

```
ics
& (depth < MAXDEPTH)
{
    inside / inside
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, classically
    if;
    radiance = SampleLight( &rand, I, &t, &light;
    e.x + radiance.y + radiance.z) > 0) && (depth <
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
    random walk - done properly, closely following
    survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

/INFOMOV/

# Optimization & Vectorization

J. Bikker - Sep-Nov 2016 - Lecture 12: "GPGPU (3)"

# Welcome!



# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



# Introduction

## Beyond “Many Independent Threads”

Many algorithms do not lend themselves to GPGPU, at least not at first sight:

- Divide and conquer algorithms
  - Sorting
- Anything with an unpredictable number of iterations
  - Walking a linked list or a tree
  - Ray tracing
- Anything that needs to emit data in a compacted array
  - Run-length encoding
  - Duplicate removal
- Anything that requires inter-thread synchronization
  - Hash table
  - Linked list

In fact, lock-free implementations of linked lists and hash tables exist and can be used in CUDA, see e.g.:

Misra & Chaudhuri, 2012, Performance Evaluation of Concurrent Lock-free Data Structures on GPUs.

*Note that the possibility of using linked lists on the GPU does not automatically justify its use.*





# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



# Prefix Sum

## Prefix Sum

The prefix sum (or cumulative sum) of a sequence of numbers is a second sequence of numbers consisting of the running totals of the input sequence:

Input:  $x_0, x_1, x_2$

Output:  $x_0, x_0 + x_1, x_0 + x_1 + x_2$  (*inclusive*) or  $0, x_0, x_0 + x_1$  (*exclusive*).

Example:

input	1	2	2	1	4	3
inclusive	1	3	5	6	10	13
exclusive	0	1	3	5	6	10

Here, addition is used; more generally we can use an arbitrary binary associative operator.



# Prefix Sum

## Prefix Sum

input	1	2	2	1	4	3
inclusive	1	3	5	6	10	13
exclusive	0	1	3	5	6	10

## In C++:

```
// exclusive scan
out[0] = 0;
for ( i = 1; i < n; i++ ) out[i] = in[i-1] + out[i-1];
```

(Note the obvious loop dependency)

```

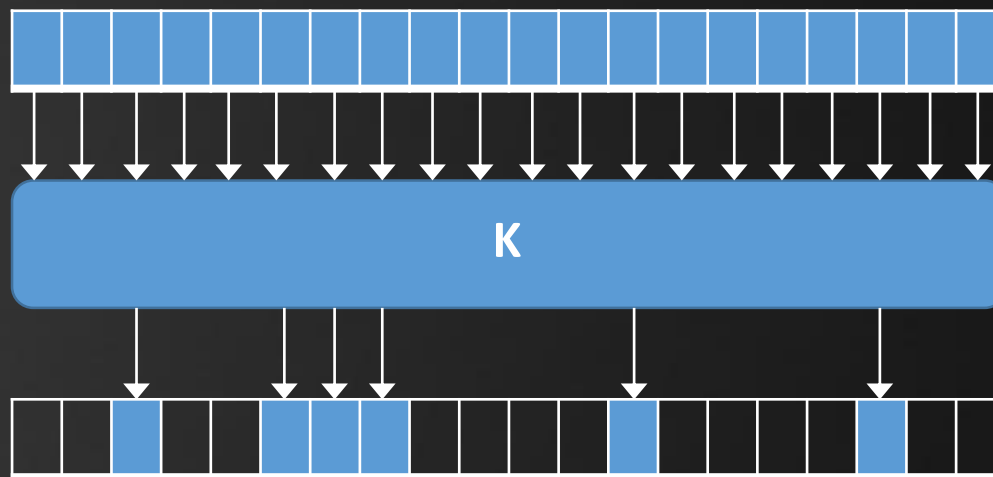
...ics
& (depth < MAXDEPTH)
...
c = inside / inside;
nt = nt / nc; odd = 1;
cos2t = 1.0f - nnt * nnt;
D, N );
)
...
at a = nt - nc, b = nt;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (1 -
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, clearly
if;
radiance = SampleLight( &rand, I, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R34, R35, R36, R37, R38, R39, R40, R41, R42, R43, R44, R45, R46, R47, R48, R49, R50, R51, R52, R53, R54, R55, R56, R57, R58, R59, R60, R61, R62, R63, R64, R65, R66, R67, R68, R69, R70, R71, R72, R73, R74, R75, R76, R77, R78, R79, R80, R81, R82, R83, R84, R85, R86, R87, R88, R89, R90, R91, R92, R93, R94, R95, R96, R97, R98, R99, R100, R101, R102, R103, R104, R105, R106, R107, R108, R109, R110, R111, R112, R113, R114, R115, R116, R117, R118, R119, R120, R121, R122, R123, R124, R125, R126, R127, R128, R129, R130, R131, R132, R133, R134, R135, R136, R137, R138, R139, R140, R141, R142, R143, R144, R145, R146, R147, R148, R149, R150, R151, R152, R153, R154, R155, R156, R157, R158, R159, R160, R161, R162, R163, R164, R165, R166, R167, R168, R169, R170, R171, R172, R173, R174, R175, R176, R177, R178, R179, R180, R181, R182, R183, R184, R185, R186, R187, R188, R189, R190, R191, R192, R193, R194, R195, R196, R197, R198, R199, R200, R201, R202, R203, R204, R205, R206, R207, R208, R209, R210, R211, R212, R213, R214, R215, R216, R217, R218, R219, R220, R221, R222, R223, R224, R225, R226, R227, R228, R229, R230, R231, R232, R233, R234, R235, R236, R237, R238, R239, R240, R241, R242, R243, R244, R245, R246, R247, R248, R249, R250, R251, R252, R253, R254, R255, R256, R257, R258, R259, R260, R261, R262, R263, R264, R265, R266, R267, R268, R269, R270, R271, R272, R273, R274, R275, R276, R277, R278, R279, R280, R281, R282, R283, R284, R285, R286, R287, R288, R289, R290, R291, R292, R293, R294, R295, R296, R297, R298, R299, R300, R301, R302, R303, R304, R305, R306, R307, R308, R309, R310, R311, R312, R313, R314, R315, R316, R317, R318, R319, R320, R321, R322, R323, R324, R325, R326, R327, R328, R329, R330, R331, R332, R333, R334, R335, R336, R337, R338, R339, R340, R341, R342, R343, R344, R345, R346, R347, R348, R349, R350, R351, R352, R353, R354, R355, R356, R357, R358, R359, R360, R361, R362, R363, R364, R365, R366, R367, R368, R369, R370, R371, R372, R373, R374, R375, R376, R377, R378, R379, R380, R381, R382, R383, R384, R385, R386, R387, R388, R389, R390, R391, R392, R393, R394, R395, R396, R397, R398, R399, R400, R401, R402, R403, R404, R405, R406, R407, R408, R409, R410, R411, R412, R413, R414, R415, R416, R417, R418, R419, R420, R421, R422, R423, R424, R425, R426, R427, R428, R429, R430, R431, R432, R433, R434, R435, R436, R437, R438, R439, R440, R441, R442, R443, R444, R445, R446, R447, R448, R449, R450, R451, R452, R453, R454, R455, R456, R457, R458, R459, R460, R461, R462, R463, R464, R465, R466, R467, R468, R469, R470, R471, R472, R473, R474, R475, R476, R477, R478, R479, R480, R481, R482, R483, R484, R485, R486, R487, R488, R489, R490, R491, R492, R493, R494, R495, R496, R497, R498, R499, R500, R501, R502, R503, R504, R505, R506, R507, R508, R509, R510, R511, R512, R513, R514, R515, R516, R517, R518, R519, R520, R521, R522, R523, R524, R525, R526, R527, R528, R529, R530, R531, R532, R533, R534, R535, R536, R537, R538, R539, R540, R541, R542, R543, R544, R545, R546, R547, R548, R549, R550, R551, R552, R553, R554, R555, R556, R557, R558, R559, R560, R561, R562, R563, R564, R565, R566, R567, R568, R569, R570, R571, R572, R573, R574, R575, R576, R577, R578, R579, R580, R581, R582, R583, R584, R585, R586, R587, R588, R589, R590, R591, R592, R593, R594, R595, R596, R597, R598, R599, R600, R601, R602, R603, R604, R605, R606, R607, R608, R609, R610, R611, R612, R613, R614, R615, R616, R617, R618, R619, R620, R621, R622, R623, R624, R625, R626, R627, R628, R629, R630, R631, R632, R633, R634, R635, R636, R637, R638, R639, R640, R641, R642, R643, R644, R645, R646, R647, R648, R649, R650, R651, R652, R653, R654, R655, R656, R657, R658, R659, R660, R661, R662, R663, R664, R665, R666, R667, R668, R669, R670, R671, R672, R673, R674, R675, R676, R677, R678, R679, R680, R681, R682, R683, R684, R685, R686, R687, R688, R689, R690, R691, R692, R693, R694, R695, R696, R697, R698, R699, R700, R701, R702, R703, R704, R705, R706, R707, R708, R709, R710, R711, R712, R713, R714, R715, R716, R717, R718, R719, R720, R721, R722, R723, R724, R725, R726, R727, R728, R729, R730, R731, R732, R733, R734, R735, R736, R737, R738, R739, R740, R741, R742, R743, R744, R745, R746, R747, R748, R749, R750, R751, R752, R753, R754, R755, R756, R757, R758, R759, R760, R761, R762, R763, R764, R765, R766, R767, R768, R769, R770, R771, R772, R773, R774, R775, R776, R777, R778, R779, R780, R781, R782, R783, R784, R785, R786, R787, R788, R789, R790, R791, R792, R793, R794, R795, R796, R797, R798, R799, R800, R801, R802, R803, R804, R805, R806, R807, R808, R809, R810, R811, R812, R813, R814, R815, R816, R817, R818, R819, R820, R821, R822, R823, R824, R825, R826, R827, R828, R829, R830, R831, R832, R833, R834, R835, R836, R837, R838, R839, R840, R841, R842, R843, R844, R845, R846, R847, R848, R849, R850, R851, R852, R853, R854, R855, R856, R857, R858, R859, R860, R861, R862, R863, R864, R865, R866, R867, R868, R869, R870, R871, R872, R873, R874, R875, R876, R877, R878, R879, R880, R881, R882, R883, R884, R885, R886, R887, R888, R889, R890, R891, R892, R893, R894, R895, R896, R897, R898, R899, R900, R901, R902, R903, R904, R905, R906, R907, R908, R909, R910, R911, R912, R913, R914, R915, R916, R917, R918, R919, R920, R921, R922, R923, R924, R925, R926, R927, R928, R929, R930, R931, R932, R933, R934, R935, R936, R937, R938, R939, R940, R941, R942, R943, R944, R945, R946, R947, R948, R949, R950, R951, R952, R953, R954, R955, R956, R957, R958, R959, R960, R961, R962, R963, R964, R965, R966, R967, R968, R969, R970, R971, R972, R973, R974, R975, R976, R977, R978, R979, R980, R981, R982, R983, R984, R985, R986, R987, R988, R989, R990, R991, R992, R993, R994, R995, R996, R997, R998, R999, R1000, R1001, R1002, R1003, R1004, R1005, R1006, R1007, R1008, R1009, R1010, R1011, R1012, R1013, R1014, R1015, R1016, R1017, R1018, R1019, R1020, R1021, R1022, R1023, R1024, R1025, R1026, R1027, R1028, R1029, R1030, R1031, R1032, R1033, R1034, R1035, R1036, R1037, R1038, R1039, R1040, R1041, R1042, R1043, R1044, R1045, R1046, R1047, R1048, R1049, R1050, R1051, R1052, R1053, R1054, R1055, R1056, R1057, R1058, R1059, R1060, R1061, R1062, R1063, R1064, R1065, R1066, R1067, R1068, R1069, R1070, R1071, R1072, R1073, R1074, R1075, R1076, R1077, R1078, R1079, R1080, R1081, R1082, R1083, R1084, R1085, R1086, R1087, R1088, R1089, R1090, R1091, R1092, R1093, R1094, R1095, R1096, R1097, R1098, R1099, R1100, R1101, R1102, R1103, R1104, R1105, R1106, R1107, R1108, R1109, R1110, R1111, R1112, R1113, R1114, R1115, R1116, R1117, R1118, R1119, R1120, R1121, R1122, R1123, R1124, R1125, R1126, R1127, R1128, R1129, R1130, R1131, R1132, R1133, R1134, R1135, R1136, R1137, R1138, R1139, R1140, R1141, R1142, R1143, R1144, R1145, R1146, R1147, R1148, R1149, R1150, R1151, R1152, R1153, R1154, R1155, R1156, R1157, R1158, R1159, R1160, R1161, R1162, R1163, R1164, R1165, R1166, R1167, R1168, R1169, R1170, R1171, R1172, R1173, R1174, R1175, R1176, R1177, R1178, R1179, R1180, R1181, R1182, R1183, R1184, R1185, R1186, R1187, R1188, R1189, R1190, R1191, R1192, R1193, R1194, R1195, R1196, R1197, R1198, R1199, R1200, R1201, R1202, R1203, R1204, R1205, R1206, R1207, R1208, R1209, R1210, R1211, R1212, R1213, R1214, R1215, R1216, R1217, R1218, R1219, R1220, R1221, R1222, R1223, R1224, R1225, R1226, R1227, R1228, R1229, R1230, R1231, R1232, R1233, R1234, R1235, R1236, R1237, R1238, R1239, R1240, R1241, R1242, R1243, R1244, R1245, R1246, R1247, R1248, R1249, R1250, R1251, R1252, R1253, R1254, R1255, R1256, R1257, R1258, R1259, R1260, R1261, R1262, R1263, R1264, R1265, R1266, R1267, R1268, R1269, R1270, R1271, R1272, R1273, R1274, R1275, R1276, R1277, R1278, R1279, R1280, R1281, R1282, R1283, R1284, R1285, R1286, R1287, R1288, R1289, R1290, R1291, R1292, R1293, R1294, R1295, R1296, R1297, R1298, R1299, R1300, R1301, R1302, R1303, R1304, R1305, R1306, R1307, R1308, R1309, R1310, R1311, R1312, R1313, R1314, R1315, R1316, R1317, R1318, R1319, R1320, R1321, R1322, R1323, R1324, R1325, R1326, R1327, R1328, R1329, R1330, R1331, R1332, R1333, R1334, R1335, R1336, R1337, R1338, R1339, R1340, R1341, R1342, R1343, R1344, R1345, R1346, R1347, R1348, R1349, R1350, R1351, R1352, R1353, R1354, R1355, R1356, R1357, R1358, R1359, R1360, R1361, R1362, R1363, R1364, R1365, R1366, R1367, R1368, R1369, R1370, R1371, R1372, R1373, R1374, R1375, R1376, R1377, R1378, R1379, R1380, R1381, R1382, R1383, R1384, R1385, R1386, R1387, R1388, R1389, R1390, R1391, R1392, R1393, R1394, R1395, R1396, R1397, R1398, R1399, R1400, R1401, R1402, R1403, R1404, R1405, R1406, R1407, R1408, R1409, R1410, R1411, R1412, R1413, R1414, R1415, R1416, R1417, R1418, R1419, R1420, R1421, R1422, R1423, R1424, R1425, R1426, R1427, R1428, R1429, R1430, R1431, R1432, R1433, R1434, R1435, R1436, R1437, R1438, R1439, R1440, R1441, R1442, R1443, R1444, R1445, R1446, R1447, R1448, R1449, R1450, R1451, R1452, R1453, R1454, R1455, R1456, R1457, R1458, R1459, R1460, R1461, R1462, R1463, R1464, R1465, R1466, R1467, R1468, R1469, R1470, R1471, R1472, R1473, R1474, R1475, R1476, R1477, R1478, R1479, R1480, R1481, R1482, R1483, R1484, R1485, R1486, R1487, R1488, R1489, R1490, R1491, R1492, R1493, R1494, R1495, R1496, R1497, R1498, R1499, R1500, R1501, R1502, R1503, R1504, R1505, R1506, R1507, R1508, R1509, R1510, R1511, R1512, R1513, R1514, R1515, R1516, R1517, R1518, R1519, R1520, R1521, R1522, R1523, R1524, R1525, R1526, R1527, R1528, R1529, R1530, R1531, R1532, R1533, R1534, R1535, R1536, R1537, R1538, R1539, R1540, R1541, R1542, R1543, R1544, R1545, R1546, R1547, R1548, R1549, R1550, R1551, R1552, R1553, R1554, R1555, R1556, R1557, R1558, R1559, R1560, R1561, R1562, R1563, R1564, R1565, R1566, R1567, R1568, R1569, R1570, R1571, R1572, R1573, R1574, R1575, R1576, R1577, R1578, R1579, R1580, R1581, R1582, R1583, R1584, R1585, R1586, R1587, R1588, R1589, R1590, R1591, R1592, R1593, R1594, R1595, R1596, R1597, R1598, R1599, R1600, R1601, R1602, R1603, R1604, R1605, R1606, R1607, R1608, R1609, R1610, R1611, R1612, R1613, R1614, R1615, R1616, R1617, R1618, R1619, R1620, R1621, R1622, R1623, R1624, R1625, R1626, R1627, R1628, R1629, R1630, R1631, R1632, R1633, R1634, R1635, R1636, R1637, R1638, R1639, R1640, R1641, R1642, R1643, R1644, R1645, R1646, R1647, R1648, R1649, R1650, R1651, R1652, R1653, R1654, R1655, R1656, R1657, R1658, R1659, R1660, R1661, R1662, R1663, R1664, R1665, R1666, R1667, R1668, R1669, R1670, R1671, R1672, R1673, R1674, R1675, R1676, R1677, R1678, R1679, R1680, R1681, R1682, R1683, R1684, R1685, R1686, R1687, R1688, R1689, R1690, R1691, R1692, R1693, R1694, R1695, R1696, R1697, R1698, R1699, R1700, R1701, R1702, R1703, R1704, R1705, R1706, R1707, R1708, R1709, R1710, R1711, R1712, R1713, R1714, R1715, R1716, R1717, R1718, R1719, R1720, R1721, R1722, R1723, R1724, R1725, R1726, R1727, R1728, R1729, R1730, R1731, R1732, R1733, R1734, R1735, R1736, R1737, R1738, R1739, R1740, R1741, R1742, R1743, R1744, R1745, R1746, R1747, R1748, R1749, R1750, R1751, R1752, R1753, R1754, R1755, R1756, R1757, R1758, R1759, R1760, R1761, R1762, R1763, R1764, R1765, R1766, R1767, R1768, R1769, R1770, R1771, R1772, R1773, R1774, R1775, R1776, R1777, R1778, R1779, R1780, R1781, R1782, R1783, R1784, R1785, R1786, R1787, R1788, R1789, R1790, R1791, R1792, R1793, R1794, R1795, R1796, R1797, R1798, R1799, R1800, R1801, R1802, R1803, R1804, R1805, R1806, R1807, R1808, R1809, R1810, R1811, R1812, R1813, R1814, R1815, R1816, R1817, R1818, R1819, R1820, R1821, R1822, R1823, R1824, R1825, R1826, R1827, R1828, R1829, R1830, R1831, R1832, R1833, R1834, R1835, R1836, R1837, R1838, R1839, R1840, R1841, R1842, R1843, R1844, R1845, R1846, R1847, R1848, R1849, R1850, R1851, R1852, R1853, R1854, R1855, R1856, R1857, R1858, R1859, R1860, R1861, R1862, R1863, R1864, R1865, R1866, R1867, R1868, R1869, R1870, R1871, R1872, R1873, R1874, R1875, R1876, R1877, R1878, R1879, R1880, R1881, R1882, R1883, R1884, R1885, R1886, R1887, R1888, R1889, R1890, R1891, R1892, R1893, R1894, R1895, R1896, R1897, R1898, R1899, R1900, R1901, R1902, R1903, R1904, R1905, R1906, R1907, R1908, R1909, R1910, R1911, R1912, R1913, R1914, R1915, R1916, R1917, R1918, R1919, R1920, R1921, R1922, R1923, R1924, R1925, R1926, R1927, R1928, R1929, R1930, R1931, R1932, R1933, R1934, R1935, R1936, R1937, R1938, R1939, R1940, R1941, R1942, R1943, R1944, R1945, R1946, R1947, R1948, R1949, R1950, R1951, R1952, R1953, R1954, R1955, R1956, R1957, R1958, R1959, R1960, R1961, R1962, R1963, R1964, R1965, R1966, R1967, R1968, R1969, R1970, R1971, R1972, R1973, R1974, R1975, R1976, R1977, R1978, R1979, R1980, R1981, R1982, R1983, R1984, R1985, R1986, R1987, R1988, R1989, R1990, R1991, R1992, R1993, R1994, R1995, R1996, R1997, R1998, R1999, R2000, R2001, R2002, R2003, R2004, R2005, R2006, R2007, R2008, R2009, R2010, R2011, R2012, R2013, R2014, R2015, R2016, R2017, R2018, R2019, R2020, R2021, R2022, R2023, R2024, R2025, R2026, R2027, R2028, R2029, R2030, R2031, R2032, R2033, R2034, R2035, R2036, R2037, R2038, R2039, R2040, R2041, R2042, R2043, R2044, R2045, R2046, R2047, R2048, R2049, R2050, R2051, R2052, R2053, R2054, R2055, R2056, R2057, R2058, R2059, R2060, R2061, R2062, R2063, R2064, R2065, R2066, R2067, R2068, R2069, R2070, R2071, R2072, R2073, R2074, R2075, R2076, R2077, R2078, R2079, R2080, R2081, R2082, R2083, R2084, R2085, R2086, R2087, R2088, R2089, R2090, R2091, R2092, R2093, R2094, R2095, R2096, R2097, R2098, R2099, R2100, R2101, R2102, R2103, R2104, R2105, R2106, R2107, R2108, R2109, R2110, R2111, R2112, R2113, R2114, R2115, R2116,
```

# Prefix Sum

## Prefix Sum

The prefix sum is used for *compaction*.

Given: kernel  $K$  which may or may not produce output for further processing.



```

...
    & (depth < MAXDEPTH)
...
    inside / inside;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0));
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( @rand, I, R, Alignment
    e.x + radiance.y + radiance.z) > 0) && (rand
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
    random walk - done properly, closely following
    vive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



# Prefix Sum

## Prefix Sum - Compaction

Given: kernel K which may or may not produce output for further processing.

```

...
(depth < MAXDEPTH)
{
    ...
    inside / ...
    nt = nt / ...
    pos2t = 1.0f - ...
    D, N );
}

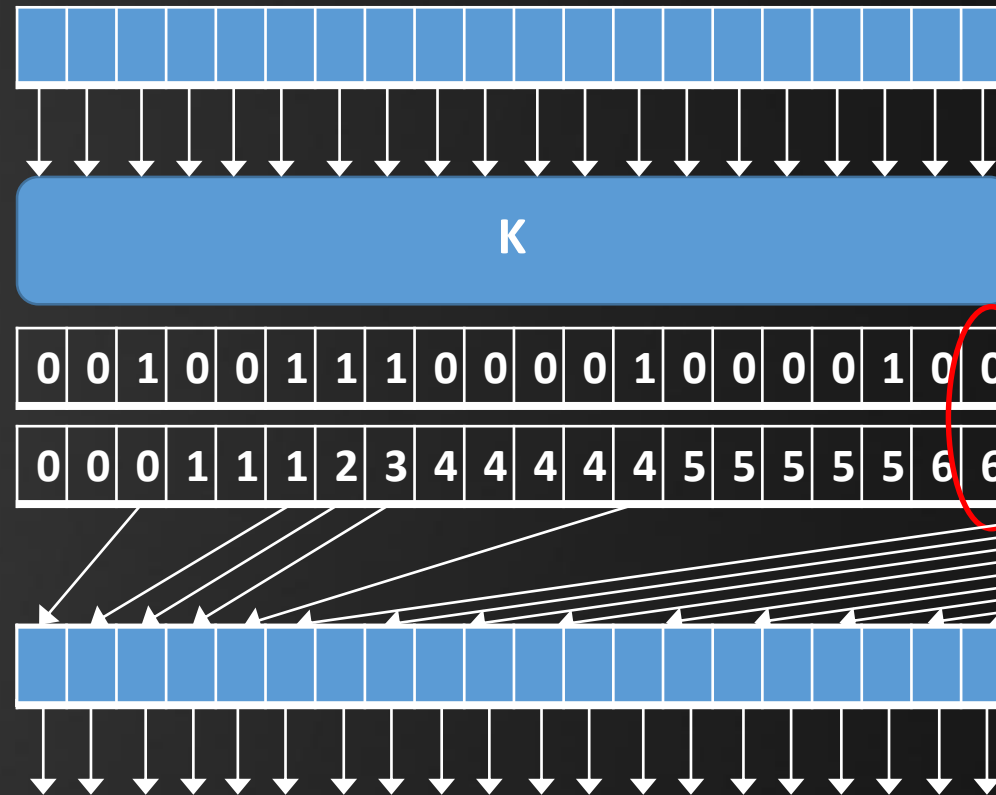
at a = nt - nc, b = nt - ...
at Tr = 1 - (R0 + (1 - R0) ...
Tr) R = (D * nnt - N * ...

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;

MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, ...
if;
radiance = SampleLight( &rand, I, R1, ...
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * ...
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following ...
ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf ...
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



output array size

boolean array

exclusive prefix sum

output array



# Prefix Sum

## Prefix Sum

```

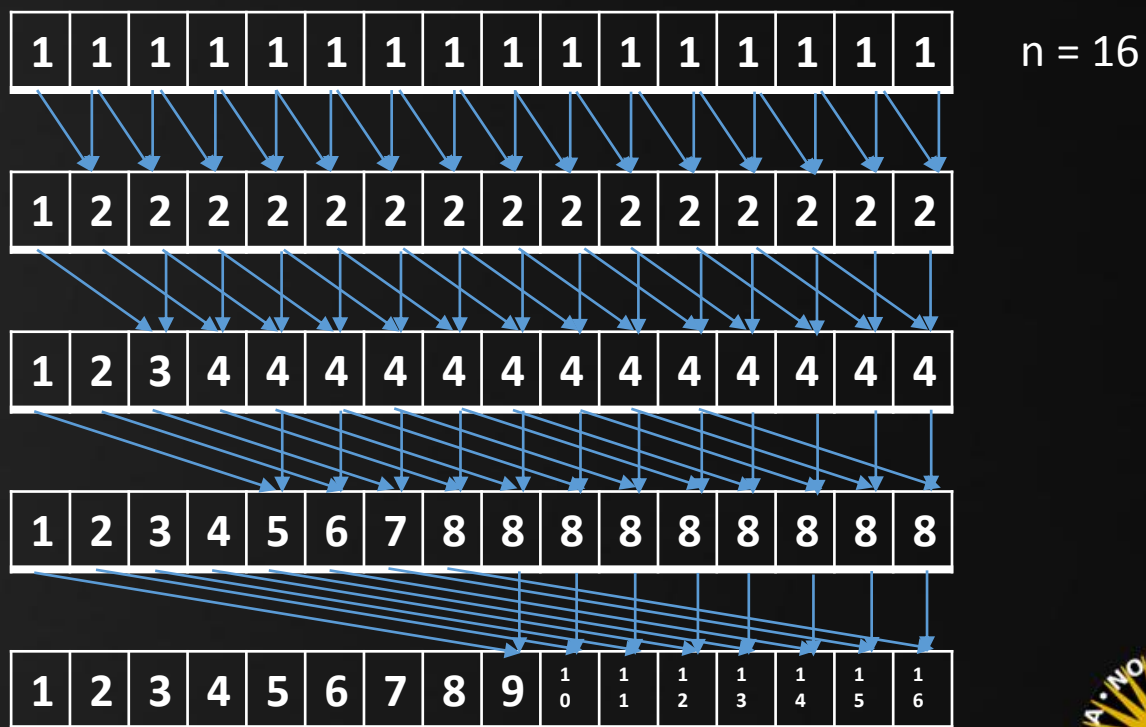
out[0] = 0;
for ( i = 1; i < n; i++ ) out[i] = in[i-1] + out[i-1];
    
```

In parallel:

```

for ( d = 1; d <= log2n; d++ )
    for all k in parallel do
        if k >= 2d-1
            x[k] += x[k - 2d-1]
    
```

- Each thread in the warp reads data
- Each thread in the warp sums 2 input elements
- Each thread in the warp writes data.



# Prefix Sum

## Prefix Sum

```

out[0] = 0;
for ( i = 1; i < n; i++ ) out[i] = in[i-1] + out[i-1];
    
```

In parallel:

```

for ( d = 1; d <= log2n; d++ )
    for all k in parallel do
        if k >= 2d-1
            x[k] += x[k - 2d-1]
    
```

For each pass:

- Each thread in the warp reads data
- Each thread in the warp sums 2 input elements
- Each thread in the warp writes data.

Notes:

- The scan happens in-place. This is only correct if we have 32 input elements, and the scan is done in a single warp. Otherwise we need to double buffer for correct results.
- Span of the algorithm is  $\log n$ , but work is  $n \log n$ ; it is not work-efficient. Efficient algorithms for large inputs can be found in:

Meril & Garland, 2016, Single-pass Parallel Prefix Scan with Decoupled Look-back.



# Prefix Sum

## Prefix Sum

```

out[0] = 0;
for ( i = 1; i < n; i++ ) out[i] = in[i-1] + out[i-1];

```

## In OpenCL:

```

int scan_exclusive( __local int* input, int lane )
{
    if (lane > 0 ) input[lane] = input[lane - 1] + input[lane];
    if (lane > 1 ) input[lane] = input[lane - 2] + input[lane];
    if (lane > 3 ) input[lane] = input[lane - 4] + input[lane];
    if (lane > 7 ) input[lane] = input[lane - 8] + input[lane];
    if (lane > 15) input[lane] = input[lane - 16] + input[lane];
    return (lane > 0) ? input[lane - 1] : 0;
}

```





# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



# Sorting

## GPU Sorting

### Observation:

- We frequently need sorting in our algorithms.

### But:

- Most sorting algorithms are divide and conquer algorithms.

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (1 + 0.0001 * depth);
    nt = nt / nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * nnt);
    Tr) R = (D * nnt - N * (1 - nnt));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse );
estimation - doing it properly, close to
if;
radiance = SampleLight( &rand, I, &t, &align,
e.x + radiance.y + radiance.z) > 0) && (survive);
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



# Sorting

## GPU Sorting: Selection Sort

```

__kernel void Sort( __global int* in, __global int* out )
{
    int i = get_global_id( 0 );
    int n = get_global_size( 0 );
    int iKey = in[i];
    // compute position of in[i] in output
    int pos = 0;
    for( int j = 0; j < n; j++ )
    {
        int jKey = in[j]; // broadcasted
        bool smaller = (jKey < iKey) || (jKey == iKey && j < i);
        pos += (smaller) ? 1 : 0;
    }
    out[pos] = iKey;
}

```



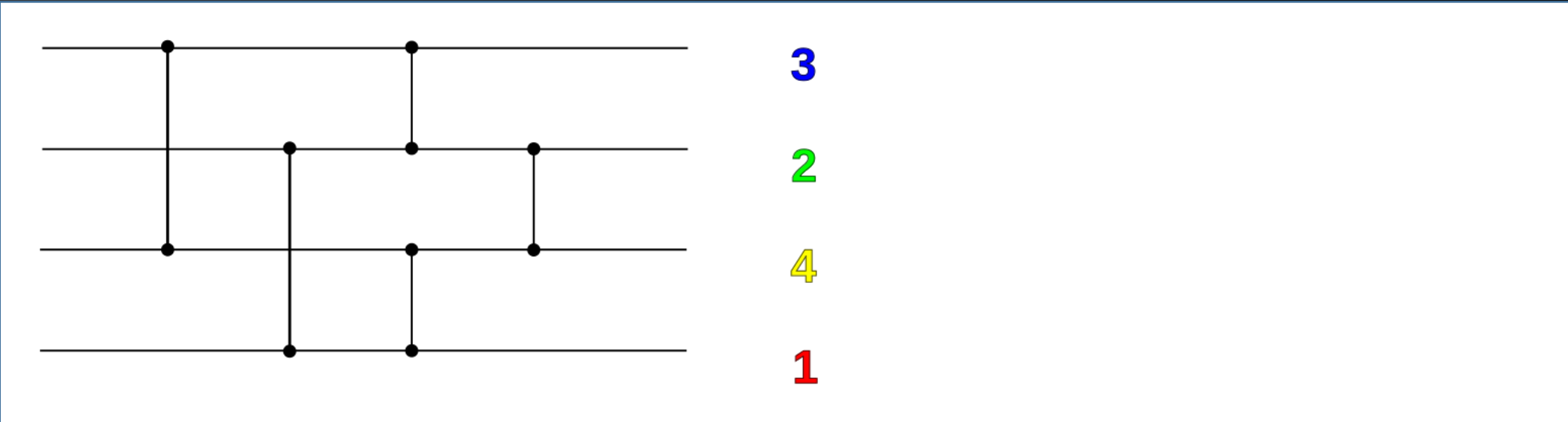
# Sorting

## GPU Sorting

```

...ics
& (depth < MAXDEPTH)
...
c = inside / (1 + depth);
nt = nt / nc; dbd = dbd / dbd;
cos2t = 1.0f - nnt;
D, N );
...
)
...
at a = nt - nc; b = nt;
at Tr = 1 - (R0 + (1 - R0));
Tr) R = (D * nnt - N * (a
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH)
D, N );
-efl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability(
estimation - doing it properly.
if;
-radiance = SampleLight( &rand,
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



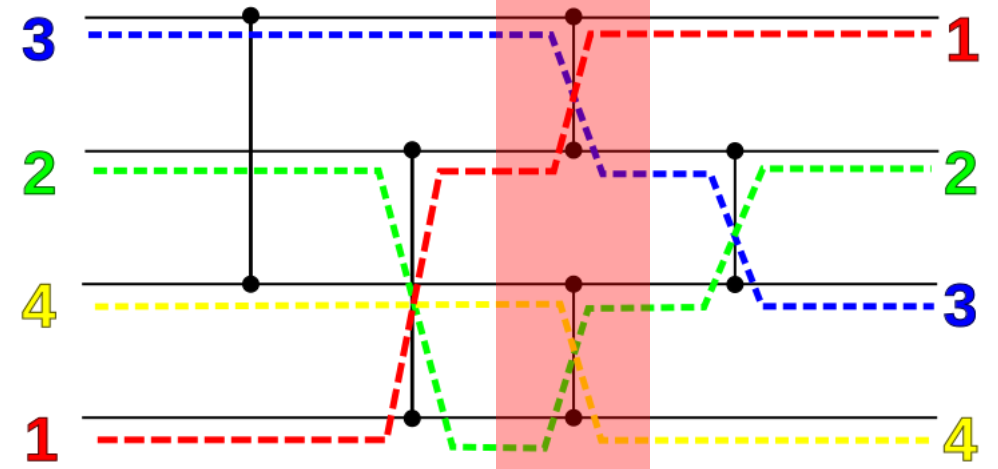
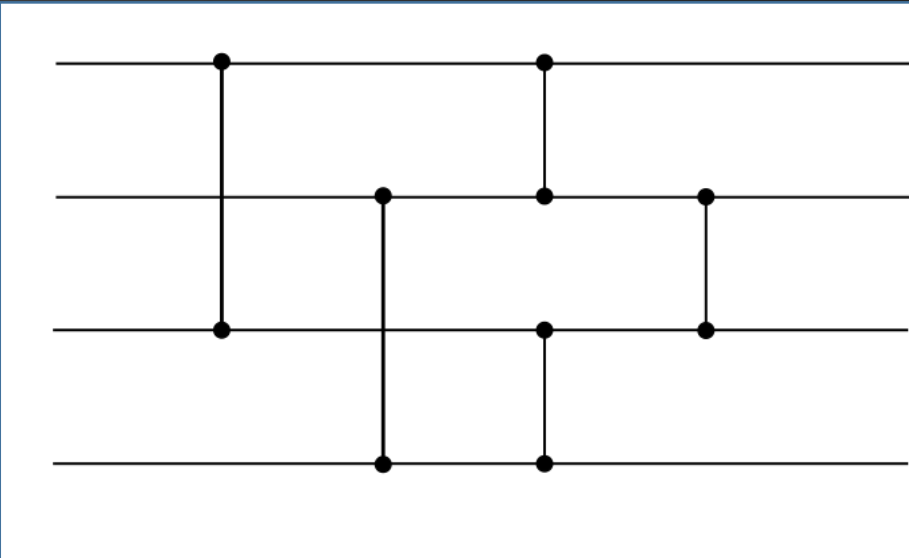
# Sorting

## GPU Sorting

```

...ics
& (depth < MAXDEPTH)
...
c = inside / inside;
nt = nt / nc;
cos2t = 1.0f - nnt;
D, N );
)
...
at a = nt - nc; b = nt;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (a
...
E * diffuse;
= true;
...
efl + refr) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability(
estimation - doing it properly);
if;
radiance = SampleLight( &rand,
e.x + radiance.y + radiance.z) > 0) && (depth <
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

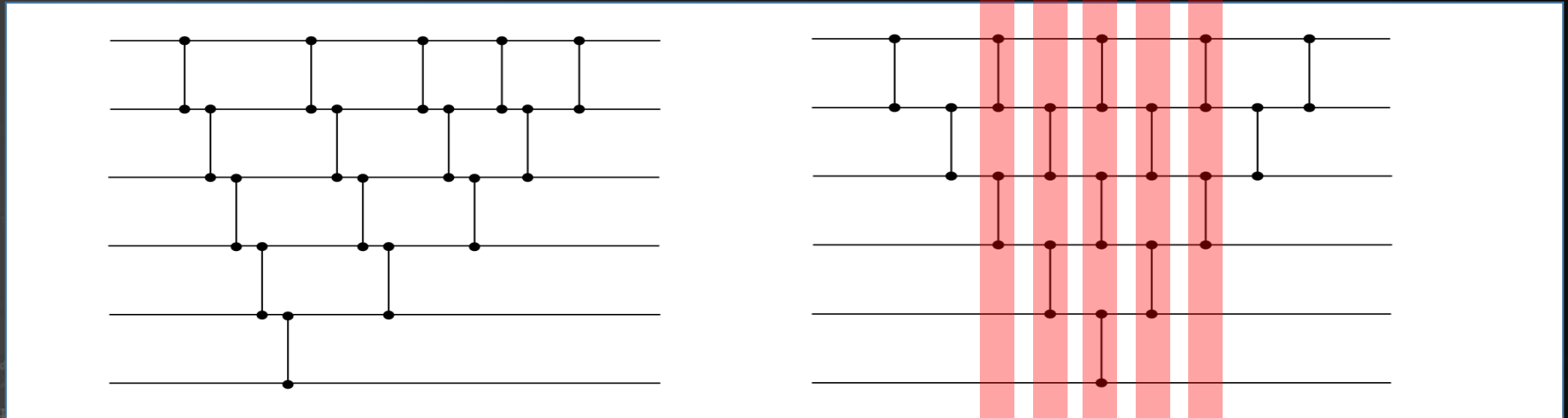
```



# Sorting

## GPU Sorting

### Bubblesort:



Size: number of comparisons (in this case:  $5 + 4 + 3 + 2 + 1 = 15$ )

Depth: number of sequential steps (in this case: 9)

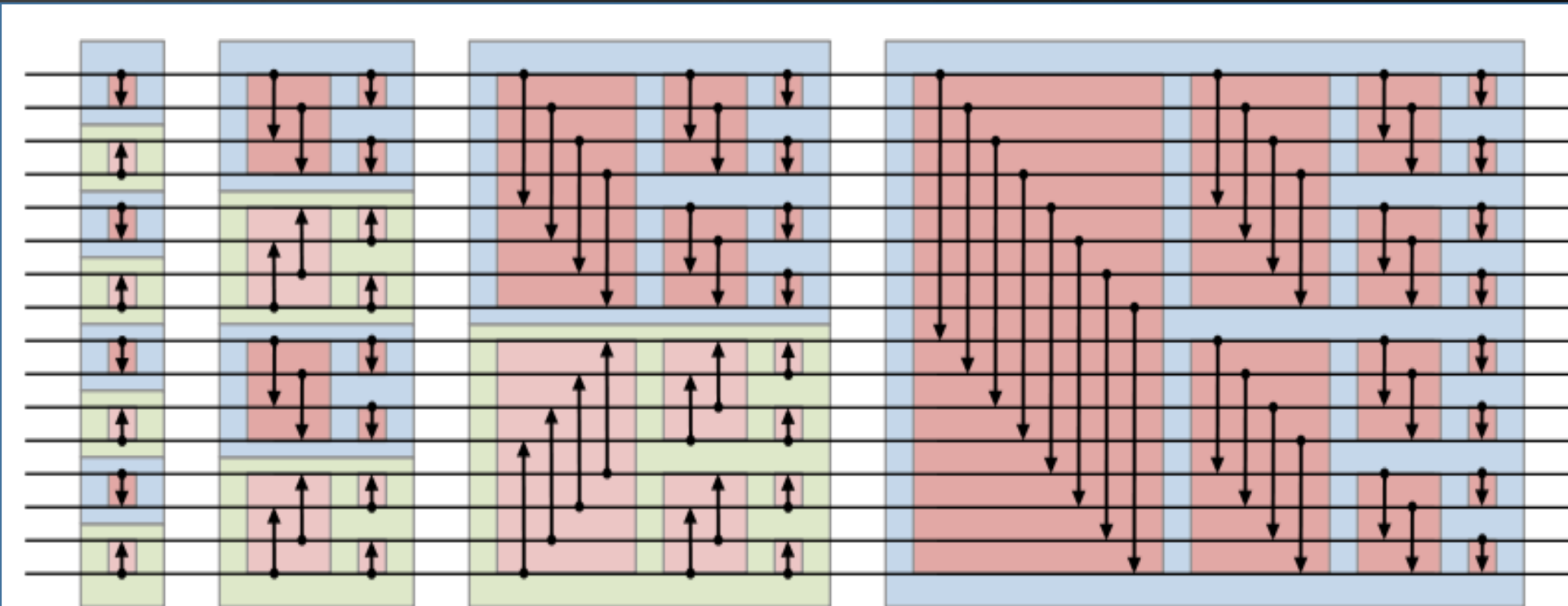


# Sorting

## GPU Sorting

Bitonic sort<sup>\*,\*\*</sup>:

- Work:  $n \log(n)^2$
- Span:  $\log(n)^2$



- Compare element in top half with element in bottom half
- Subdivide red box and recurse until a single comparison is left

*All boxes can execute in parallel.*

```

...
(depth)
...
inside
nt = nt / n
os2t = 1.0f
D, N );
...
a = nt -
at Tr = 1 -
Tr) R = (D
...
diffuse
= true;
...
refl + refr)
...
D, N );
refl * E *
= true;
...
MAXDEPTH)
survive = S
estimation
if;
-radiance =
e.x + radia
...
w = true;
st brdfPdf
st3 factor
st weight =
st cosTheta
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following the
(ive)
...
st3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, spdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```

\*: Batcher, '68, Sorting Networks and their Applications.

\*\* : Bitonic Sorting Network for n Not a Power of 2;

<http://www.itl.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm>



# Sorting

## GPU Sorting

You can find an implementation of the bitonic sort in the OpenCL template:

```
cl_int Buffer::ParallelSort()
```

This replaces the contents of a buffer with the sorted values.

```

...
    & (depth < MAXDEPTH)
...
    c = inside / (1 + depth);
    nt = nt / nc;
    nnt = 1.0f - nnt;
    D, N );
}

...
    at a = nt - nc;
    at Tr = 1 - (R0 + (1 - R0));
    Tr) R = (D * nnt - N * (1 - nnt));
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &align, &dir, &norm, &pos, &rot );
e.x + radiance.y + radiance.z) > 0) && (rand.N < survive);
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance.x + radiance.y + radiance.z);
...
random walk - done properly, closely following
survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    r1 = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



# Sorting

## Take-away:

GPGPU requires massive parallelism. Algorithms that do not exhibit this need to be replaced.

The parallel scan is an important ingredient that serves as a building block for larger algorithms, or between kernels.

```

...
    & (depth < MAXDEPTH)
...
    c = inside / (1 + refl);
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a0
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( @rand, I, @t, @align;
e.x + radiance.y + radiance.z) > 0) && (rand
...
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiant
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



# Compaction

## Stream Filtering

```
for ( int i = 0; i < items; i++ )
{
    // do something elaborate, 'items' can be 0..10
}

```

```
void ComplexTask( int taskID )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // do additional work
    }
}

```



# Compaction

## Stream Filtering

```
bool needsAdditionalWork[...];
void ComplexTaskPart1( int taskID )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // do additional work
        needsAdditionalWork[taskID] = true;
    }
}
```

```
void ComplexTaskPart2( int taskID )
{
    if (needsAdditionalWork[taskID])
    {
        ...
    }
}
```

```
void ComplexTask( int taskID )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // do additional work
    }
}
```



# Compaction

## Stream Filtering

```
void ComplexTaskPart1( int taskID,
    __global int* taskIDs,
    __global int* taskCount )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // schedule additional work
        taskIDs[taskCount++] = taskID;
    }
}
```

```
void ComplexTaskPart2( int idx )
{
    DoWork( taskIDs[idx] );
}
```

```
void ComplexTask( int taskID )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // do additional work
    }
}
```



# Compaction

## Stream Filtering

```

void ComplexTaskPart1( int taskID,
    __global int* taskCount,
    __global int* taskIDs )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // schedule additional work
        int arrayIdx = atomic_add( taskCount, 1 );
        taskIDs[arrayIdx] = taskID;
    }
}

void ComplexTaskPart2( int idx )
{
    DoWork( taskIDs[idx] );
}
    
```

```

void ComplexTask( int taskID )
{
    // do generic work
    ...
    if (condition == true) // true 50% of the time
    {
        // do additional work
    }
}
    
```

## Reducing the number of atomics:

- Store ‘1’ or ‘0’ in an array depending on ‘condition’;
- Do a prefix sum over this array;
- Do a single atomic\_add, which yields the base index;
- Use the values in the array as offsets to this base index.



# Compaction

## Stream Filtering

Stream filtering is used in *multi-pass kernels*.

Examples:

- 10k threads need to find an element in a linked list or a tree
- 10k threads trace a path from the camera to the light
- 10k threads update tanks and decide if the tank needs to fire

In all cases, the conditional code is executed by a continuous set of threads.

Compaction is used to restore *occupancy*.

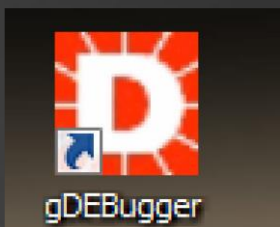


# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



# Optimizing GPGPU



The screenshot shows the gDEBugger CL interface for debugging a process named 'oc\_lab'. The interface includes several panels:

- Function Calls History:** Shows a list of function calls.
- Calls Stack:** Displays the current call stack, including '0x012e231f - oc\_lab.exe' and system DLLs like 'kernel32.dll' and 'ntdll.dll'.
- Debugged Process Events:** Lists various events such as 'Thread Terminated' and 'Process Exit'.
- Performance Graph:** A line graph showing performance metrics over time (0:07 to 0:24).
- Counter Name Table:** A table listing performance counters and their values.
 

Counter Name	Value	Scale
GDB GL Frames/sec: GL Context 1	0	0 [1]
OS CPUs Average Utilization	15	15 [1]
GDB CL Kernel Commands Utilization: CL Context 1 Queue 1	89	89 [1]
GDB CL Write Commands Utilization: CL Context 1 Queue 1	0	0 [1/]
GDB CL Read Commands Utilization: CL Context 1 Queue 1	0	0 [1/]
GDB CL Queue Busy: CL Context 1 Queue 1	90	90 [1]
GDB CL Queue Idle: CL Context 1 Queue 1	10	10 [1]
GDB CL Work items/sec: CL Context 1 Queue 1	55...	55,6...
- Performance Dashboard:** A bar chart showing performance metrics for different components: Frame..., CPUs..., Kerne..., Write..., Read..., Queue..., Queue..., and Work... with values ranging from 0.0 to 109M.



# Optimizing GPGPU

gDEBugger CL - OpenCL Command Queues Viewer

Command Name	Parameters	Duration (ms)	Accumulated Time (ms)	Wait for Submit (ms)	Wait for Execution (ms)
NDRangeKernel	P4 Kernel 1 (dearGrid)	0.293024	0.293024	4.117472	0.053120
NDRangeKernel	P5 Kernel 1 (fillGrid)	2.422400	2.715424	0.043840	0.033216
AcquireGLObjets	Image 1	0.000000	2.715424	0.000000	0.000000
NDRangeKernel	P3 Kernel 1 (render2)	5.932416	8.647840	0.037440	0.037376
ReleaseGLObjets	Image 1	0.000000	8.647840	0.000000	0.000000
AcquireGLObjets	Image 1	0.000000	8.647840	0.000000	0.000000
NDRangeKernel	P1 Kernel 1 (dear)	2.637280	11.285120	0.042272	0.228288
ReleaseGLObjets	Image 1	0.000000	11.285120	0.000000	0.000000
NDRangeKernel	P6 Kernel 1 (simulate1)	1.011360	12.296480	0.030368	0.039968
NDRangeKernel	P4 Kernel 1 (dearGrid)	0.283648	12.580128	0.030944	0.530304
NDRangeKernel	P5 Kernel 1 (fillGrid)	2.423712	15.003840	0.048512	0.514304
CopyBuffer	Buffer 1 -> Buffer 2	0.479808	15.483648	0.061024	0.038240
NDRangeKernel	P8 Kernel 1 (simulateX)	23.699552	39.183200	0.041696	0.030400
NDRangeKernel	P4 Kernel 1 (dearGrid)	0.286400	39.469600	0.055392	0.042784
NDRangeKernel	P5 Kernel 1 (fillGrid)	2.394496	41.864096	0.045376	0.030656
AcquireGLObjets	Image 1	0.000000	41.864096	0.000000	0.000000
NDRangeKernel	P3 Kernel 1 (render2)	6.064768	47.928864	0.048928	0.033504
ReleaseGLObjets	Image 1	0.000000	47.928864	0.000000	0.000000
AcquireGLObjets	Image 1	0.000000	47.928864	0.000000	0.000000
NDRangeKernel	P1 Kernel 1 (dear)	2.611840	50.540704	0.059072	0.231744
ReleaseGLObjets	Image 1	0.000000	50.540704	0.000000	0.000000
NDRangeKernel	P6 Kernel 1 (simulate1)	1.011456	51.552160	0.038048	0.031424

Properties View: NDRangeKernel - P5 Kernel 1 (fillGrid)  
 Duration: 2,394,496 ns  
 Waited for Submission: 45,376 ns  
 Waited for Execution: 30,656 ns  
 Kernel: Program 5 - Kernel 1  
 Work Dimensions: 1  
 Global Work Offset: (0)  
 Global Work Size: 200000  
 Local Work Size: 0  
 Event Wait List: Empty  
 Event: None

Graph View: Device 3 C1 Q1 | ND... | NDRangeKernel - P5 Kern 1 (f... | NDRangeKernel - P3 Kern 1 (render2)

Performance Dashboard

Metric	Value
Frame...	0.0
CPUs ...	10.9
Kerne...	86.5
Write...	0.0
Read ...	0.0
Queue...	87.7
Queue...	12.3
Work ...	86.7%

Properties: Process Not Running  
 To start debugging your application, press the Start Debugging button (F5)



# Optimizing GPGPU

The screenshot displays the gDEBugger CL interface, which is used for debugging OpenCL applications. The main window, titled "gDEBugger CL - Textures, Buffers and Images viewer", shows the "CL Context 1 (shared - GL2)" selected. The "Images list" pane on the left shows "Image 1 (GL2 Tex1)". The "Buffers list" pane shows "OpenCL Buffers" including "CL Buffer 1", "CL Buffer 2", and "CL Buffer 3". The "Properties view" pane shows details for "CL Buffer 3", including its name, handle (0x09b06250), size (40,500KB), and flags (CL MEM READ WRITE). The central pane displays a grid of 22 rows and 3 columns (X, Y, Z) with numerical values. The "Information panel" at the bottom shows "Buffer Format: Vertex", "V3F", "Offset: 0", and "Stride: 0". On the right side, there are several performance monitoring windows, including a "Timing" window and a bar chart showing metrics like "Frame...", "CPUs...", "Kerne...", "Write...", "Read...", "Queue...", "Queue...", and "Work...". The bar chart shows values of 87.7, 12.3, and 86.78. At the bottom left, there is a code editor showing a snippet of OpenCL code:

```

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, spdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



# Optimizing GPGPU

## Faster OpenCL

### 1. Optimize memory usage

- Read data from global memory once
- Use local memory when possible
- Careful: reading the same global address in 32 threads is not a good idea!

```
temp = input[3] // input is in global mem

Instead, use:

if (get_local_id(0) == 0) local = input[3]
barrier(CLK_LOCAL_MEM_FENCE);
temp = local
```

### 2. Make sure there is enough work to hide latency

- On AMD: use multiples of 64 threads (called a ‘wavefront’)
- Tweak manually for performance, ideally per vendor / device

### 3. Minimize the number of host-to-device transfers, then their size

### 4. Minimize the number of kernel invocations

<http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/opencl-optimization-guide>



# Optimizing GPGPU

## Faster OpenCL

### Smaller things:

- Use float4 whenever possible
- Use predication rather than control flow
- Bypass short-circuiting
- Remove conditional code
- AOS vs SOA performance
- Reducing atomics
- Reduced precision math
- Pinned memory

```
pinned = clCreateBuffer( Kernel::GetContext(),
    CL_MEM_ALLOC_HOST_PTR | CL_MEM_WRITE_ONLY,
    sizeof( myData ), 0, 0 );
```

```
native_log
native_exp
native_sqrt
native_sin
native_pow
...
```

```
If (A>B) C += D; else C -= D;

Replace this with:

int factor = (A>B) ? 1:-1;
C += factor*D;
```

```
if(a&& b&& c&& d){...}

becomes

bool cond = a&& b&& c&& d;
if(cond){...}
```

```
if(x==1) r=0.5;
if(x==2) r=1.0;

becomes

r = select(r, 0.5, x==1);
r = select(r, 1.0, x==2);
```



# Today's Agenda:

- Introduction
- The Prefix Sum
- Parallel Sorting
- Stream Filtering
- Optimizing GPU code



/INFOMOV/

END of “GPGPU (3)”

next lecture: TBD

```
ics
& (depth < MAXDEPTH)
{
    t = inside / (1 + 100 * depth);
    nt = nt / nc;
    nnt = nnt * nt;
    D, N );
}

at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * (a * t + b * (1 - t)));

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
D, N );
-efl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, clearly
if;
radiance = SampleLight( &rand, I, &t, &align, &dir, &norm, &pos, &rot, &mat, &obj );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance.x + radiance.y + radiance.z);

random walk - done properly, closely following the path of the photon
ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

