Chapter 4: Parallel Program Structures III

Elements of Parallel Computing

Eric Aubanel

Single Program Multiple Data (SPMD)

- Applies to shared memory programming and distributed memory programming
- Fully explicit programming model: programmer specifies work for each thread (or process), based on its *id*
  - shared memory: explicit synchronization
  - distributed memory: explicit communication

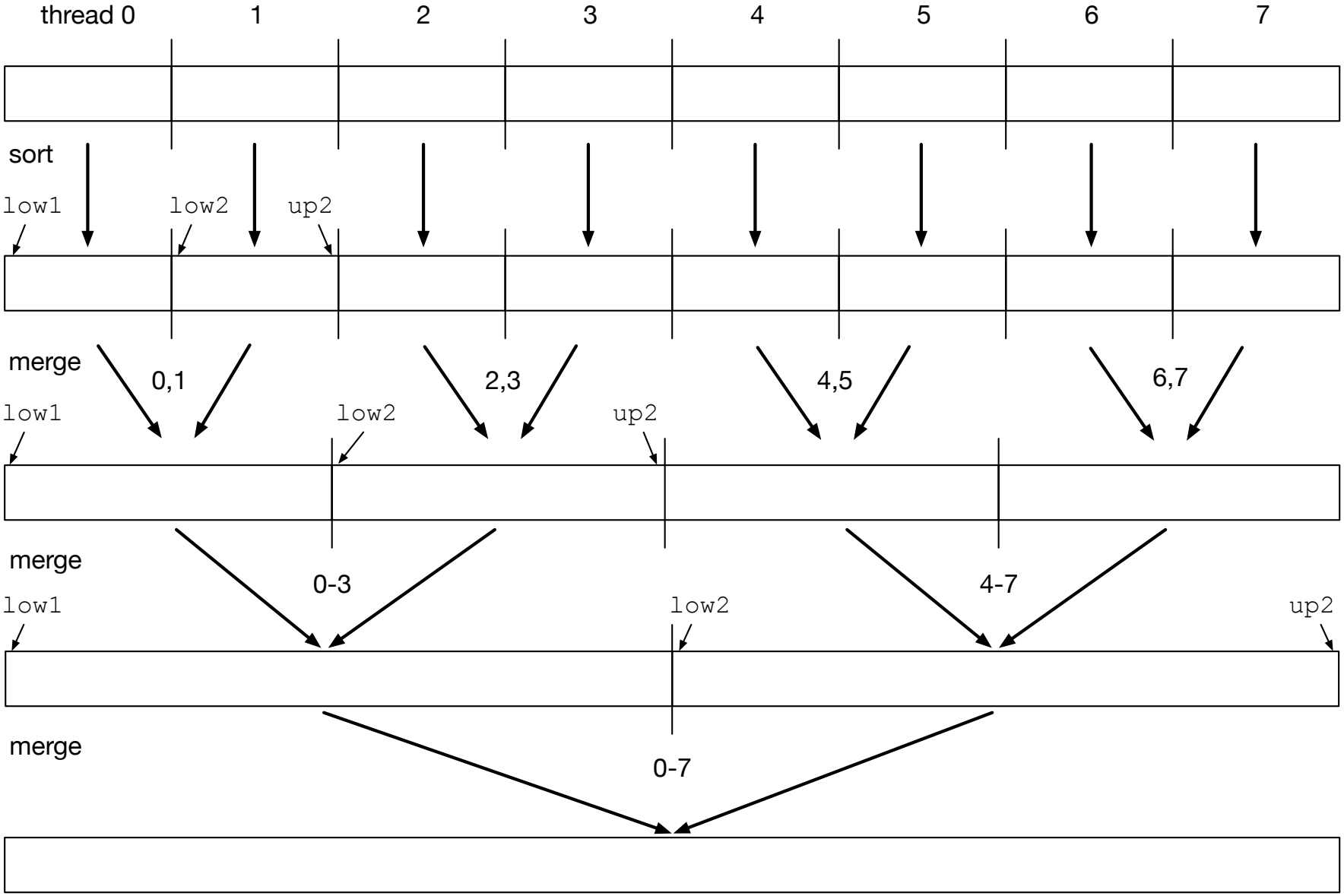
SPMD Matrix-Vector Multiplication

shared a, b, c
$istart \leftarrow \lfloor id * nrows/nt \rfloor$
$iend \leftarrow \lfloor (id+1) * nrows/nt \rfloor - 1$
for $i \leftarrow istart$ *to* $iend$ **do**
    $c[i] \leftarrow 0$
    foreach *column* j *of* a **do**
        $c[i] \leftarrow c[i] + a[i,j] * b[j]$
    end
end

SPMD Fractal

```
// chunk << n is chunk size of round-robin
   assignment of rows to threads
shared kount // array of pixel values is shared
istart ← id * chunk
iend ← (id + 1) * chunk − 1
ax ← len/n
ymax ← ymin + len
while istart < n do
    for i ← istart to iend do
        cx ← ax * i + xmin
        ...
    end
    istart ← istart + nt * chunk
    iend ← min(istart + chunk − 1, n − 1)
end
```

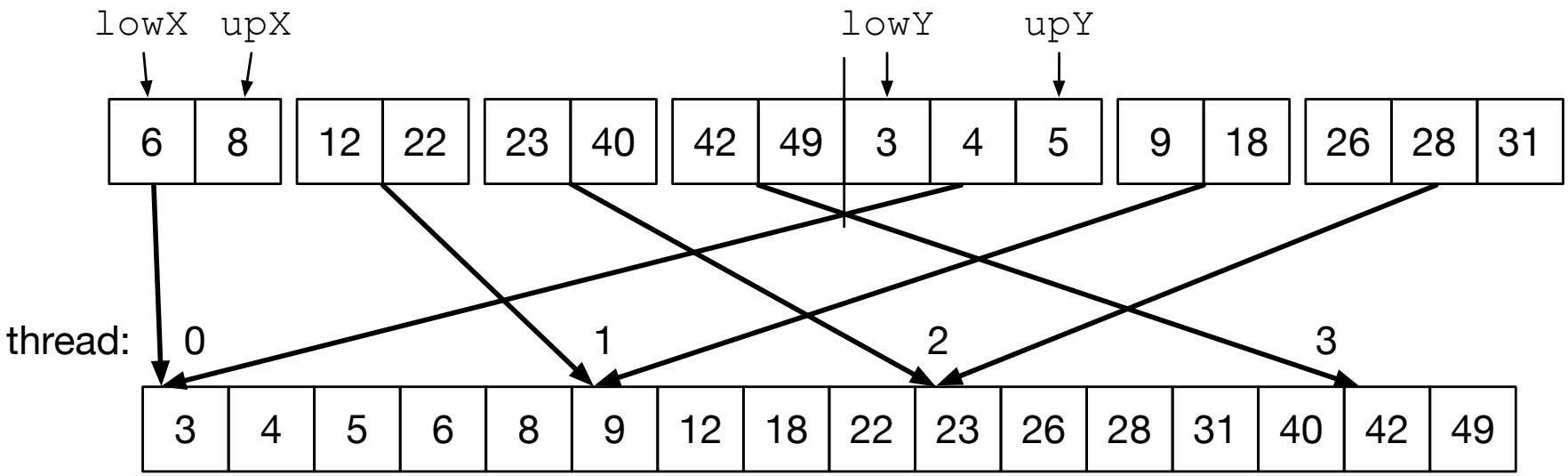
SPMD Merge Sort



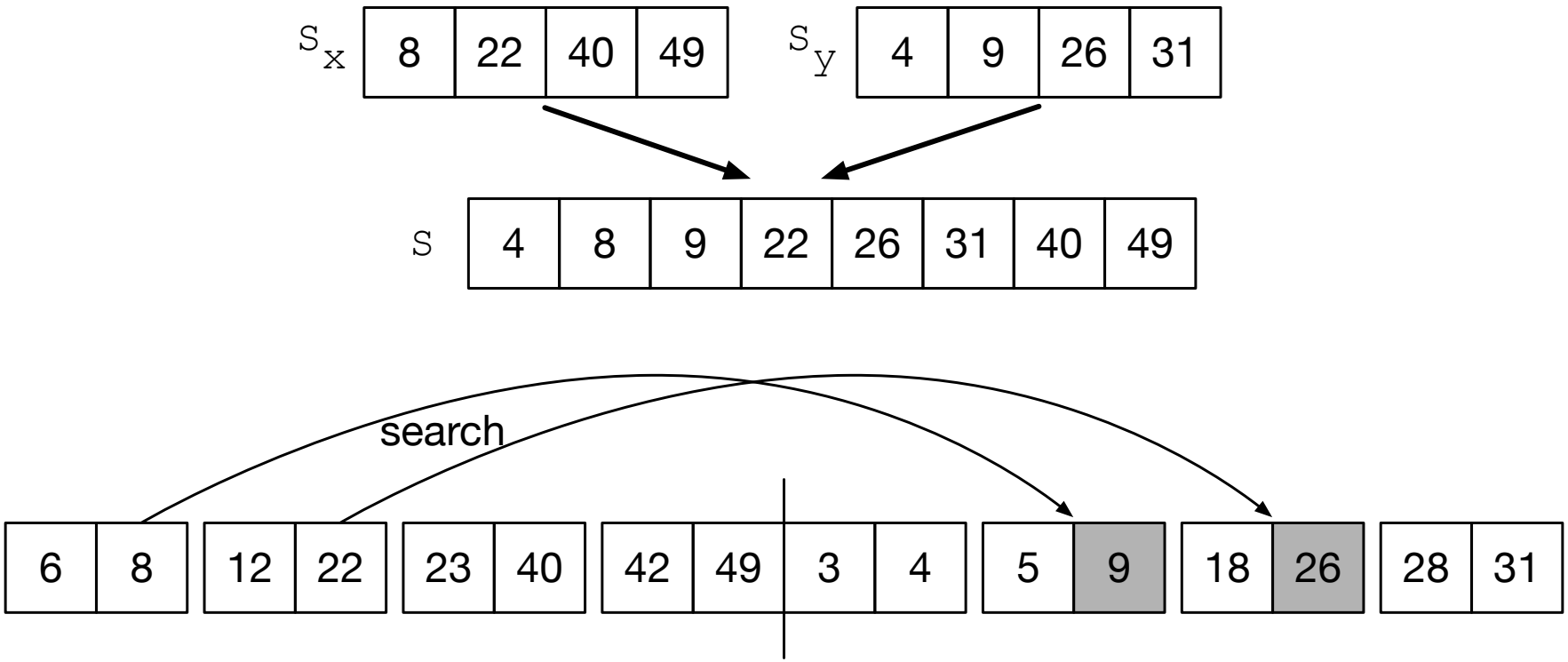
```
Procedure spmdMergeSort(a, b)
    shared a, b
    lower ← ⌊id * n/nt⌋, upper ← ⌊(id + 1) * n/nt⌋
    sequentialSort(a, lower, upper, b)
    barrier()
    nmt ← 1
    for i ← 1 to log nt do
        Swap references to a and b
        chunk ← nmt * n/nt
        nmt ← nmt * 2
        idc ← ⌊id/nmt⌋ * nmt // threads id = idc to
            idc + nmt − 1 do each merge
        low1 ← idc * n/nt, low2 ← low1 + chunk
        up2 ← low2 + chunk − 1
        // nmt threads merge a[low1, low2) with
            a[low2, up2] into b starting at index low1
        spmdMerge(a, low1, low2, up2, b, nmt)
        barrier()
    end
```

```
Procedure spmdMerge(a, low1, low2, up2, b, nmt)
    shared a, b
    idm ← id mod nmt, lowX ← ⌊idm * n/(2 * nt)⌋ + low1
    upX ← ⌊(idm + 1) * n/(2 * nt)⌋ + low1 − 1
    if idm ≠ 0 then
        lowY := binarySearch(a, low2, up2 + 1, lowX − 1)
    else
        lowY ← low2
    end
    if idm < nmt − 1 then
        upY ← binarySearch(a, lowY, up2 + 1, upX) − 1
    else
        upY ← up2
    end
    start ← lowX + lowY − low2
    // merges a[lowX, upX] with a[lowY, upY] into b
       starting at index start
    sequentialMerge(a, lowX, upX + 1, lowY, upY + 1, b,
    start)
```

SPMD Merge



Improved Merge



GPU Programming

- SPMD model: write program for single thread, based on id.
- Also specify organization of threads into groups
- Threads in a group share local memory
- Barrier synchronization enabled for threads in a group
- Threads in each block further grouped into warps during execution
- SIMT execution per warp

Reduction on GPU

```
Procedure reduceGPU(a)
    // set group size and number of groups and
       allocate memory on GPU
    ...
    reduceToGroup(a, c)
    sum ← 0
    foreach item in c do
        sum ← sum + item
    end
    return sum
end
```

```
Procedure reduceToGroup(a, c)
    shared b // shared among threads in group
    tid ← getThreadID(), gid ← getGroupID()
    grpSz ← getGroupSize()
    nt ← grpSz * getNumGroups(), id ← gid * grpSz + tid
    istart ← ⌊id * n/nt⌋, iend ← ⌊(id + 1) * n/nt⌋ − 1
    psum ← 0
    for i ← istart to iend do
        psum ← psum + a[i]
    end
    b[tid] ← psum
    syncGroup()
    for k ← log grpSz − 1 to 0 do
        j ← 2^k
        if tid < j then b[i] ← b[i] + b[i + j]
        syncGroup()
    end
    c[gid] ← b[0]
end
```