

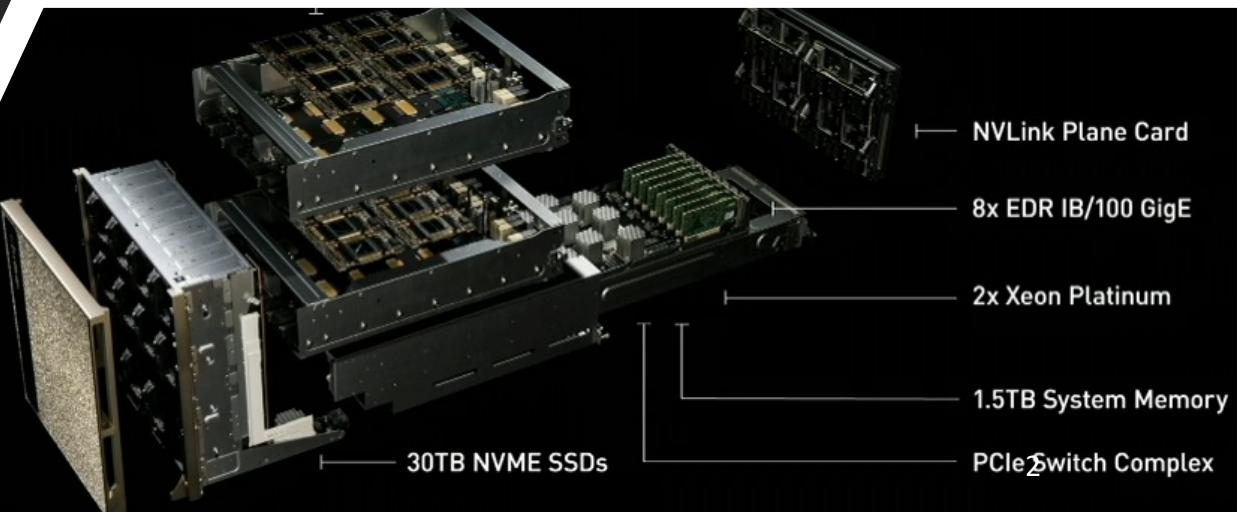
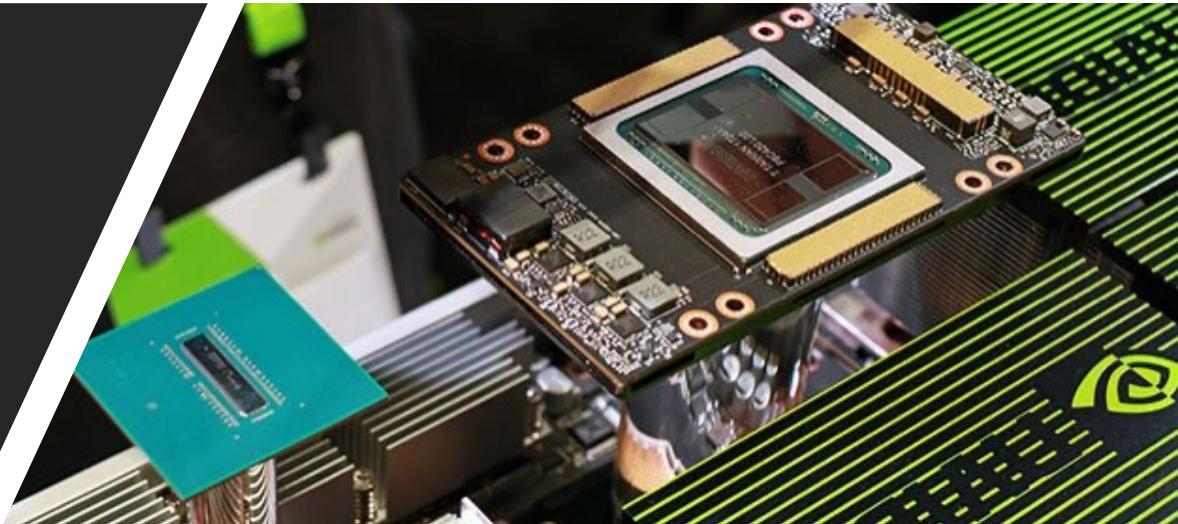
Parallel Architectures Parallel Algorithms CUDA

Chris Rossbach

cs378h

Outline for Today

- Questions?
- Administrivia
 - pedagogical-* machines should be available
- Agenda
 - Parallel Algorithms
 - CUDA
- Acknowledgements:
http://developer.download.nvidia.com/compute/developertools/prerequisites/presentations/cuda_language/Introduction_to_CUDA_C.pptx



Faux Quiz Questions

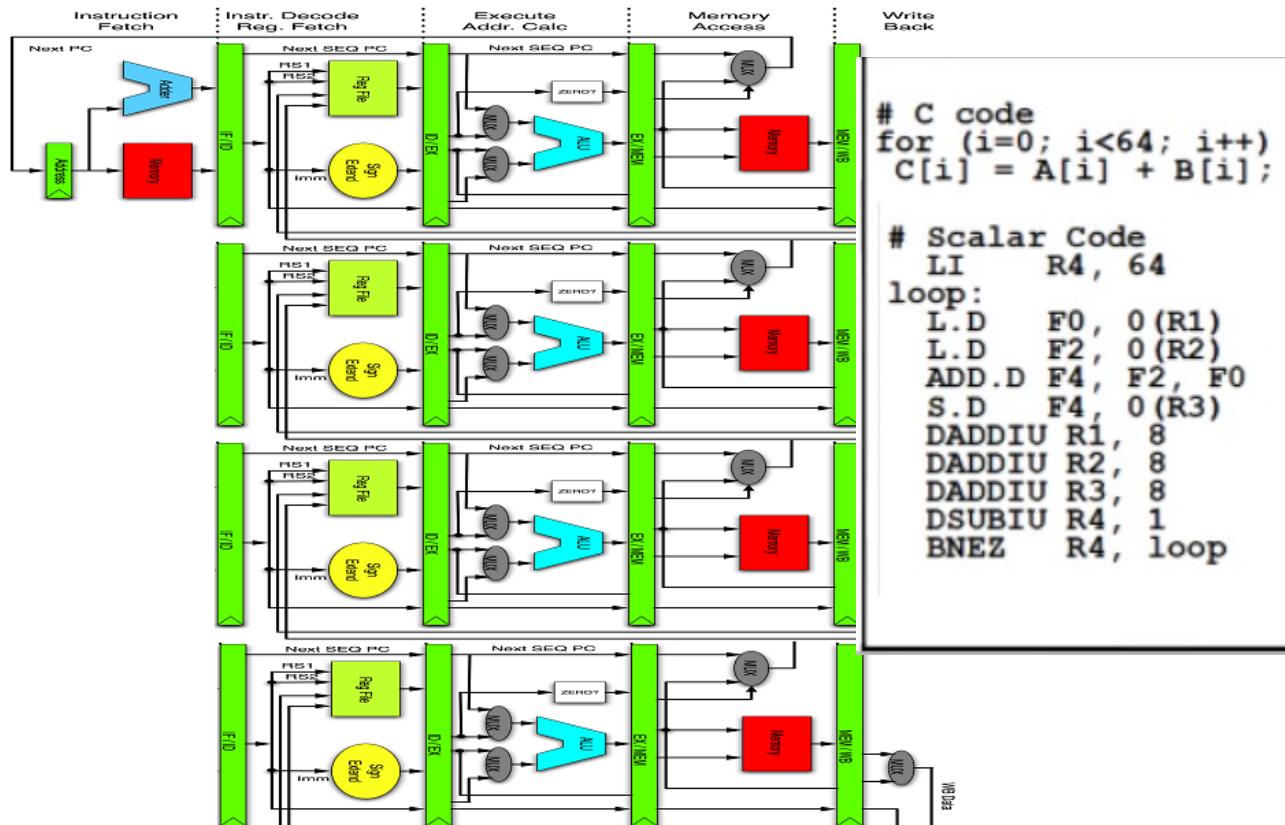
- What is a reduction? A prefix sum? Why are they hard to parallelize and what basic techniques can be used to parallelize them?
- Define flow dependence, output dependence, and anti-dependence: give an example of each. Why/how do compilers use them to detect loop-independent vs loop-carried dependences?
- What is the difference between a thread-block and a warp?
- How/Why must programmers copy data back and forth to a GPU?
- What is “shared memory” in CUDA? Describe a setting in which it might be useful.
- CUDA kernels have implicit barrier synchronization. Why is `__syncthreads()` necessary in light of this fact?
- How might one implement locks on a GPU?
- What ordering guarantees does a GPU provide across different hardware threads’ access to a single memory location? To two disjoint locations?
- When is it safe for one GPU thread to wait (e.g. by spinning) for another?

Review: what is a vector processor?

```
# C code
for (i=0; i<64; i++)
    C[i] = A[i] + B[i];

# Scalar Code
LI    R4, 64
loop:
    L.D   F0, 0(R1)
    L.D   F2, 0(R2)
    ADD.D F4, F2, F0
    S.D   F4, 0(R3)
    DADDIU R1, 8
    DADDIU R2, 8
    DADDIU R3, 8
    DSUBIU R4, 1
    BNEZ  R4, loop
```

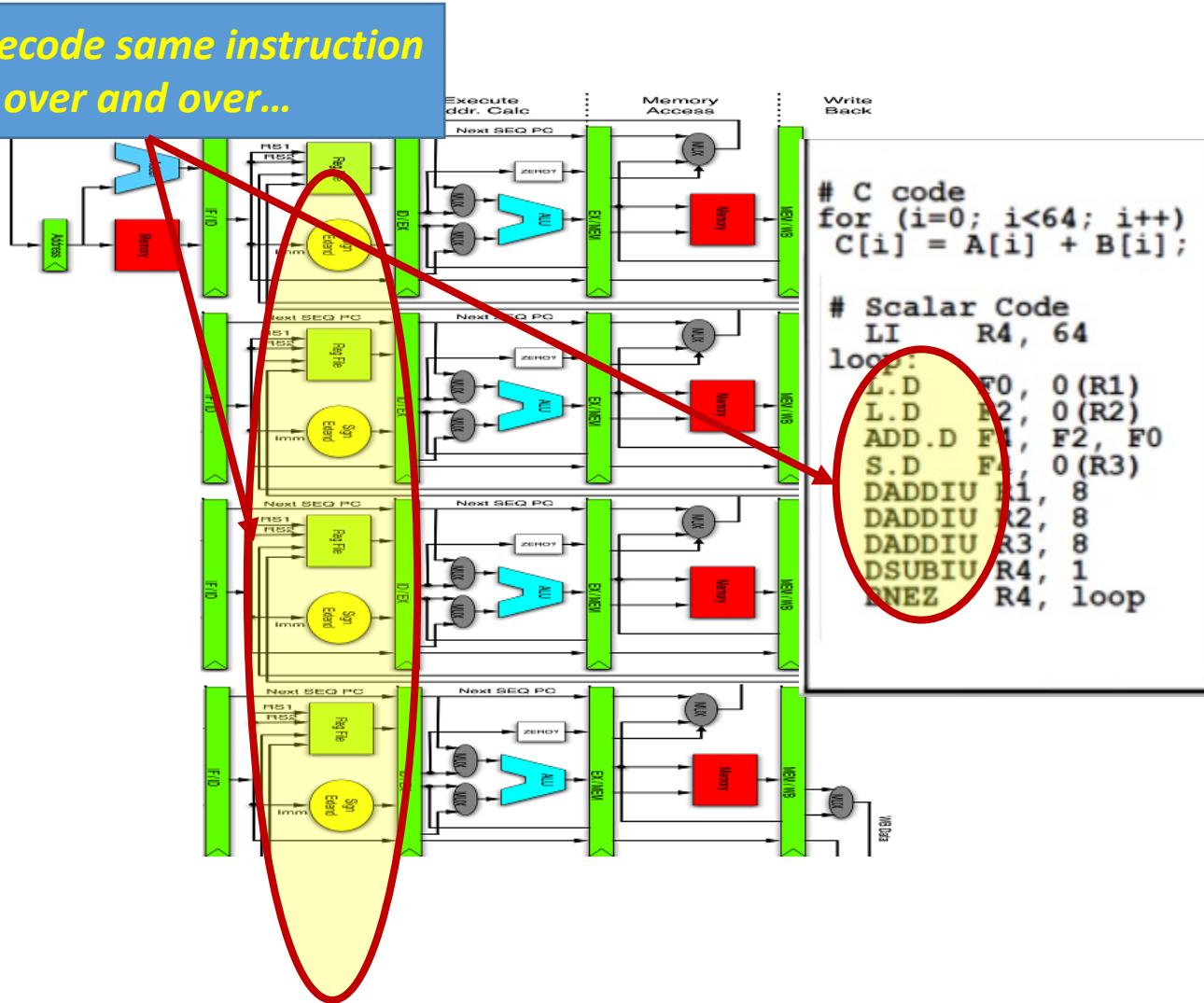
Review: what is a vector processor?



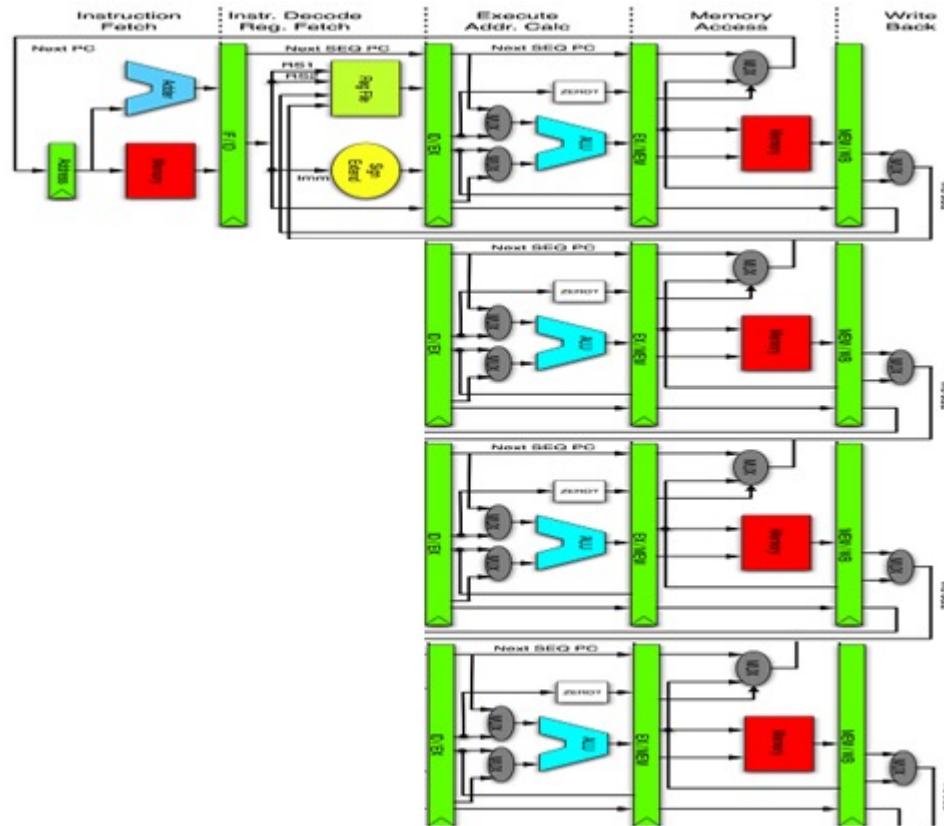
```
# C code  
for (i=0; i<64; i++)  
    C[i] = A[i] + B[i];  
  
# Scalar Code  
LI    R4, 64  
loop:  
    L.D   F0, 0(R1)  
    L.D   F2, 0(R2)  
    ADD.D F4, F2, F0  
    S.D   F4, 0(R3)  
    DADDIU R1, 8  
    DADDIU R2, 8  
    DADDIU R3, 8  
    DSUBIU R4, 1  
    BNEZ  R4, loop
```

Review: what is a vector processor?

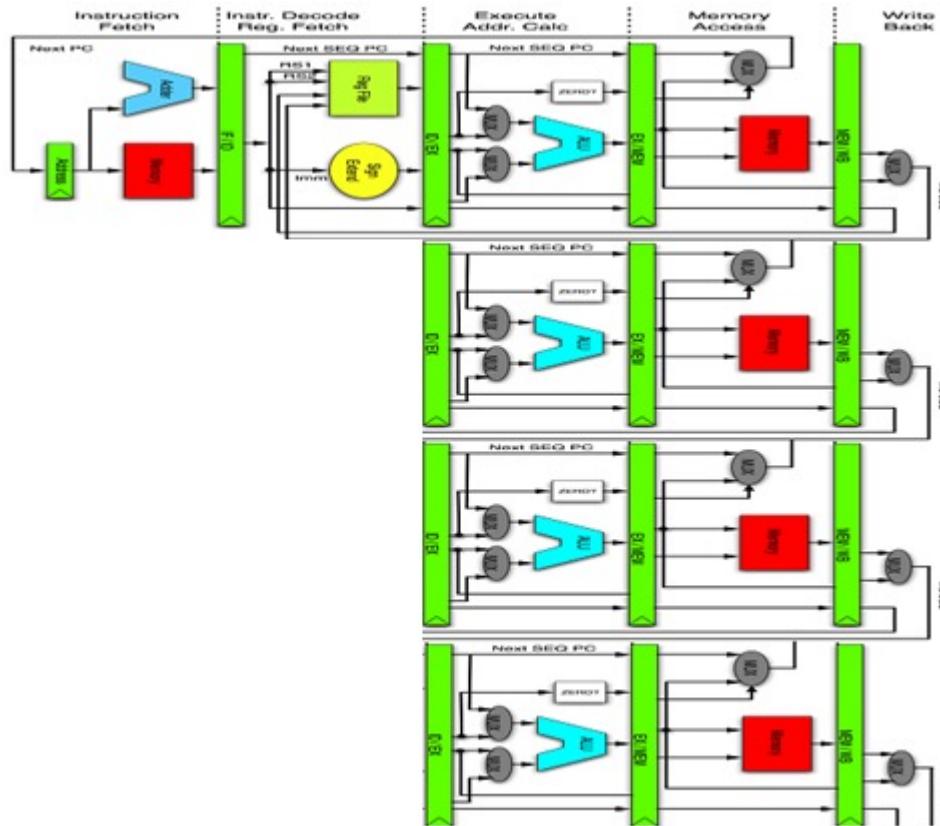
Dont decode same instruction over and over...



Review: what is a vector processor?



Review: what is a vector processor?

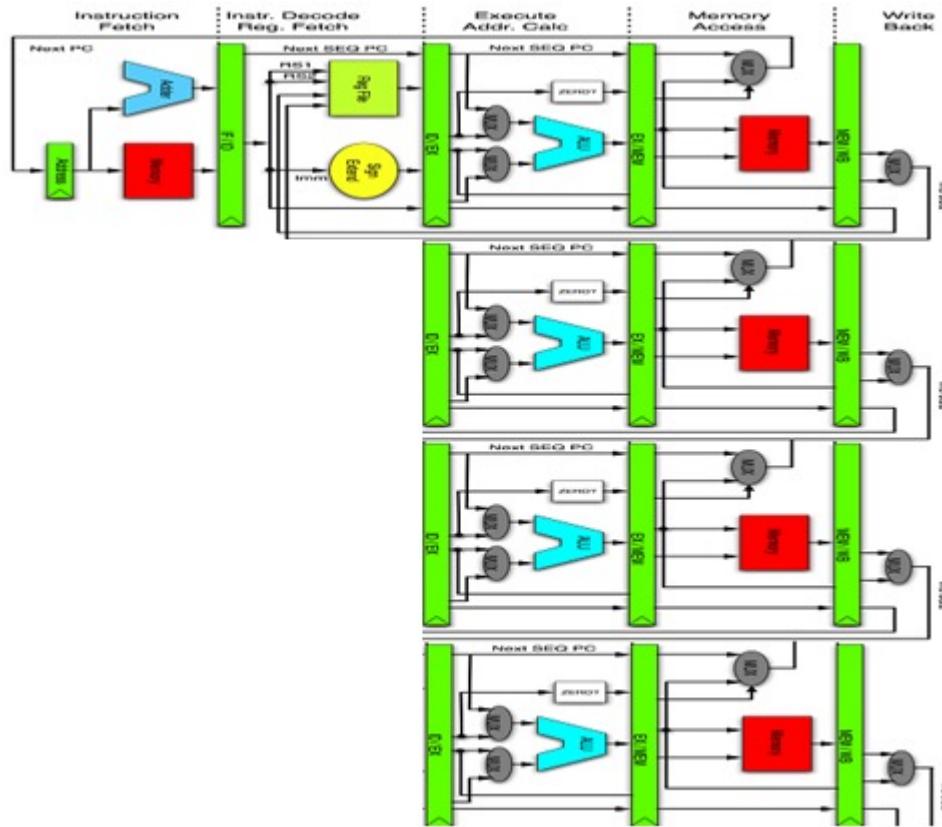


```
# C code  
for (i=0; i<64; i++)  
C[i] = A[i] + B[i];
```

```
# Scalar Code  
LI R4, 64  
loop:  
L.D F0, 0(R1)  
L.D F2, 0(R2)  
ADD.D F4, F2, F0  
S.D F4, 0(R3)  
DADDIU R1, 8  
DADDIU R2, 8  
DADDIU R3, 8  
DSUBIU R4, 1  
BNEZ R4, loop
```

```
# Vector Code  
LI VLR, 64  
LV V1, R1  
LV V2, R2  
ADDV.D V3, V1, V2  
SV V3, R3
```

Review: what is a vector processor?

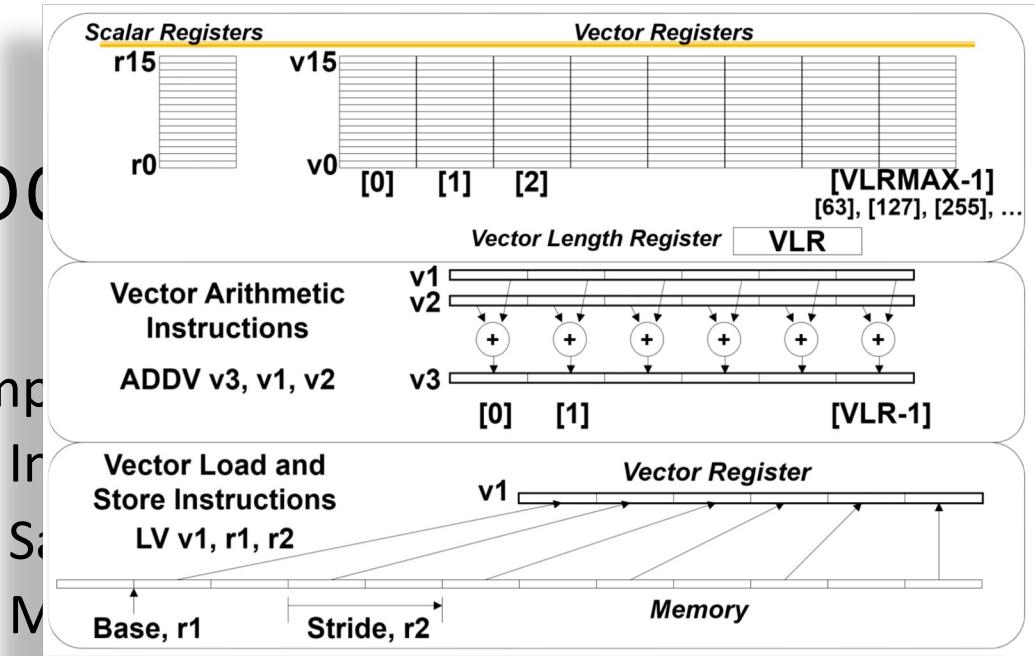
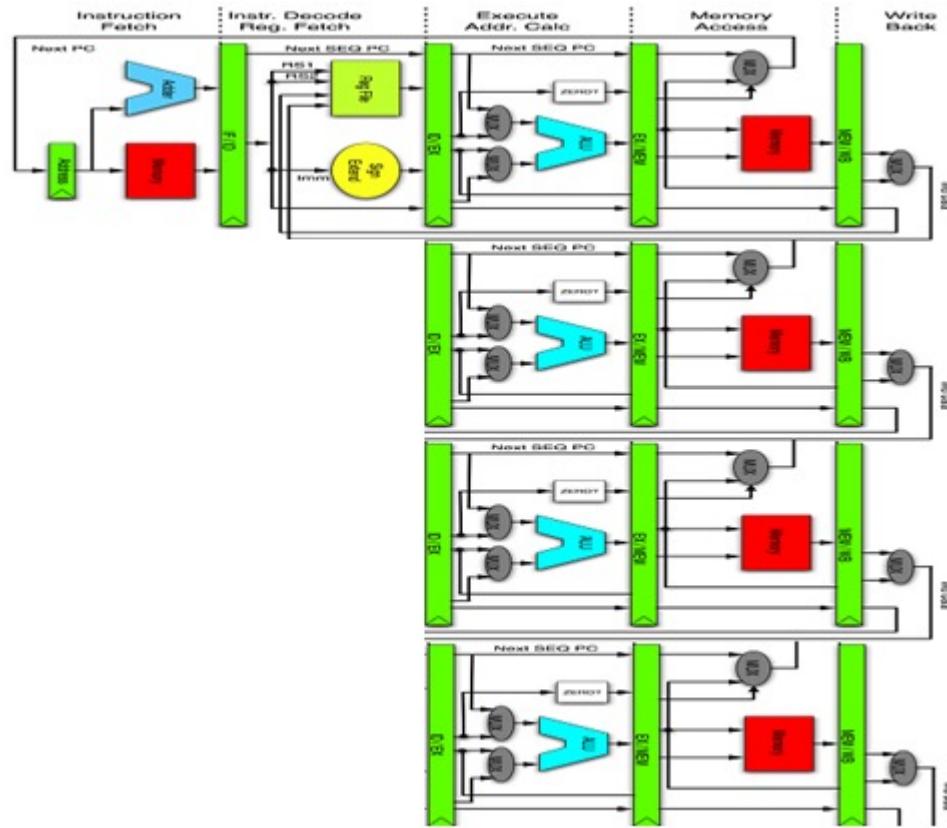


Implementation:

- Instruction fetch control logic shared
- Same instruction stream executed on
- Multiple pipelines
- Multiple different operands in parallel

# C code for (i=0; i<64; i++) C[i] = A[i] + B[i];	# Scalar Code LI R4, 64 loop: L.D F0, 0(R1) L.D F2, 0(R2) ADD.D F4, F2, F0 S.D F4, 0(R3) DADDIU R1, 8 DADDIU R2, 8 DADDIU R3, 8 DSUBIU R4, 1 BNEZ R4, loop	# Vector Code LI VLR, 64 LV V1, R1 LV V2, R2 ADDV.D V3, V1, V2 SV V3, R3
---	---	---

Review: what is a vector processor?



Implementation

- Instruction parallelism
- Scalar parallelism
- Memory parallelism
- Multiple different operands in parallel

```
# C code
for (i=0; i<64; i++)
    C[i] = A[i] + B[i];
```

```
# Scalar Code
LI R4, 64
loop:
    L.D F0, 0(R1)
    L.D F2, 0(R2)
    ADD.D F4, F2, F0
    S.D F4, 0(R3)
    DADDIU R1, 8
    DADDIU R2, 8
    DADDIU R3, 8
    DSUBIU R4, 1
    BNEZ R4, loop
```

```
# Vector Code
LI VLR, 64
LV V1, R1
LV V2, R2
ADDV.D V3, V1, V2
SV V3, R3
```

Review: Hardware multi-threading

Review: Hardware multi-threading

- Address memory bottleneck

Review: Hardware multi-threading

- Address memory bottleneck
- Share exec unit across
 - Instruction streams
 - Switch on stalls

Review: Hardware multi-threading

- Address memory bottleneck
- Share exec unit across
 - Instruction streams
 - Switch on stalls

ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0) MM0	ST(1) MM1	MOVAVEX RAX	RDH WE R8	RDH WE R12	CR0	CR4
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2) MM2	ST(3) MM3	MOVBX3 RDH	RDH WE R9	RDH WE R13	CR1	CR5
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4) MM4	ST(5) MM5	MOVCX3 RCX	RDH WE R10	RDH WE R14	CR2	CR6
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6) MM6	ST(7) MM7	MOVDX3 RDH	RDH WE R11	RDH WE R15	CR3	CR7
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	ST(8) MM8	ST(9) MM9	MOVBP3 RDH	RDH WE R12	RDH WE R16	CR4	CR8
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	CW	FP_IP	FP_DP	FP_P	ES	CS	MSW
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	SW	SW	SW	SW	DR5	DR5	CR9
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	TW	TP	TP	TP	DR6	DR6	CR10
ZMM16	YMM16	XMM16	ZMM17	YMM17	XMM17	FP_DS	FP DS	FP DS	FP DS	DR7	DR7	CR11
ZMM18	YMM18	XMM18	ZMM19	YMM19	XMM19	FF_OPC	FF_OPC	FF_OPC	FF_OPC	DR8	DR8	CR12
ZMM20	YMM20	XMM20	ZMM21	YMM21	XMM21	TR	TR	TR	TR	DR9	DR9	CR13
ZMM22	YMM22	XMM22	ZMM23	YMM23	XMM23	CDTR	CDTR	CDTR	CDTR	DR10	DR10	CR14
ZMM24	YMM24	XMM24	ZMM25	YMM25	XMM25	DR0	DR0	DR0	DR0	DR11	DR11	CR15
ZMM26	YMM26	XMM26	ZMM27	YMM27	XMM27	DR1	DR1	DR1	DR1	DR12	DR12	MXCSR
ZMM28	YMM28	XMM28	ZMM29	YMM29	XMM29	DR2	DR2	DR2	DR2	DR13	DR13	
ZMM30	YMM30	XMM30	ZMM31	YMM31	XMM31	DR3	DR3	DR3	DR3	DR14	DR14	
ZMM32	YMM32	XMM32	ZMM33	YMM33	XMM33	DR4	DR4	DR4	DR4	DR15	DR15	
ZMM34	YMM34	XMM34	ZMM35	YMM35	XMM35	DR5	DR5	DR5	DR5	DR16	DR16	



Review: Hardware multi-threading

- Address memory bottleneck
- Share exec unit across
 - Instruction streams
 - Switch on stalls

ZMM00	YMM00 XMM0	ZMM01	YMM01 XMM01	ST(0) MM00	ST(1) MM01	AHXEX RAX	-Inv no. 88	-Inv no. 82	C0	C4
ZMM02	YMM02 XMM02	ZMM03	YMM03 XMM03	ST(2) MM02	ST(3) MM03	AHXEX RBX	-Inv no. 89	-Inv no. 83	C1	C5
ZMM04	YMM04 XMM04	ZMM05	YMM05 XMM05	ST(4) MM04	ST(5) MM05	AHXEX RCX	-Inv no. 10	-Inv no. 14	C2	C6
ZMM06	YMM06 XMM06	ZMM07	YMM07 XMM07	ST(6) MM06	ST(7) MM07	AHXEX RDX	-Inv no. 11	-Inv no. 15	C3	C7
ZMM08	YMM08 XMM08	ZMM09	YMM09 XMM09	W	W	W	W	W	W	W
ZMM10	YMM10 XMM10	ZMM11	YMM11 XMM11	CW	FP_IP [FP_DP]P_C5	W	CS [S] GS	W	W	MSW CR9
ZMM12	YMM12 XMM12	ZMM13	YMM13 XMM13	SW	W	W	W	W	W	CR10
ZMM14	YMM14 XMM14	ZMM15	YMM15 XMM15	TW	W	W	W	W	W	CR11
ZMM16	YMM16 XMM16	ZMM17	YMM17 XMM17	FP DS	W	W	W	W	W	CR12
ZMM18	YMM18 XMM18	ZMM19	YMM19 XMM19	W	W	W	W	W	W	CR13
ZMM20	YMM20 XMM20	ZMM21	YMM21 XMM21	FF_OPC FP_DP FP_IP	W	W	W	W	W	CR14
ZMM22	YMM22 XMM22	ZMM23	YMM23 XMM23	W	W	W	W	W	W	CR15 MXCSR
ZMM24	YMM24 XMM24	ZMM25	YMM25 XMM25	W	W	W	W	W	W	CR16
ZMM26	YMM26 XMM26	ZMM27	YMM27 XMM27	W	W	W	W	W	W	CR17
ZMM28	YMM28 XMM28	ZMM29	YMM29 XMM29	W	W	W	W	W	W	CR18
ZMM30	YMM30 XMM30	ZMM31	YMM31 XMM31	W	W	W	W	W	W	CR19
ZMM32	YMM32 XMM32	ZMM33	YMM33 XMM33	W	W	W	W	W	W	CR20
ZMM34	YMM34 XMM34	ZMM35	YMM35 XMM35	W	W	W	W	W	W	CR21
ZMM36	YMM36 XMM36	ZMM37	YMM37 XMM37	W	W	W	W	W	W	CR22
ZMM38	YMM38 XMM38	ZMM39	YMM39 XMM39	W	W	W	W	W	W	CR23
ZMM40	YMM40 XMM40	ZMM41	YMM41 XMM41	W	W	W	W	W	W	CR24
ZMM42	YMM42 XMM42	ZMM43	YMM43 XMM43	W	W	W	W	W	W	CR25
ZMM44	YMM44 XMM44	ZMM45	YMM45 XMM45	W	W	W	W	W	W	CR26
ZMM46	YMM46 XMM46	ZMM47	YMM47 XMM47	W	W	W	W	W	W	CR27
ZMM48	YMM48 XMM48	ZMM49	YMM49 XMM49	W	W	W	W	W	W	CR28
ZMM50	YMM50 XMM50	ZMM51	YMM51 XMM51	W	W	W	W	W	W	CR29
ZMM52	YMM52 XMM52	ZMM53	YMM53 XMM53	W	W	W	W	W	W	CR30
ZMM54	YMM54 XMM54	ZMM55	YMM55 XMM55	W	W	W	W	W	W	CR31
ZMM56	YMM56 XMM56	ZMM57	YMM57 XMM57	W	W	W	W	W	W	CR32
ZMM58	YMM58 XMM58	ZMM59	YMM59 XMM59	W	W	W	W	W	W	CR33
ZMM60	YMM60 XMM60	ZMM61	YMM61 XMM61	W	W	W	W	W	W	CR34
ZMM62	YMM62 XMM62	ZMM63	YMM63 XMM63	W	W	W	W	W	W	CR35
ZMM64	YMM64 XMM64	ZMM65	YMM65 XMM65	W	W	W	W	W	W	CR36
ZMM66	YMM66 XMM66	ZMM67	YMM67 XMM67	W	W	W	W	W	W	CR37
ZMM68	YMM68 XMM68	ZMM69	YMM69 XMM69	W	W	W	W	W	W	CR38
ZMM70	YMM70 XMM70	ZMM71	YMM71 XMM71	W	W	W	W	W	W	CR39
ZMM72	YMM72 XMM72	ZMM73	YMM73 XMM73	W	W	W	W	W	W	CR40
ZMM74	YMM74 XMM74	ZMM75	YMM75 XMM75	W	W	W	W	W	W	CR41
ZMM76	YMM76 XMM76	ZMM77	YMM77 XMM77	W	W	W	W	W	W	CR42
ZMM78	YMM78 XMM78	ZMM79	YMM79 XMM79	W	W	W	W	W	W	CR43
ZMM80	YMM80 XMM80	ZMM81	YMM81 XMM81	W	W	W	W	W	W	CR44
ZMM82	YMM82 XMM82	ZMM83	YMM83 XMM83	W	W	W	W	W	W	CR45
ZMM84	YMM84 XMM84	ZMM85	YMM85 XMM85	W	W	W	W	W	W	CR46
ZMM86	YMM86 XMM86	ZMM87	YMM87 XMM87	W	W	W	W	W	W	CR47
ZMM88	YMM88 XMM88	ZMM89	YMM89 XMM89	W	W	W	W	W	W	CR48
ZMM90	YMM90 XMM90	ZMM91	YMM91 XMM91	W	W	W	W	W	W	CR49
ZMM92	YMM92 XMM92	ZMM93	YMM93 XMM93	W	W	W	W	W	W	CR50
ZMM94	YMM94 XMM94	ZMM95	YMM95 XMM95	W	W	W	W	W	W	CR51
ZMM96	YMM96 XMM96	ZMM97	YMM97 XMM97	W	W	W	W	W	W	CR52
ZMM98	YMM98 XMM98	ZMM99	YMM99 XMM99	W	W	W	W	W	W	CR53
ZMM100	YMM100 XMM100	ZMM101	YMM101 XMM101	W	W	W	W	W	W	CR54
ZMM102	YMM102 XMM102	ZMM103	YMM103 XMM103	W	W	W	W	W	W	CR55
ZMM104	YMM104 XMM104	ZMM105	YMM105 XMM105	W	W	W	W	W	W	CR56
ZMM106	YMM106 XMM106	ZMM107	YMM107 XMM107	W	W	W	W	W	W	CR57
ZMM108	YMM108 XMM108	ZMM109	YMM109 XMM109	W	W	W	W	W	W	CR58
ZMM110	YMM110 XMM110	ZMM111	YMM111 XMM111	W	W	W	W	W	W	CR59
ZMM112	YMM112 XMM112	ZMM113	YMM113 XMM113	W	W	W	W	W	W	CR60
ZMM114	YMM114 XMM114	ZMM115	YMM115 XMM115	W	W	W	W	W	W	CR61
ZMM116	YMM116 XMM116	ZMM117	YMM117 XMM117	W	W	W	W	W	W	CR62
ZMM118	YMM118 XMM118	ZMM119	YMM119 XMM119	W	W	W	W	W	W	CR63
ZMM120	YMM120 XMM120	ZMM121	YMM121 XMM121	W	W	W	W	W	W	CR64
ZMM122	YMM122 XMM122	ZMM123	YMM123 XMM123	W	W	W	W	W	W	CR65
ZMM124	YMM124 XMM124	ZMM125	YMM125 XMM125	W	W	W	W	W	W	CR66
ZMM126	YMM126 XMM126	ZMM127	YMM127 XMM127	W	W	W	W	W	W	CR67
ZMM128	YMM128 XMM128	ZMM129	YMM129 XMM129	W	W	W	W	W	W	CR68
ZMM130	YMM130 XMM130	ZMM131	YMM131 XMM131	W	W	W	W	W	W	CR69
ZMM132	YMM132 XMM132	ZMM133	YMM133 XMM133	W	W	W	W	W	W	CR70
ZMM134	YMM134 XMM134	ZMM135	YMM135 XMM135	W	W	W	W	W	W	CR71
ZMM136	YMM136 XMM136	ZMM137	YMM137 XMM137	W	W	W	W	W	W	CR72
ZMM138	YMM138 XMM138	ZMM139	YMM139 XMM139	W	W	W	W	W	W	CR73
ZMM140	YMM140 XMM140	ZMM141	YMM141 XMM141	W	W	W	W	W	W	CR74
ZMM142	YMM142 XMM142	ZMM143	YMM143 XMM143	W	W	W	W	W	W	CR75
ZMM144	YMM144 XMM144	ZMM145	YMM145 XMM145	W	W	W	W	W	W	CR76
ZMM146	YMM146 XMM146	ZMM147	YMM147 XMM147	W	W	W	W	W	W	CR77
ZMM148	YMM148 XMM148	ZMM149	YMM149 XMM149	W	W	W	W	W	W	CR78
ZMM150	YMM150 XMM150	ZMM151	YMM151 XMM151	W	W	W	W	W	W	CR79
ZMM152	YMM152 XMM152	ZMM153	YMM153 XMM153	W	W	W	W	W	W	CR80
ZMM154	YMM154 XMM154	ZMM155	YMM155 XMM155	W	W	W	W	W	W	CR81
ZMM156	YMM156 XMM156	ZMM157	YMM157 XMM157	W	W	W	W	W	W	CR82
ZMM158	YMM158 XMM158	ZMM159	YMM159 XMM159	W	W	W	W	W	W	CR83
ZMM160	YMM160 XMM160	ZMM161	YMM161 XMM161	W	W	W	W	W	W	CR84
ZMM162	YMM162 XMM162	ZMM163	YMM163 XMM163	W	W	W	W	W	W	CR85
ZMM164	YMM164 XMM164	ZMM165	YMM165 XMM165	W	W	W	W	W	W	CR86
ZMM166	YMM166 XMM166	ZMM167	YMM167 XMM167	W	W	W	W	W	W	CR87
ZMM168	YMM168 XMM168	ZMM169	YMM169 XMM169	W	W	W	W	W	W	CR88
ZMM170	YMM170 XMM170	ZMM171	YMM171 XMM171	W	W	W	W	W	W	CR89
ZMM172	YMM172 XMM172	ZMM173	YMM173 XMM173	W	W	W	W	W	W	CR90
ZMM174	YMM174 XMM174	ZMM175	YMM175 XMM175	W	W	W	W	W	W	CR91
ZMM176	YMM176 XMM176	ZMM177	YMM177 XMM177	W	W	W	W	W	W	CR92
ZMM178	YMM178 XMM178	ZMM179	YMM179 XMM179	W	W	W	W	W	W	CR93
ZMM180	YMM180 XMM180	ZMM181	YMM181 XMM181	W	W	W	W	W	W	CR94
ZMM182	YMM182 XMM182	ZMM183	YMM183 XMM183	W	W	W	W	W	W	CR95
ZMM184	YMM184 XMM184	ZMM185	YMM185 XMM185	W	W	W	W	W	W	CR96
ZMM186	YMM186 XMM186	ZMM187	YMM187 XMM187	W	W	W	W	W	W	CR97
ZMM188	YMM188 XMM188	ZMM189	YMM189 XMM189	W	W	W	W	W	W	CR98
ZMM190	YMM190 XMM190	ZMM191	YMM191 XMM191	W	W	W	W	W	W	CR99
ZMM192	YMM192 XMM192	ZMM193	YMM193 XMM193	W	W	W	W	W	W	CR100
ZMM194	YMM194 XMM194	ZMM195	YMM195 XMM195	W	W	W	W	W	W	CR101
ZMM196	YMM196 XMM196	ZMM197	YMM197 XMM197	W	W	W	W	W	W	CR102
ZMM198	YMM198 XMM198	ZMM199	YMM199 XMM199	W	W	W	W	W	W	CR103
ZMM200	YMM200 XMM200	ZMM201	YMM201 XMM201	W	W	W	W	W	W	CR104
ZMM202	YMM202 XMM202	ZMM203	YMM203 XMM203	W	W	W	W	W	W	CR105
ZMM204	YMM204 XMM204	ZMM205	YMM205 XMM205	W	W	W	W	W	W	CR106
ZMM206	YMM206 XMM206	ZMM207	YMM207 XMM207	W	W	W	W	W	W	CR107
ZMM208	YMM208 XMM208	ZMM209	YMM209 XMM209	W	W	W	W	W	W	CR108
ZMM210	YMM210 XMM210	ZMM211	YMM211 XMM211	W	W	W	W	W	W	CR109
ZMM212	YMM212 XMM212	ZMM213	YMM213 XMM213	W	W	W	W	W	W	CR110
ZMM214	YMM214 XMM214	ZMM215	YMM215 XMM215	W	W	W	W	W	W	CR111
ZMM216	YMM216 XMM216	ZMM217	YMM217 XMM217	W	W	W	W	W	W	CR112
ZMM218	YMM218 XMM218	ZMM219	YMM219 XMM219	W	W	W	W	W	W	CR113
ZMM220	YMM220 XMM220	ZMM221	YMM221 XMM221	W	W	W	W	W	W	CR114
ZMM222	YMM222 XMM222	ZMM223	YMM223 XMM223	W	W	W	W	W	W	CR115
ZMM224	YMM224 XMM224	ZMM225	YMM225 XMM225	W						

Review: Hardware multi-threading

- Address memory bottleneck
- Share exec unit across
 - Instruction streams
 - Switch on stalls
- Looks like multiple cores to the OS

ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0) MM0	ST(1) MM1	AHIANBX RAX	-Inv me. 88	-Inv me. R12	CR0	CR4	
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2) MM2	ST(3) MM3	AHIBEX RBX	-Inv me. 89	-Inv me. R13	CR1	CR5	
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4) MM4	ST(5) MM5	AHICEX RCX	-Inv me. R10	-Inv me. R14	CR2	CR6	
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6) MM6	ST(7) MM7	AHIDEX RDX	-Inv me. R11	-Inv me. R15	CR3	CR7	
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	ZW	CW	AHDPEPFBP	-ID ED(RD)	-IP DP RIP	CR4	CR8	
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	SW	FP DS	IP IP	FP DP	P CS	MSW	CR9	
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	TW	FP DS	IP IP	FP DP	P CS	CR10		
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	FP DS	FF OPC	FP DP	FP IP	CS SS DS	GDTR	IDTR	DRO DR6 CR13
ZMM16	YMM16	XMM16	ZMM17	YMM17	XMM17	FP DS	ES FS GS	TR	LDTR	DR2 DR7 CR14	DR2 DR8 CR15 MXCSR		
ZMM18	YMM18	XMM18	ZMM19	YMM19	XMM19	FP DS	FP DS	IP IP	FP DP	P CS	DR4 DR10 DR12 DR14		
ZMM20	YMM20	XMM20	ZMM21	YMM21	XMM21	FP DS	FP DS	IP IP	FP DP	P CS	DR5 DR11 DR13 DR15		



ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0) MM0	ST(1) MM1	AHIANBX RAX	-Inv me. 88	-Inv me. R12	CR0	CR4	
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2) MM2	ST(3) MM3	AHIBEX RBX	-Inv me. 89	-Inv me. R13	CR1	CR5	
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4) MM4	ST(5) MM5	AHICEX RCX	-Inv me. R10	-Inv me. R14	CR2	CR6	
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6) MM6	ST(7) MM7	AHIDEX RDX	-Inv me. R11	-Inv me. R15	CR3	CR7	
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	ZW	CW	AHDPEPFBP	-ID ED(RD)	-IP DP RIP	CR4	CR8	
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	SW	FP DS	IP IP	FP DP	P CS	MSW	CR9	
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	TW	FP DS	IP IP	FP DP	P CS	CR10		
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	FP DS	FF OPC	FP DP	FP IP	CS SS DS	GDTR	IDTR	DRO DR6 CR13
ZMM16	YMM16	XMM16	ZMM17	YMM17	XMM17	FP DS	ES FS GS	TR	LDTR	DR2 DR7 CR14	DR2 DR8 CR15 MXCSR		
ZMM18	YMM18	XMM18	ZMM19	YMM19	XMM19	FP DS	FP DS	IP IP	FP DP	P CS	DR4 DR10 DR12 DR14		
ZMM20	YMM20	XMM20	ZMM21	YMM21	XMM21	FP DS	FP DS	IP IP	FP DP	P CS	DR5 DR11 DR13 DR15		



ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0) MM0	ST(1) MM1	AHIANBX RAX	-Inv me. 88	-Inv me. R12	CR0	CR4	
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2) MM2	ST(3) MM3	AHIBEX RBX	-Inv me. 89	-Inv me. R13	CR1	CR5	
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4) MM4	ST(5) MM5	AHICEX RCX	-Inv me. R10	-Inv me. R14	CR2	CR6	
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6) MM6	ST(7) MM7	AHIDEX RDX	-Inv me. R11	-Inv me. R15	CR3	CR7	
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	ZW	CW	AHDPEPFBP	-ID ED(RD)	-IP DP RIP	CR4	CR8	
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	SW	FP DS	IP IP	FP DP	P CS	MSW	CR9	
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	TW	FP DS	IP IP	FP DP	P CS	CR10		
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	FP DS	FF OPC	FP DP	FP IP	CS SS DS	GDTR	IDTR	DRO DR6 CR13
ZMM16	YMM16	XMM16	ZMM17	YMM17	XMM17	FP DS	ES FS GS	TR	LDTR	DR2 DR7 CR14	DR2 DR8 CR15 MXCSR		
ZMM18	YMM18	XMM18	ZMM19	YMM19	XMM19	FP DS	FP DS	IP IP	FP DP	P CS	DR4 DR10 DR12 DR14		
ZMM20	YMM20	XMM20	ZMM21	YMM21	XMM21	FP DS	FP DS	IP IP	FP DP	P CS	DR5 DR11 DR13 DR15		

Review: Hardware multi-threading

- Address memory bottleneck
- Share exec unit across
 - Instruction streams
 - Switch on stalls
- Looks like multiple cores to the OS
- Three variants:
 - Coarse
 - Fine-grain
 - Simultaneous

ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0) MM0	ST(1) MM1	AHIANBX RAX	-Inv me. 88	Inv me. R12	CR0	CR4
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2) MM2	ST(3) MM3	AHIANBX RBX	-Inv me. 89	Inv me. R13	CR1	CR5
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4) MM4	ST(5) MM5	AHIANBX RCX	-Inv me. R10	Inv me. R14	CR2	CR6
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6) MM6	ST(7) MM7	AHIANBX RDX	-Inv me. R11	Inv me. R15	CR3	CR7
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR3 CR6		
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	CW	FP IP	FP DP	P CS	CS SS DS	GDT IDTR	DR0 DR6 CR13
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	SW	TP	TP DP	P GS	TR FS GS	LDTR	DR2 DR7 CR14
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	TW						DR3 DR9 CR15 MXCSR
ZMM16	YMM16	XMM16	ZMM17	YMM17	XMM17	FP DS	FP OPC	FP DP	P IP	CS SS DS	CS SS DS	DR4 DR10 DR12 DR14
ZMM18	YMM18	XMM18	ZMM19	YMM19	XMM19	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	MSW CR9		
ZMM20	YMM20	XMM20	ZMM21	YMM21	XMM21	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR10		
ZMM22	YMM22	XMM22	ZMM23	YMM23	XMM23	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR11		
ZMM24	YMM24	XMM24	ZMM25	YMM25	XMM25	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR12		
ZMM26	YMM26	XMM26	ZMM27	YMM27	XMM27	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR13		
ZMM28	YMM28	XMM28	ZMM29	YMM29	XMM29	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR14		
ZMM30	YMM30	XMM30	ZMM31	YMM31	XMM31	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR15		
ZMM32	YMM32	XMM32	ZMM33	YMM33	XMM33	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR16		
ZMM34	YMM34	XMM34	ZMM35	YMM35	XMM35	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR17		
ZMM36	YMM36	XMM36	ZMM37	YMM37	XMM37	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR18		
ZMM38	YMM38	XMM38	ZMM39	YMM39	XMM39	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR19		
ZMM40	YMM40	XMM40	ZMM41	YMM41	XMM41	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR20		
ZMM42	YMM42	XMM42	ZMM43	YMM43	XMM43	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR21		
ZMM44	YMM44	XMM44	ZMM45	YMM45	XMM45	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR22		
ZMM46	YMM46	XMM46	ZMM47	YMM47	XMM47	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR23		
ZMM48	YMM48	XMM48	ZMM49	YMM49	XMM49	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR24		
ZMM50	YMM50	XMM50	ZMM51	YMM51	XMM51	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR25		
ZMM52	YMM52	XMM52	ZMM53	YMM53	XMM53	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR26		
ZMM54	YMM54	XMM54	ZMM55	YMM55	XMM55	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR27		
ZMM56	YMM56	XMM56	ZMM57	YMM57	XMM57	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR28		
ZMM58	YMM58	XMM58	ZMM59	YMM59	XMM59	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR29		
ZMM60	YMM60	XMM60	ZMM61	YMM61	XMM61	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR30		
ZMM62	YMM62	XMM62	ZMM63	YMM63	XMM63	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR31		
ZMM64	YMM64	XMM64	ZMM65	YMM65	XMM65	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR32		
ZMM66	YMM66	XMM66	ZMM67	YMM67	XMM67	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR33		
ZMM68	YMM68	XMM68	ZMM69	YMM69	XMM69	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR34		
ZMM70	YMM70	XMM70	ZMM71	YMM71	XMM71	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR35		
ZMM72	YMM72	XMM72	ZMM73	YMM73	XMM73	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR36		
ZMM74	YMM74	XMM74	ZMM75	YMM75	XMM75	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR37		
ZMM76	YMM76	XMM76	ZMM77	YMM77	XMM77	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR38		
ZMM78	YMM78	XMM78	ZMM79	YMM79	XMM79	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR39		
ZMM80	YMM80	XMM80	ZMM81	YMM81	XMM81	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR40		
ZMM82	YMM82	XMM82	ZMM83	YMM83	XMM83	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR41		
ZMM84	YMM84	XMM84	ZMM85	YMM85	XMM85	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR42		
ZMM86	YMM86	XMM86	ZMM87	YMM87	XMM87	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR43		
ZMM88	YMM88	XMM88	ZMM89	YMM89	XMM89	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR44		
ZMM90	YMM90	XMM90	ZMM91	YMM91	XMM91	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR45		
ZMM92	YMM92	XMM92	ZMM93	YMM93	XMM93	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR46		
ZMM94	YMM94	XMM94	ZMM95	YMM95	XMM95	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR47		
ZMM96	YMM96	XMM96	ZMM97	YMM97	XMM97	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR48		
ZMM98	YMM98	XMM98	ZMM99	YMM99	XMM99	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR49		
ZMM100	YMM100	XMM100	ZMM101	YMM101	XMM101	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR50		
ZMM102	YMM102	XMM102	ZMM103	YMM103	XMM103	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR51		
ZMM104	YMM104	XMM104	ZMM105	YMM105	XMM105	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR52		
ZMM106	YMM106	XMM106	ZMM107	YMM107	XMM107	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR53		
ZMM108	YMM108	XMM108	ZMM109	YMM109	XMM109	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR54		
ZMM110	YMM110	XMM110	ZMM111	YMM111	XMM111	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR55		
ZMM112	YMM112	XMM112	ZMM113	YMM113	XMM113	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR56		
ZMM114	YMM114	XMM114	ZMM115	YMM115	XMM115	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR57		
ZMM116	YMM116	XMM116	ZMM117	YMM117	XMM117	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR58		
ZMM118	YMM118	XMM118	ZMM119	YMM119	XMM119	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR59		
ZMM120	YMM120	XMM120	ZMM121	YMM121	XMM121	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR60		
ZMM122	YMM122	XMM122	ZMM123	YMM123	XMM123	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR61		
ZMM124	YMM124	XMM124	ZMM125	YMM125	XMM125	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR62		
ZMM126	YMM126	XMM126	ZMM127	YMM127	XMM127	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR63		
ZMM128	YMM128	XMM128	ZMM129	YMM129	XMM129	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR64		
ZMM130	YMM130	XMM130	ZMM131	YMM131	XMM131	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR65		
ZMM132	YMM132	XMM132	ZMM133	YMM133	XMM133	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR66		
ZMM134	YMM134	XMM134	ZMM135	YMM135	XMM135	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR67		
ZMM136	YMM136	XMM136	ZMM137	YMM137	XMM137	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR68		
ZMM138	YMM138	XMM138	ZMM139	YMM139	XMM139	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR69		
ZMM140	YMM140	XMM140	ZMM141	YMM141	XMM141	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR70		
ZMM142	YMM142	XMM142	ZMM143	YMM143	XMM143	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR71		
ZMM144	YMM144	XMM144	ZMM145	YMM145	XMM145	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR72		
ZMM146	YMM146	XMM146	ZMM147	YMM147	XMM147	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR73		
ZMM148	YMM148	XMM148	ZMM149	YMM149	XMM149	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR74		
ZMM150	YMM150	XMM150	ZMM151	YMM151	XMM151	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR75		
ZMM152	YMM152	XMM152	ZMM153	YMM153	XMM153	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR76		
ZMM154	YMM154	XMM154	ZMM155	YMM155	XMM155	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR77		
ZMM156	YMM156	XMM156	ZMM157	YMM157	XMM157	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR78		
ZMM158	YMM158	XMM158	ZMM159	YMM159	XMM159	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR79		
ZMM160	YMM160	XMM160	ZMM161	YMM161	XMM161	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR80		
ZMM162	YMM162	XMM162	ZMM163	YMM163	XMM163	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR81		
ZMM164	YMM164	XMM164	ZMM165	YMM165	XMM165	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR82		
ZMM166	YMM166	XMM166	ZMM167	YMM167	XMM167	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR83		
ZMM168	YMM168	XMM168	ZMM169	YMM169	XMM169	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR84		
ZMM170	YMM170	XMM170	ZMM171	YMM171	XMM171	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR85		
ZMM172	YMM172	XMM172	ZMM173	YMM173	XMM173	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR86		
ZMM174	YMM174	XMM174	ZMM175	YMM175	XMM175	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR87		
ZMM176	YMM176	XMM176	ZMM177	YMM177	XMM177	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR88		
ZMM178	YMM178	XMM178	ZMM179	YMM179	XMM179	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR89		
ZMM180	YMM180	XMM180	ZMM181	YMM181	XMM181	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR90		
ZMM182	YMM182	XMM182	ZMM183	YMM183	XMM183	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR91		
ZMM184	YMM184	XMM184	ZMM185	YMM185	XMM185	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR92		
ZMM186	YMM186	XMM186	ZMM187	YMM187	XMM187	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR93		
ZMM188	YMM188	XMM188	ZMM189	YMM189	XMM189	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR94		
ZMM190	YMM190	XMM190	ZMM191	YMM191	XMM191	ZWOPDSPRBP	ZWOPDSPRIP	WD EDI RDW	IP RIP	CR95		
ZMM192</td												

Programming Model

- *GPUs are I/O devices, managed by user-code*
- “kernels” == “shader programs”
- 1000s of HW-scheduled threads per kernel
- Threads grouped into independent blocks.
 - Threads in a block can synchronize (barrier)
 - This is the **only** synchronization
- “Grid” == “launch” == “invocation” of a kernel
 - a group of blocks (or warps)

Programming Model

- *GPUs are I/O devices, managed by user-code*
- “kernels” == “shader programs”
- 1000s of HW-scheduled threads per kernel
- Threads grouped into independent blocks.
 - Threads in a block can synchronize (barrier)
 - This is the **only** synchronization
- “Grid” == “launch” == “invocation” of a kernel
 - a group of blocks (or warps)

Need codes that are 1000s-X parallel....

Parallel Algorithms

- Sequential algorithms often do not permit easy parallelization
 - Does not mean there work has no parallelism
 - A different approach can yield parallelism
 - but often changes the algorithm
 - Parallelizing != just adding locks to a sequential algorithm
- Parallel Patterns
 - Map
 - Scatter, Gather
 - Reduction
 - Scan
 - Search, Sort

Parallel Algorithms

- Sequential algorithms often do not permit easy parallelization
 - Does not mean there work has no parallelism
 - A different approach can yield parallelism
 - but often changes the algorithm
 - Parallelizing != just adding locks to a sequential algorithm
- Parallel Patterns
 - Map
 - Scatter, Gather
 - Reduction
 - Scan
 - Search, Sort

If you can express your algorithm using these patterns, an apparently fundamentally sequential algorithm can be made parallel

Map

- Inputs
 - Array A
 - Function $f(x)$
- $\text{map}(A, f) \rightarrow$ apply $f(x)$ on all elements in A
- Parallelism trivially exposed
 - $f(x)$ can be applied in parallel to all elements, in principle

Map

- Inputs
 - Array A
 - Function $f(x)$
- $\text{map}(A, f) \rightarrow$ apply $f(x)$ on all elements in A
- Parallelism trivially exposed
 - $f(x)$ can be applied in parallel to all elements, in principle

```
for(i=0; i<numPoints; i++) {  
    labels[i] = findNearestCenter(points[i]);  
}
```



```
map(points, findNearestCenter)
```

Scatter and Gather

Scatter and Gather

- **Gather:**
 - Read multiple items to single /packed location

Scatter and Gather

- **Gather:**
 - Read multiple items to single /packed location
- **Scatter:**
 - Write single/packed data item to multiple locations

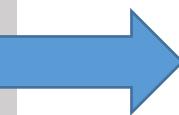
Scatter and Gather

- **Gather:**
 - Read multiple items to single /packed location
- **Scatter:**
 - Write single/packed data item to multiple locations
- Inputs: x , y , indeces, N

Scatter and Gather

- Gather:
 - Read multiple items to single /packed location
- Scatter:
 - Write single/packed data item to multiple locations
- Inputs: x, y, indeces, N

```
for (i=0; i<N; ++i)  
    x[i] = y[idx[i]];
```



```
gather(x, y, idx)
```

```
for (i=0; i<N; ++i)  
    y[idx[i]] = x[i];
```



```
scatter(x, y, idx)
```

Scatter and Gather

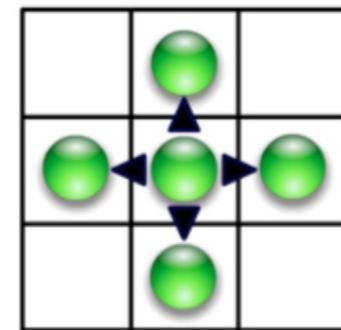
- Gather:
 - Read multiple items to single /packed location
- Scatter:
 - Write single/packed data item to multiple locations
- Inputs: x, y, indeces, N

```
for (i=0; i<N; ++i)  
    x[i] = y[idx[i]];
```

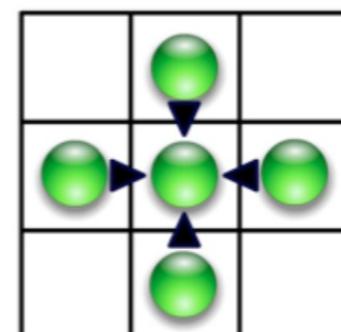
gather(x, y, idx)

```
for (i=0; i<N; ++i)  
    y[idx[i]] = x[i];
```

scatter(x, y, idx)



Scatter



Gather

Reduce

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
- $\text{Reduce}(\text{op}, s)$ returns a **op** a **op** b **op** $c \dots$ **op** z

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
- $\text{Reduce}(\text{op}, s)$ returns a **op** b **op** $c \dots \text{op} z$

```
for(i=0; i<N; ++i) {  
    accum += point[i]  
}
```

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
- $\text{Reduce}(\text{op}, s)$ returns a **op** b **op** $c \dots$ **op** z

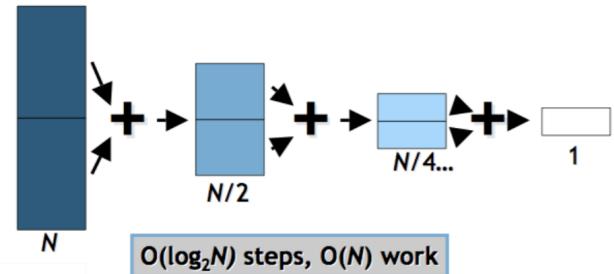
```
for(i=0; i<N; ++i) {  
    accum += point[i]  
}
```



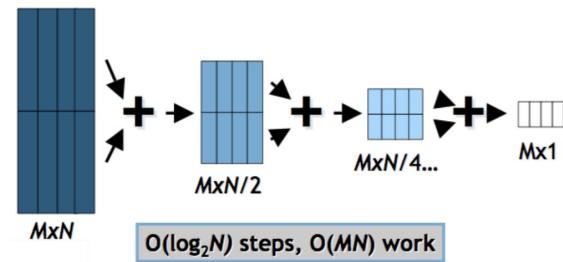
```
accum = reduce(+, point)
```

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
- $\text{Reduce}(\text{op}, s)$ returns a **op** b **op** $c \dots$ **op** z



$O(\log_2 N)$ steps, $O(N)$ work



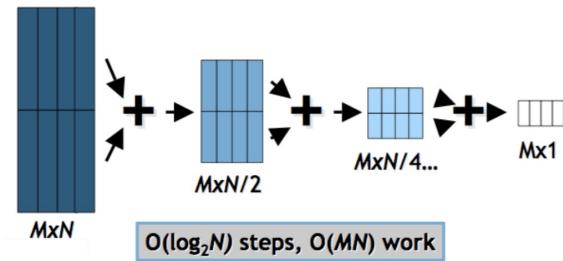
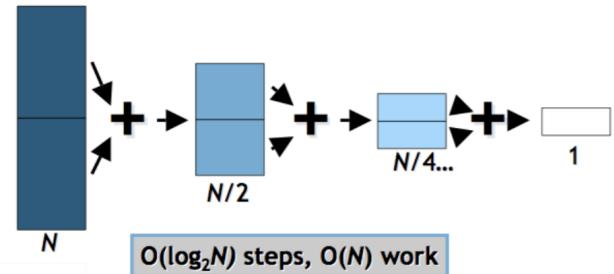
$O(\log_2 N)$ steps, $O(MN)$ work

```
for(i=0; i<N; ++i) {  
    accum += point[i]  
}
```

→ accum = $\text{reduce}(+, \text{point})$

Reduce

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
- $\text{Reduce}(\text{op}, s)$ returns a **op** a **op** b **op** $c \dots$ **op** z



```
for(i=0; i<N; ++i) {  
    accum += point[i]  
}
```

→ accum = $\text{reduce}(+, \text{point})$

Why must op be associative?

Scan (prefix sum)

Scan (prefix sum)

- Input
 - Associative operator **op**
 - Ordered set $s = [a, b, c, \dots z]$
 - Identity I

Scan (prefix sum)

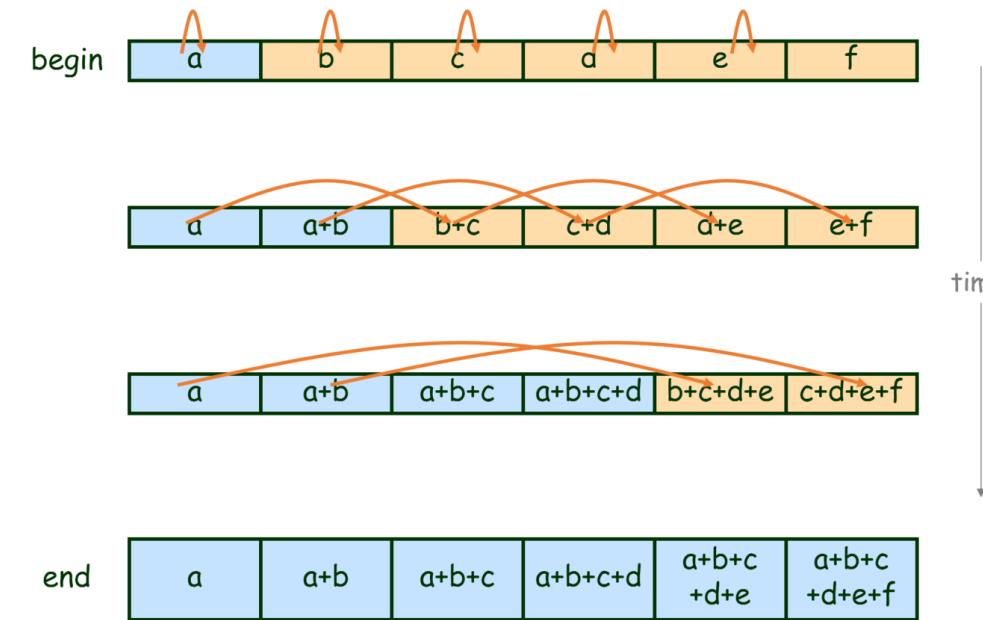
- Input
 - Associative operator op
 - Ordered set $s = [a, b, c, \dots z]$
 - Identity I
- $\text{scan}(\text{op}, s) = [I, a, (a \text{ op } b), (a \text{ op } b \text{ op } c) \dots]$

Scan (prefix sum)

- Input
 - Associative operator op
 - Ordered set $s = [a, b, c, \dots z]$
 - Identity I
- $\text{scan}(\text{op}, s) = [I, a, (a \text{ op } b), (a \text{ op } b \text{ op } c) \dots]$
- Scan is the workhorse of parallel algorithms:
 - Sort, histograms, sparse matrix, string compare, ...

Scan (prefix sum)

- Input
 - Associative operator op
 - Ordered set $s = [a, b, c, \dots z]$
 - Identity I
- $\text{scan}(\text{op}, s) = [I, a, (a \text{ op } b), (a \text{ op } b \text{ op } c) \dots]$
- Scan is the workhorse of parallel algorithms:
 - Sort, histograms, sparse matrix, string compare, ...



Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

```
var res = ints.GroupBy(x => x);
```

Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

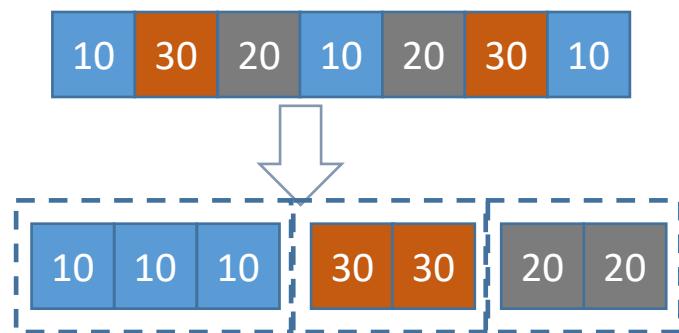
```
var res = ints.GroupBy(x => x);
```



Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

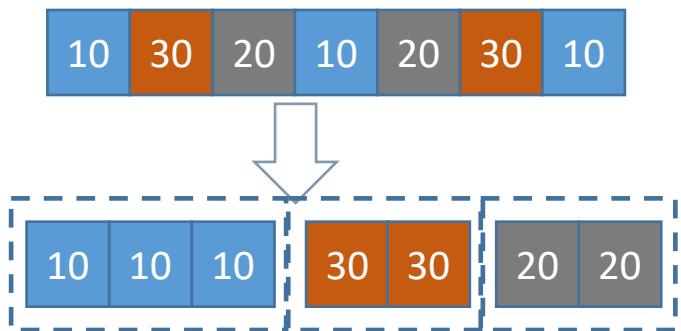
```
var res = ints.GroupBy(x => x);
```



Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

```
var res = ints.GroupBy(x => x);
```



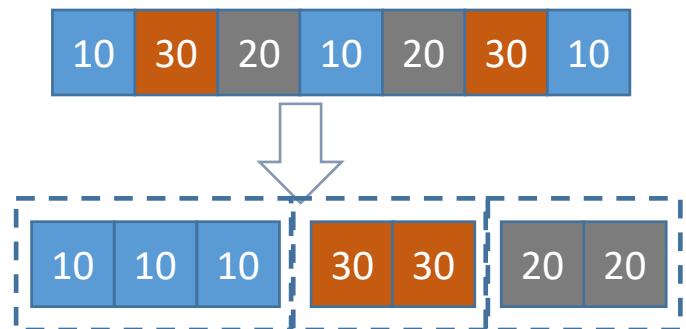
```
foreach(T elem in PF(ints))
{
    key      = KeyLambda(elem);
    group   = GetGroup(key) 
    group.Add(elem);
}
```

Example: Parallel GroupBy

- Group a collection by key
- Lambda function maps elements → key

```
var res = ints.GroupBy(x => x);
```

- *Insufficient Parallelism*
- *Requires synchronization*



A code snippet within a large blue circle with a diagonal slash through it, indicating it is incorrect or inefficient:

```
foreach (T elem in ints)
{
    key = KeyLambda(elem);
    group = GetGroup(key);
    group.Add(elem);
}
```

The code inside the circle shows a `foreach` loop that iterates over an array `ints`. For each element, it calls a `KeyLambda` function to determine a key, then uses this key to get a group from a collection, and finally adds the element to that group. This approach is marked as incorrect due to its lack of parallelism and potential synchronization issues.

Parallel GroupBy



Parallel GroupBy

Process each input element in parallel

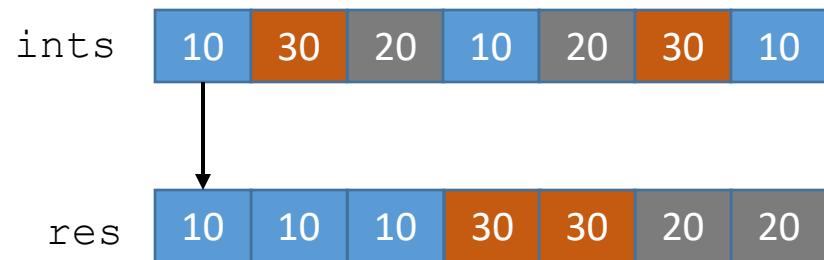
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

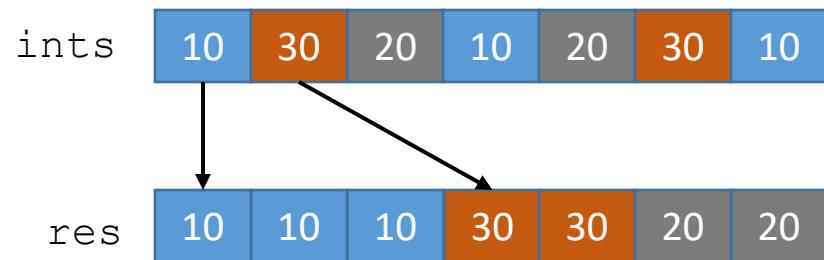
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

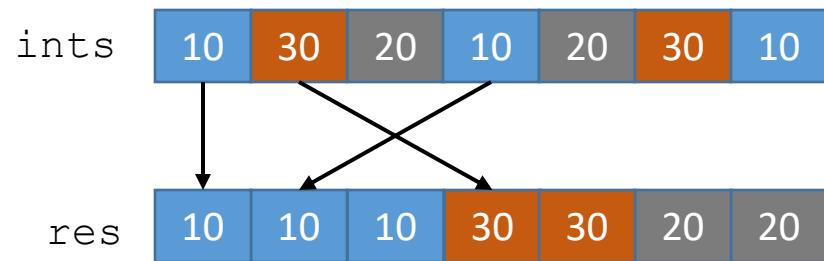
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

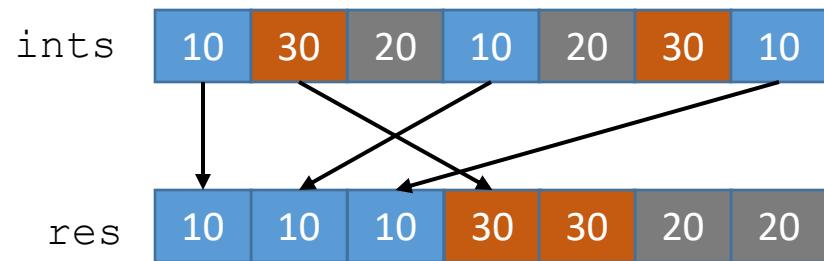
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

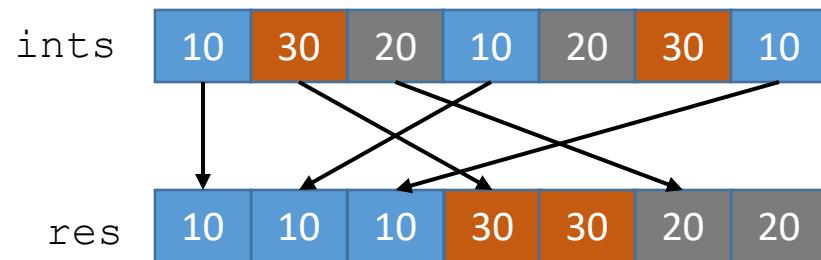
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

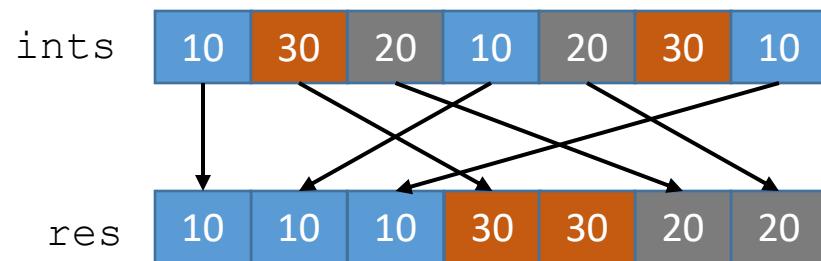
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

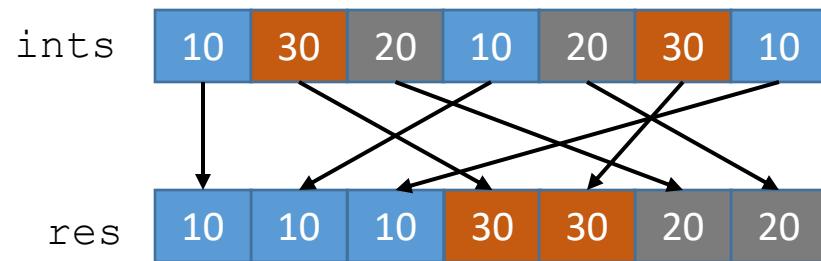
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

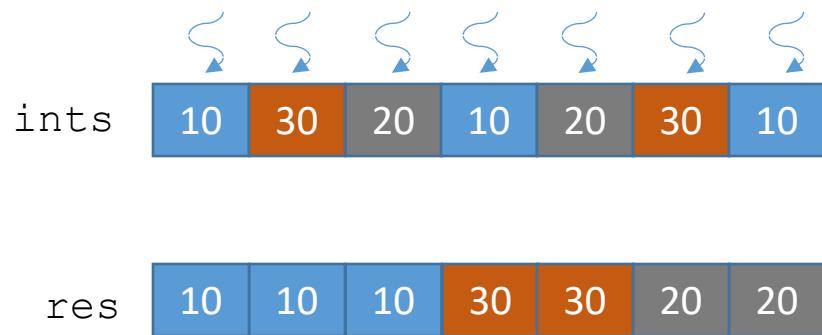
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

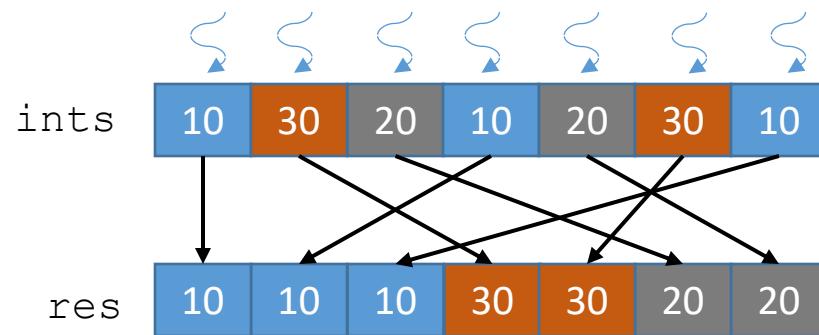
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

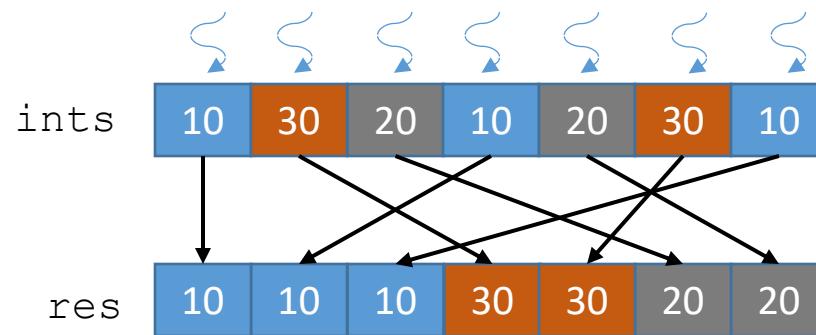
- grouping ~ shuffling
- input item → output offset such that groups are contiguous



Parallel GroupBy

Process each input element in parallel

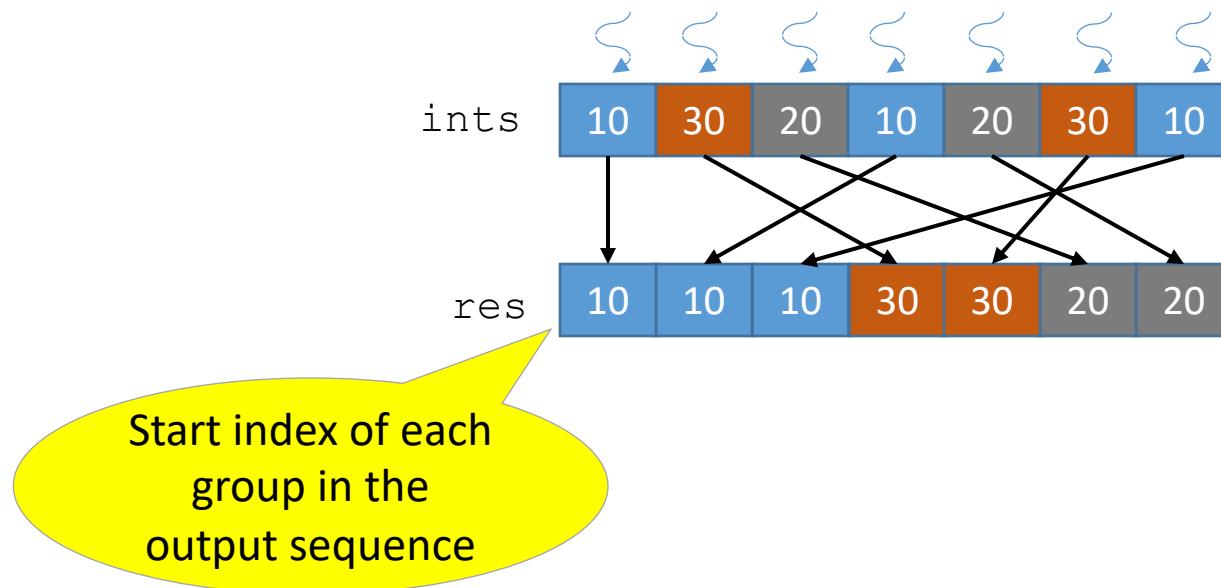
- grouping ~ shuffling
- input item → output offset such that groups are contiguous
- output offset = group offset + item number
- ... but how to get the group offset, item number?



Parallel GroupBy

Process each input element in parallel

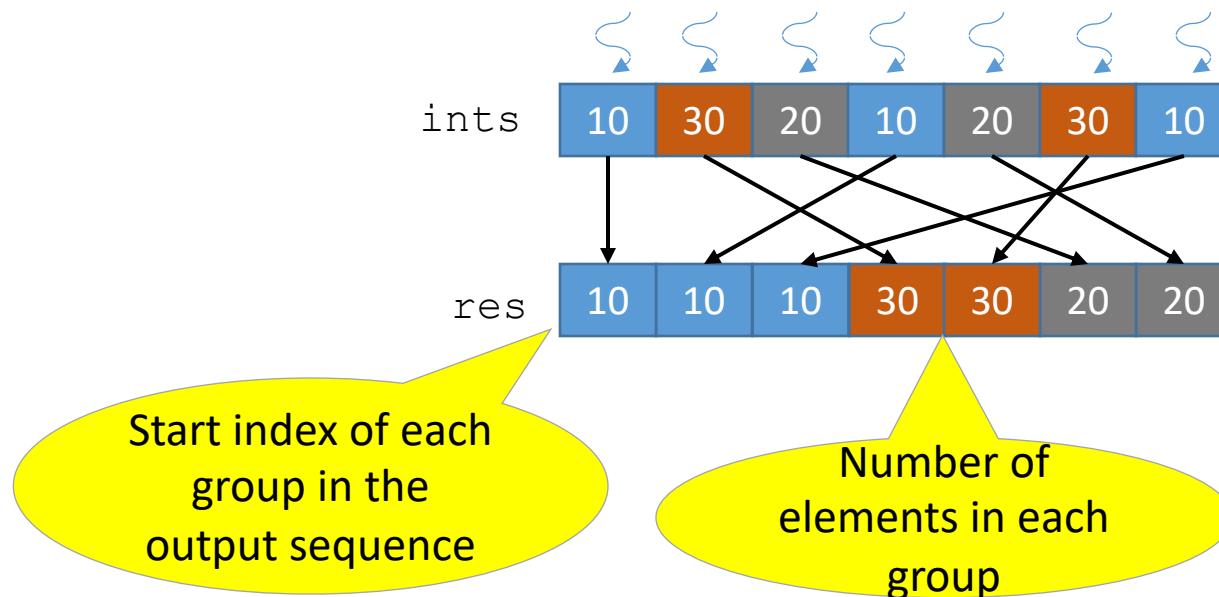
- grouping ~ shuffling
- input item → output offset such that groups are contiguous
- output offset = group offset + item number
- ... but how to get the group offset, item number?



Parallel GroupBy

Process each input element in parallel

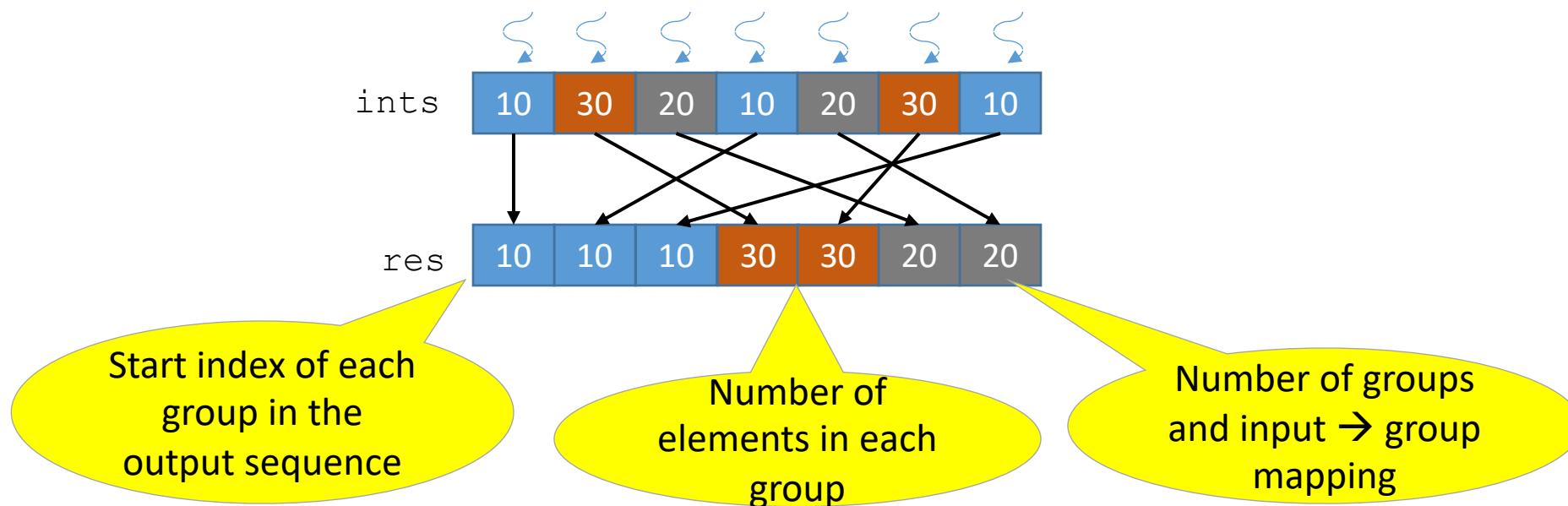
- grouping ~ shuffling
- input item → output offset such that groups are contiguous
- output offset = group offset + item number
- ... but how to get the group offset, item number?



Parallel GroupBy

Process each input element in parallel

- grouping ~ shuffling
- input item → output offset such that groups are contiguous
- output offset = group offset + item number
- ... but how to get the group offset, item number?

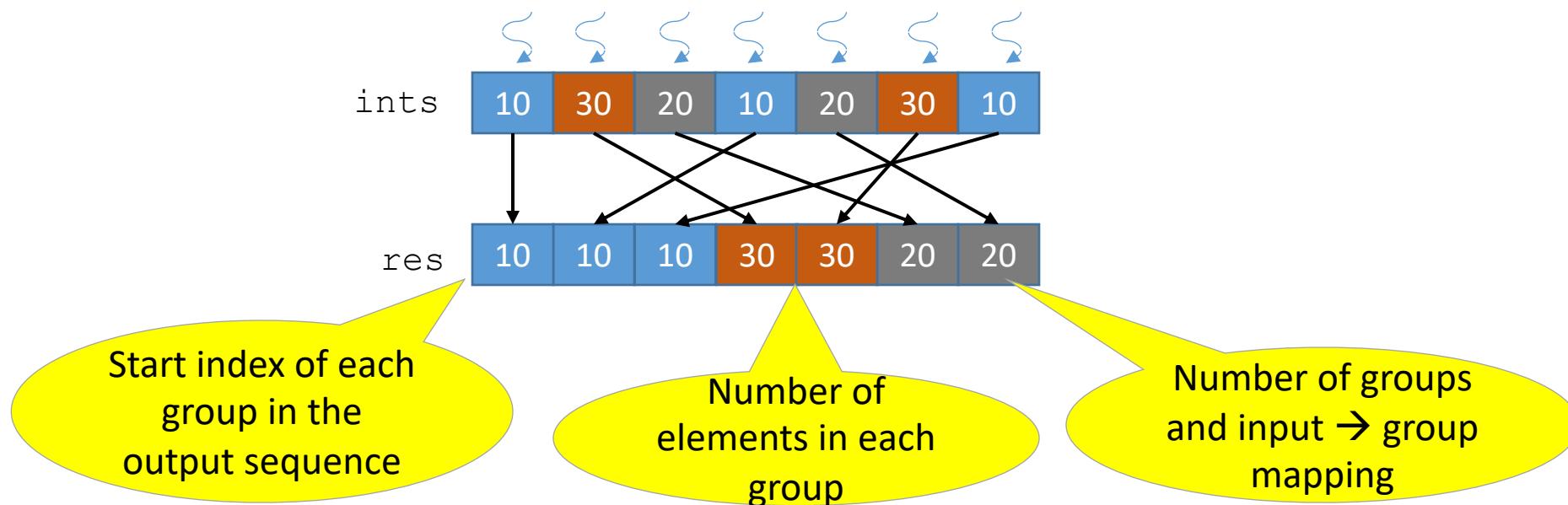


Parallel GroupBy

Process each input element in parallel



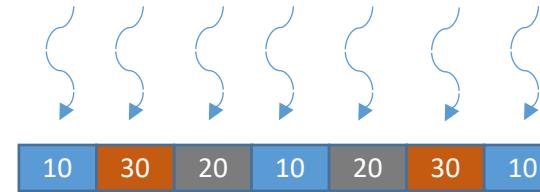
- grouping ~ shuffling
- input item → output offset such that groups are contiguous
- output offset = group offset + item number
- ... but how to get the group offset, item number?



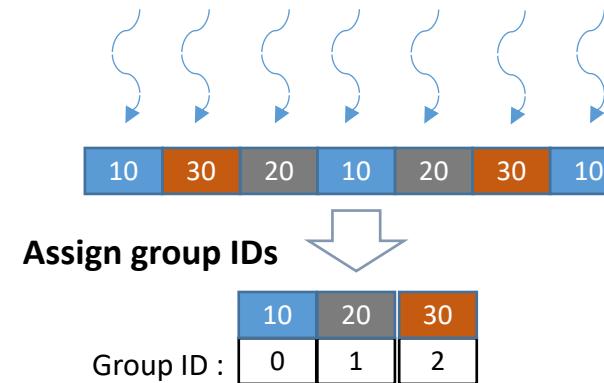
GroupBy using parallel primitives



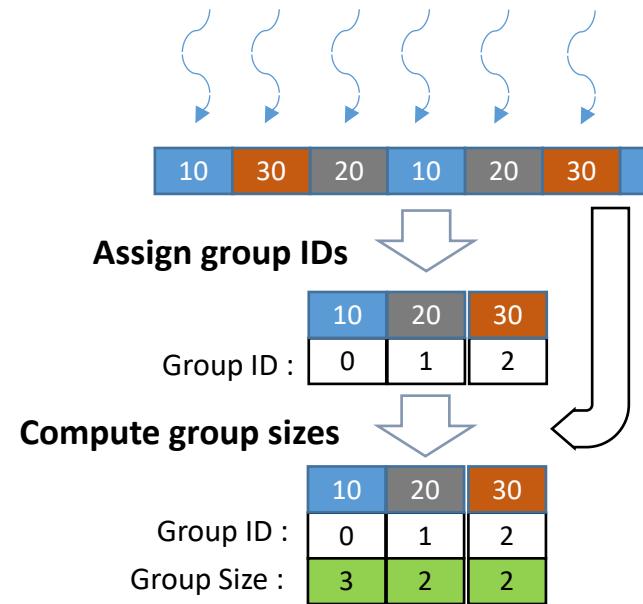
GroupBy using parallel primitives



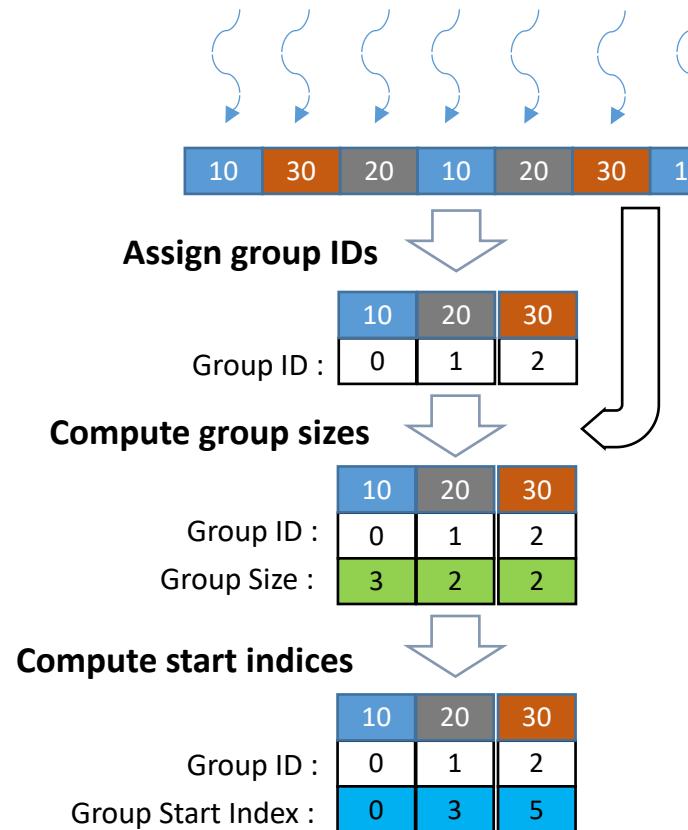
GroupBy using parallel primitives



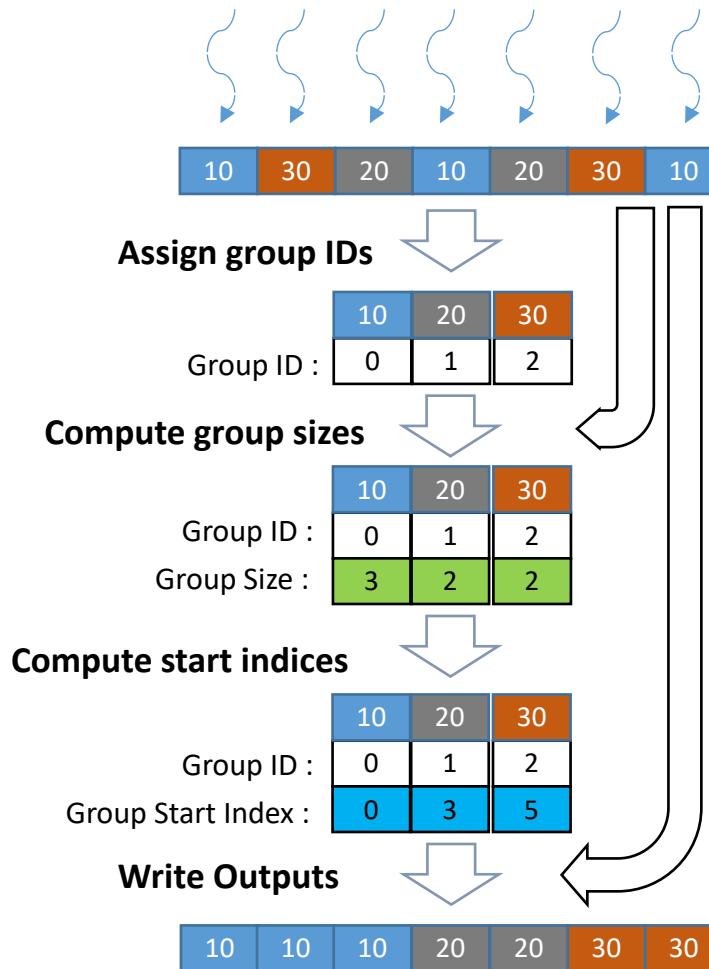
GroupBy using parallel primitives



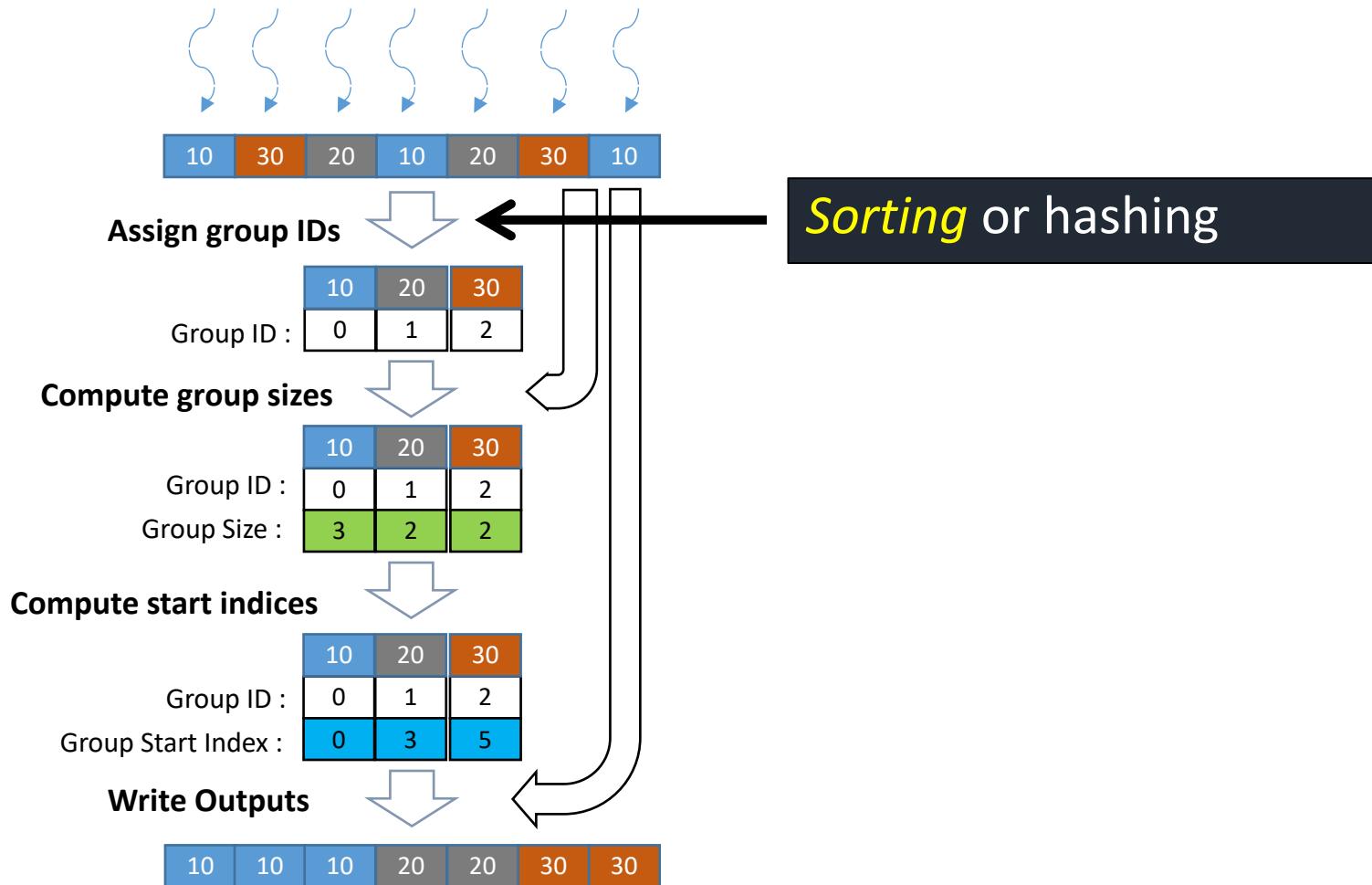
GroupBy using parallel primitives



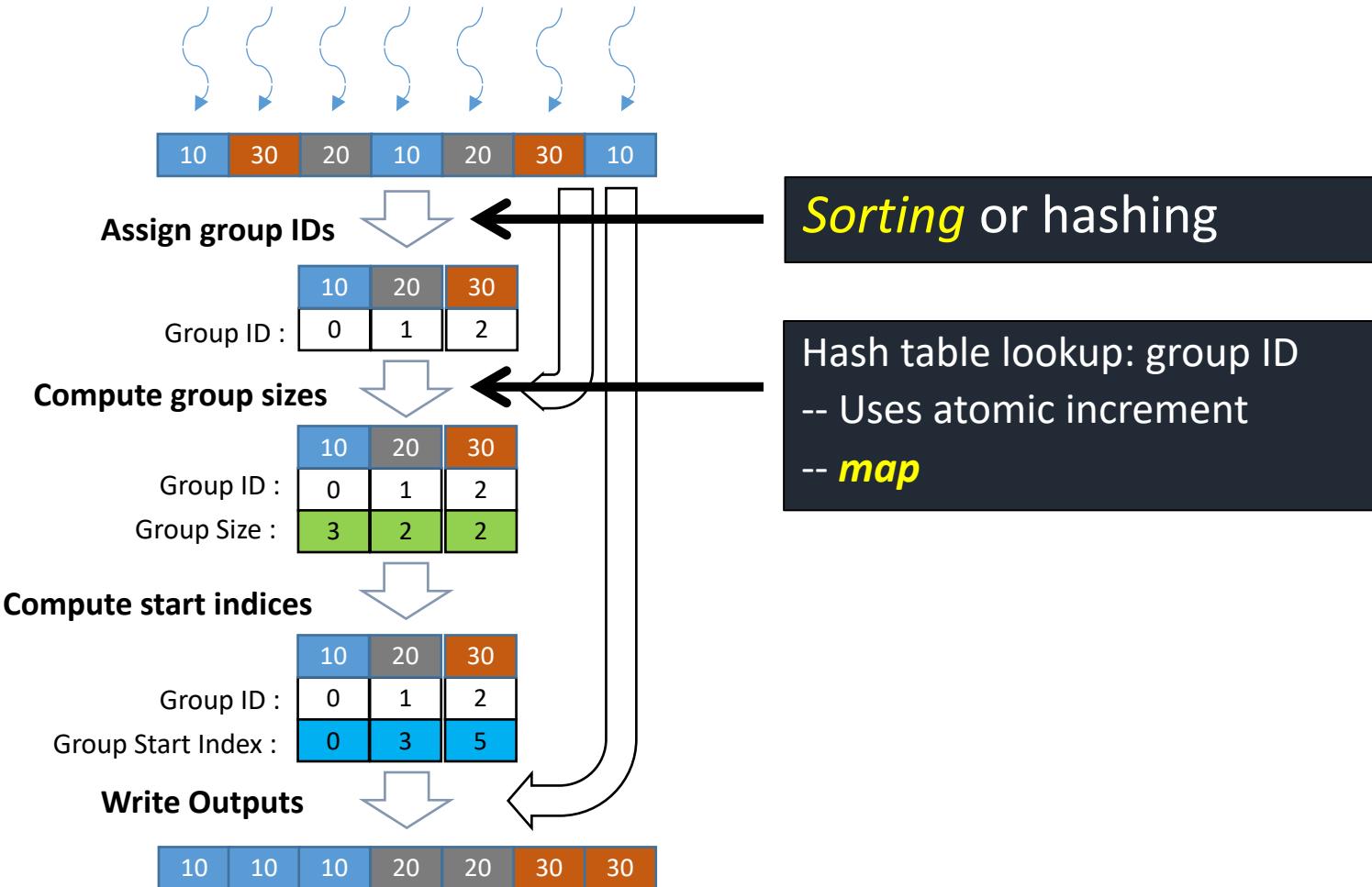
GroupBy using parallel primitives



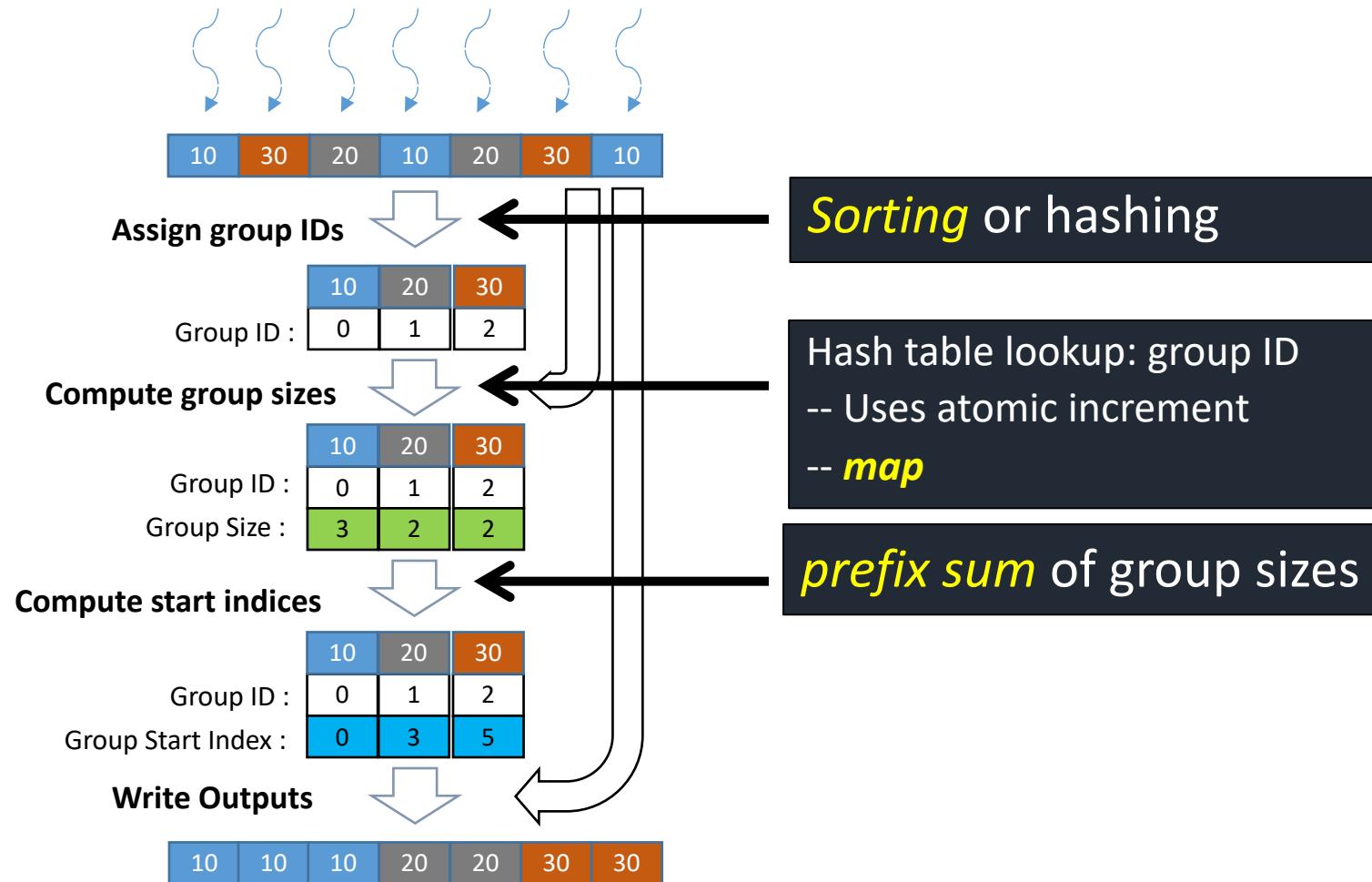
GroupBy using parallel primitives



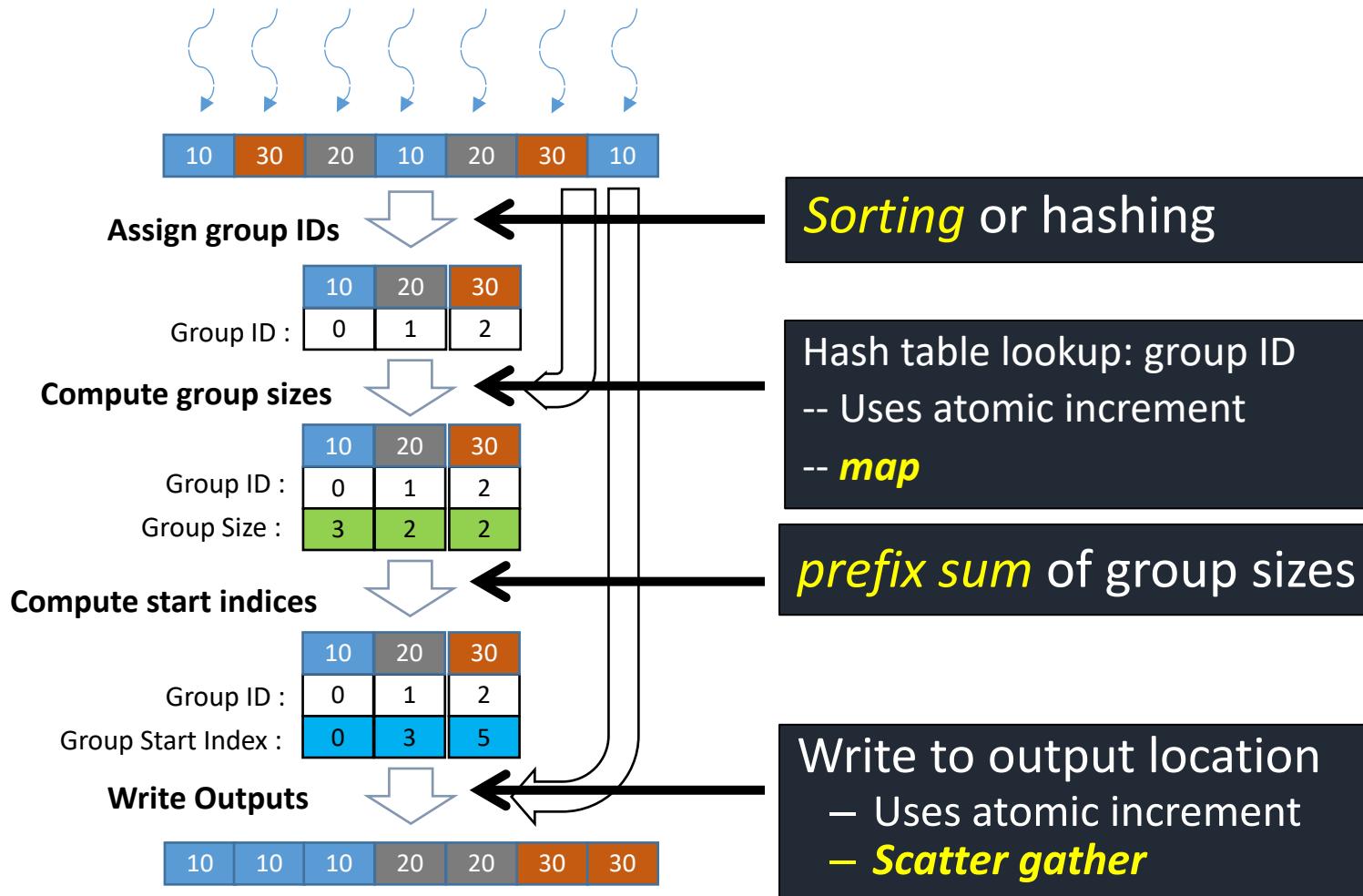
GroupBy using parallel primitives



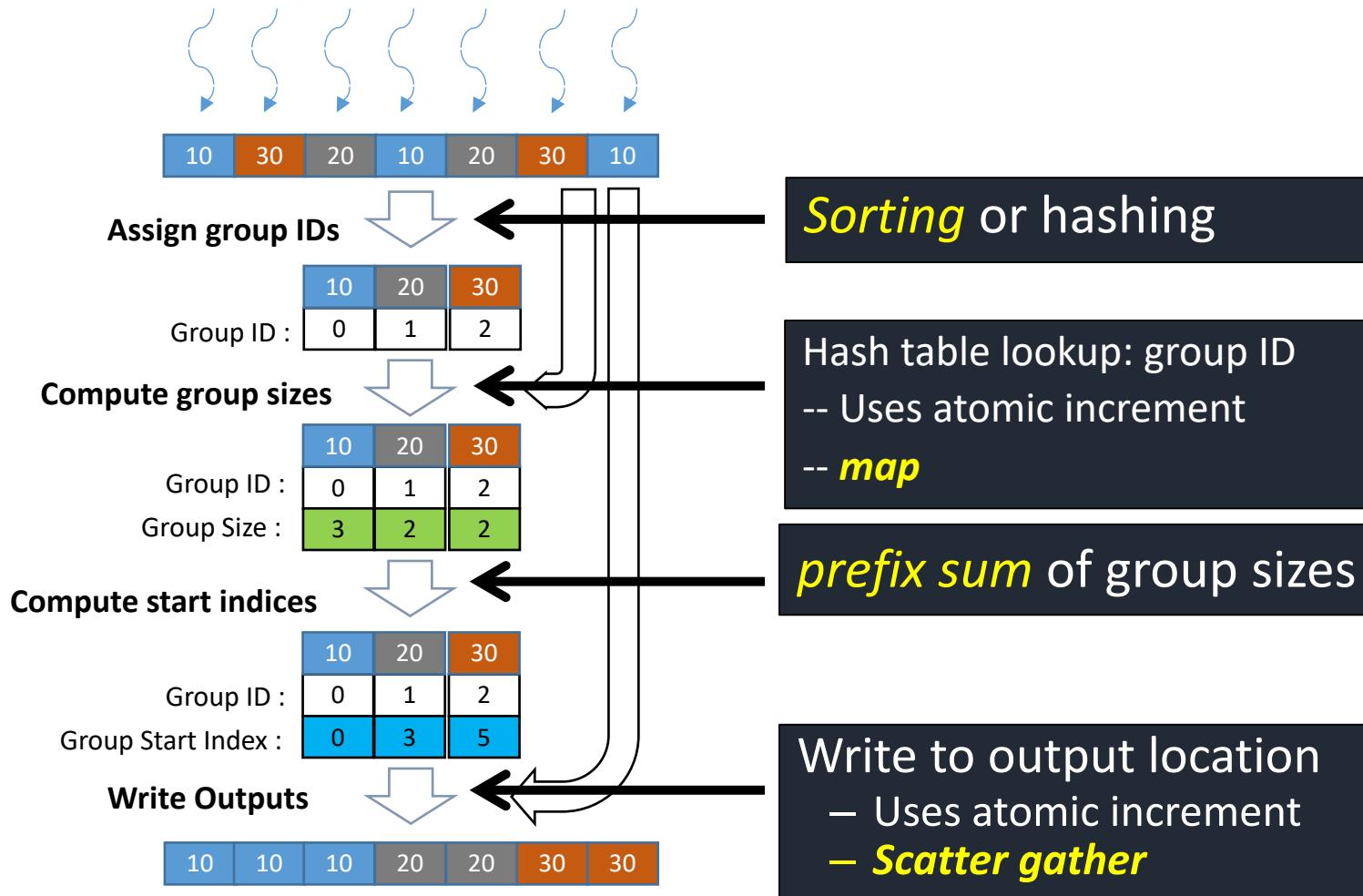
GroupBy using parallel primitives



GroupBy using parallel primitives



GroupBy using parallel primitives



We'll revisit after
more CUDA
background...

Sort

Many variations

- Enumeration sort
- Bitonic sort
- Merge sort
- Parallel Quicksort
- Radix sort
- Sample sort
- ...

Summary

Re-expressing apparently sequential algorithms as combinations of parallel patterns is a common technique when targeting GPUs

- Reductions
- Scans
- Re-orderings (scatter/gather)
- Sort
- Map