\ 3 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 & 2 \\ 3 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 & 2 \\ 3 & 0 & 1 & 2 & 3 \end{bmatrix}\right]\right]$$

- 输出张量 0 形状 (1, 3, 4, 5)

$$\left[\left[\begin{bmatrix} -0. & -16. & -12. & -8. & -4. \\ -5. & -1. & -17. & -13. & -9. \\ -10. & -6. & -2. & -18. & -14. \\ -15. & -11. & -7. & -3. & -19. \end{bmatrix} \begin{bmatrix} -20. & -36. & -32. & -28. & -24. \\ -25. & -21. & -37. & -33. & -29. \\ -30. & -26. & -22. & -38. & -34. \\ -35. & -31. & -27. & -23. & -39. \end{bmatrix} \begin{bmatrix} -40. & -56. & -52. & -48. & -44. \\ -45. & -41. & -57. & -53. & -49. \\ -50. & -46. & -42. & -58. & -54. \\ -55. & -51. & -47. & -43. & -59. \end{bmatrix}\right]\right]$$

- 含义：参考 [Onnx ScatterElements 算子](#) 和 [TensorRT C++ API 说明](#)

- 数据张量 data、索引张量 index、更新张量 update、输出张量 output 形状相同（$dim = r$），均为 $[d_0, d_1, \dots, d_{r-1}]$，指定 $axis = p$（$0 \le p < r$），则
- 命循环变量 i_j 满足 $0 \le j < r, 0 \le i_j < d_j$，则计算过程用 numpy 语法可以写作：$output[i_0, i_1, \dots, i_{p-1}, index[i_0, i_1, \dots, i_{p-1}, i_p, i_{p+1}, \dots, i_{r-1}], i_{p+1}, \dots, i_{r-1}] = data[i_0, i_1, \dots, i_{p-1}, i_p, i_{p+1}, \dots, i_{r-1}]$
- 对于上面的范例代码，就是：

$$
\begin{aligned}
index[0,0,0,0] = \color{red}{0} \Rightarrow output[0,0,\color{red}{0},0] = update[0,0,0,0] = -0. \\
index[0,0,0,1] = \color{red}{1} \Rightarrow output[0,0,\color{red}{1},1] = update[0,0,0,1] = -1. \\
index[0,0,0,2] = \color{red}{2} \Rightarrow output[0,0,\color{red}{2},2] = update[0,0,0,2] = -2. \\
\dots \\
index[0,0,1,0] = \color{green}{1} \Rightarrow output[0,0,\color{green}{1},0] = update[0,0,1,0] = -5. \\
index[0,0,1,1] = \color{green}{2} \Rightarrow output[0,0,\color{green}{2},1] = update[0,0,2,1] = -6. \\
index[0,0,1,2] = \color{green}{3} \Rightarrow output[0,0,\color{green}{3},2] = update[0,0,3,2] = -7. \\
\dots \\
index[0,1,0,0] = \color{blue}{0} \Rightarrow output[0,1,\color{blue}{0},0] = update[0,1,0,0] = -20. \\
index[0,1,0,1] = \color{blue}{1} \Rightarrow output[0,1,\color{blue}{1},1] = update[0,1,0,1] = -21. \\
index[0,1,0,2] = \color{blue}{2} \Rightarrow output[0,1,\color{blue}{2},2] = update[0,1,0,2] = -22. \\
\dots
\end{aligned}
$$

- 计算公式恰好为 GatherElement 算子公式等号左右两项的索引进行交换
- output 元素的更新没有次序保证。如果两次更新指向 output 同一位置，且两次更新的值不同，则不能保证 output 该位置上的值选哪一次更新的结果。例如，将范例代码中 data1 改为 `data1 = np.zeros([nIn, cIn, hIn, wIn], dtype=np.int32)`，那么 output[:,:,0,:] 值会是来自 update 的负的随机整数

ND 模式

```python
import numpy as np
from cuda import cudart
import tensorrt as trt

np.random.seed(97)
data0 = np.arange(2 * 3 * 4 * 5, dtype=np.float32).reshape(2, 3, 4, 5)  # 输入数据
data1 = np.array([[[0, 2, 1, 1], [1, 0, 3, 2], [0, 1, 2, 3]], [[1, 2, 1, 1], [0, 0, 3, 2], [1, 1, 2, 3]]], dtype=np.int32)
data2 = -np.arange(2 * 3, dtype=np.float32).reshape(2, 3)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

def scatterCPU(data0,data1,data2):
    output = data0
    for i in range(2):
        for j in range(3):
            print("i=%d,j=%d,index=%s,updateValue=%f"%(i,j,data1[i,j],data2[i,j]))
            output[data1[i,j][0],data1[i,j][1],data1[i,j][2],data1[i,j][3]] = data2[i,j]
    return output

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (2, 3, 4, 5))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (2, 3, 4))
inputT2 = network.add_input('inputT2', trt.DataType.FLOAT, (2, 3))
```

```python
#------------------------------------------------------------ --------------------# 替换部分
scatterLayer = network.add_scatter(inputT0, inputT1, inputT2, trt.ScatterMode.ND)
#------------------------------------------------------------ --------------------# 替换部分
network.mark_output(scatterLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
inputH2 = np.ascontiguousarray(data2.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(3), dtype=trt.nptype(engine.get_binding_dtype(3)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, inputD2 = cudart.cudaMallocAsync(inputH2.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD2, inputH2.ctypes.data, inputH2.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(inputD2), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH1 :", data1.shape)
print(data1)
print("inputH2 :", data2.shape)
print(data2)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)

resCPU = scatterCPU(data0,data1,data2)
print("diff:\n",outputH0 - resCPU)
```

- 输入张量 0 形状 (2, 3, 4, 5)

$$\left[\begin{matrix} \left[\begin{matrix} \left[\begin{matrix} 0. & 1. & 2. & 3. & 4. \\ 5. & 6. & 7. & 8. & 9. \\ 10. & 11. & 12. & 13. & 14. \\ 15. & 16. & 17. & 18. & 19. \end{matrix}\right] \left[\begin{matrix} 20. & 21. & 22. & 23. & 24. \\ 25. & 26. & 27. & 28. & 29. \\ 30. & 31. & 32. & 33. & 34. \\ 35. & 36. & 37. & 38. & 39. \end{matrix}\right] \left[\begin{matrix} 40. & 41. & 42. & 43. & 44. \\ 45. & 46. & 47. & 48. & 49. \\ 50. & 51. & 52. & 53. & 54. \\ 55. & 56. & 57. & 58. & 59. \end{matrix}\right] \end{matrix}\right] \\ \left[\begin{matrix} \left[\begin{matrix} 60. & 61. & 62. & 63. & 64. \\ 65. & 66. & 67. & 68. & 69. \\ 70. & 71. & 72. & 73. & 74. \\ 75. & 76. & 77. & 78. & 79. \end{matrix}\right] \left[\begin{matrix} 80. & 81. & 82. & 83. & 84. \\ 85. & 86. & 87. & 88. & 89. \\ 90. & 91. & 92. & 93. & 94. \\ 95. & 96. & 97. & 98. & 99. \end{matrix}\right] \left[\begin{matrix} 100. & 101. & 102. & 103. & 104. \\ 105. & 106. & 107. & 108. & 109. \\ 110. & 111. & 112. & 113. & 114. \\ 115. & 116. & 117. & 118. & 119. \end{matrix}\right] \end{matrix}\right] \end{matrix}\right]$$

- 输入张量 1 形状 (2, 3, 4)

$$\begin{bmatrix}\begin{bmatrix}0 & 2 & 1 & 1\\1 & 0 & 3 & 2\\0 & 1 & 2 & 3\end{bmatrix} & \begin{bmatrix}1 & 2 & 1 & 1\\0 & 0 & 3 & 2\\1 & 1 & 2 & 3\end{bmatrix}\end{bmatrix}$$

- 输入张量 2 形状 (2, 3)

$$\begin{bmatrix}-0. & -1. & -2.\\-3. & -4. & -5.\end{bmatrix}$$

- 输出张量 0 形状 (2, 3, 4, 5)

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}0. & 1. & 2. & 3. & 4.\\5. & 6. & 7. & 8. & 9.\\10. & 11. & 12. & 13. & 14.\\15. & 16. & -4. & 18. & 19.\end{bmatrix} & \begin{bmatrix}20. & 21. & 22. & 23. & 24.\\25. & 26. & 27. & 28. & 29.\\30. & 31. & 32. & -2. & 34.\\35. & 36. & 37. & 38. & 39.\end{bmatrix} & \begin{bmatrix}40. & 41. & 42. & 43. & 44.\\45. & -0. & 47. & 48. & 49.\\50. & 51. & 52. & 53. & 54.\\55. & 56. & 57. & 58. & 59.\end{bmatrix}\end{bmatrix}\\\begin{bmatrix}\begin{bmatrix}60. & 61. & 62. & 63. & 64.\\65. & 66. & 67. & 68. & 69.\\70. & 71. & 72. & 73. & 74.\\75. & 76. & -1. & 78. & 79.\end{bmatrix} & \begin{bmatrix}80. & 81. & 82. & 83. & 84.\\85. & 86. & 87. & 88. & 89.\\90. & 91. & 92. & -5. & 94.\\95. & 96. & 97. & 98. & 99.\end{bmatrix} & \begin{bmatrix}100. & 101. & 102. & 103. & 104.\\105. & -3. & 107. & 108. & 109.\\110. & 111. & 112. & 113. & 114.\\115. & 116. & 117. & 118. & 119.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 范例代码 2，修改上面代码的若干行：

```
data0 = np.arange(2 * 3 * 4 * 5, dtype=np.float32).reshape(2, 3, 4, 5)
data1 = np.array([[0, 2, 1], [1, 0, 3], [0, 1, 2], [1, 2, 1], [0, 0, 3], [1, 1, 2]], dtype=np.int32)
data2 = -np.arange(6*5, dtype=np.float32).reshape(6,5)

def scatterCPU(data0,data1,data2):
    output = data0
    for i in range(6):
            print("i=%d,index=%s,updateValue="%(i,data1[i]),data2[i])
            output[data1[i][0],data1[i][1],data1[i][2]] = data2[i]
    return output

inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (2, 3, 4, 5))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (6, 3))
inputT2 = network.add_input('inputT2', trt.DataType.FLOAT, (6, 5))
```

- 含义：参考 Onnx ScatterND 算子 和 TensorRT C++ API 说明
- 数据张量 data 形状 $[d_0, d_1, \dots, d_{r-1}]$（r 维），索引张量 index 形状 $[a_0, a_1, \dots, a_{q-1}]$（q 维），更新张量 update 形状 $[b_0, b_1, \dots, b_{s-1}]$（$s = r - a_{q-1} + q - 1$ 维），输出张量 output 形状与 data 相同
- 若 $r = a_{q-1}$（范例代码 1），则 $s = q - 1$，此时对于 $0 \le i < q$，有 $b_i = a_i$（update 比 index 少一维，且各维尺寸跟 index 去掉最低维后对应相等）
- 若 $r > a_{q-1}$（范例代码 2），则 $s > q - 1$，此时对于 $0 \le i < q$，有 $b_i = a_i$，对于 $q \le i < s$，有 $b_i = d_{a_{q-1}+i-q}$（也即 $b_q = d_{a_{q-1}}, b_{q+1} = d_{a_{q-1}+1, \dots}$，最后一项 $i = s - 1$，此时 $b_{s-1} = d_{a_{q-1}+s-1-q} = d_{r-1}$，恰好取到 data 的最后一维的尺寸）
- 用 numpy 语法，记

```
q = len(index.shape)
nIndex = np.prod(index.shape[:-1])
index2D = index.reshape(nIndex,index.shape[-1])
update2D = update.reshape(nIndex,*update.shape[q-1:])
```

- 那么计算结果可以表示为

```
for i in nIndex:
    output[*index2D[i]] = update2D[i]
```

- 对于上面的范例代码 1，就是：

$$i = 0 \Rightarrow index2D[0] = [0, 2, 1, 1] \Rightarrow output[0, 2, 1, 1] = update2D[0] = -0.$$
$$\cdots$$
$$i = 5 \Rightarrow index2D[5] = [1, 1, 2, 3] \Rightarrow output[1, 1, 2, 3] = update2D[5] = -5.$$

- 或者还原回 index 和 update 的原始下标来表示，就是：

$$\begin{aligned}
i = 0, j = 0 &\Rightarrow index[0, 0] = [0, 2, 1, 1] \Rightarrow output[0, 2, 1, 1] = update[0, 0] = -0. \\
i = 0, j = 1 &\Rightarrow index[0, 1] = [1, 0, 3, 2] \Rightarrow output[1, 0, 3, 2] = update[0, 1] = -1. \\
i = 0, j = 2 &\Rightarrow index[0, 2] = [0, 1, 2, 3] \Rightarrow output[0, 1, 2, 3] = update[0, 2] = -2. \\
&\cdots \\
i = 1, j = 0 &\Rightarrow index[1, 0] = [1, 2, 1, 1] \Rightarrow output[1, 2, 1, 1] = update[1, 0] = -3. \\
i = 1, j = 1 &\Rightarrow index[1, 1] = [0, 0, 3, 2] \Rightarrow output[0, 0, 3, 2] = update[1, 1] = -4. \\
i = 1, j = 2 &\Rightarrow index[1, 2] = [1, 1, 2, 3] \Rightarrow output[1, 1, 2, 3] = update[1, 2] = -5.
\end{aligned}$$

- 对于上面的范例代码 2，就是：

$$\begin{aligned}
&i = 0 \Rightarrow index2D[0] = [0, 2, 1] \Rightarrow output[0, 2, 1] = update2D[0] = [-0., -1., -2., -3., -4.] \\
&\cdots \\
&i = 5 \Rightarrow index2D[5] = [1, 1, 2] \Rightarrow output[1, 1, 2] = update2D[5] = [-25., -26., -27., -28., -29.,]
\end{aligned}$$

- 说明：
    - 记 $nIndex = a_0 * a_1 * \ldots * a_{q-2}$，
    - 把 index 变形为 $nIndex$ 行 a_{q-1} 列的矩阵 index2D，用其每一行来索引 data，同时把 update 变形为 nIndex 组形状为 $[b_{q-1}, \ldots, b_{s-1}]$ 的张量 update2D
    - 如果 $r = a_{q-1}$（范例代码 1），那么 index 的第 i 行作为索引恰好取到 output（或 data） 的一个元素（np.shape(output[*index2D[i]]==[])）；而此时 $b_{q-1} = b_{s-1} = 1$（因为 update 只有 $q-1$ 维，全在 $nIndex$ 维度上了），该索引在 update2D中也索引到一个元素，于是使用 update2D 的该元素来替换 output 的对应元素
    - 如果 $r > a_{q-1}$（范例代码 2），记 $nD = r - a_{q-1}$，那么 index 的第 i 行作为索引会取到 output（或 data） 的一个 nD 维子张量（len(np.shape(output[*index2D[i]]))==nD），形状 $[d_{a_{q-1}}, \ldots d_{r-1}]$；此时该索引在 update2D 中也索引到一个 nD 维的子张量，形状也是 $[d_{a_{q-1}}, \ldots d_{r-1}]$，于是使用 update2D 的该元素来替换 output 的对应元素
- 不满足 $s = r - a_{q-1} + q - 1$ 时报错：

```
[TRT] [E] 4: [graphShapeAnalyzer.cpp::computeOutputExtents::1032] Error Code 4: Miscellaneous ((Unnamed Layer* 0) [Scatter]: error while lowering shape of node)
```

- b_i 不满足约束条件时报错：

```
[TRT] [E] 4: [graphShapeAnalyzer.cpp::processCheck::581] Error Code 4: Internal Error ((Unnamed Layer* 0) [Scatter]: dimensions not compatible for ScatterND)
```