

# Accelerating Cosmology Codes

Volodymyr Kindratenko, Dylan Roeh, Guochun Shi (NCSA), Robert Brunner (Department of Astronomy), University of Illinois

## Two-point Angular Correlation

Two-point angular correlation function (TPACF),  $\omega(\theta)$ , is the frequency distribution of angular separations  $\theta$  between celestial objects in the interval  $(\theta, \theta + \Delta\theta)$

- angular distance between two points is  $\theta = \arccos(p \cdot q) = \arccos(x_p x_q + y_p y_q + z_p z_q)$

Example: Red (random data) are, on average, randomly distributed, black (observed data) are clustered

- random points:  $\omega(\theta)=0$
- observed points:  $\omega(\theta)>0$

TPACF can vary as a function of angular distance,  $\theta$  (blue circles)

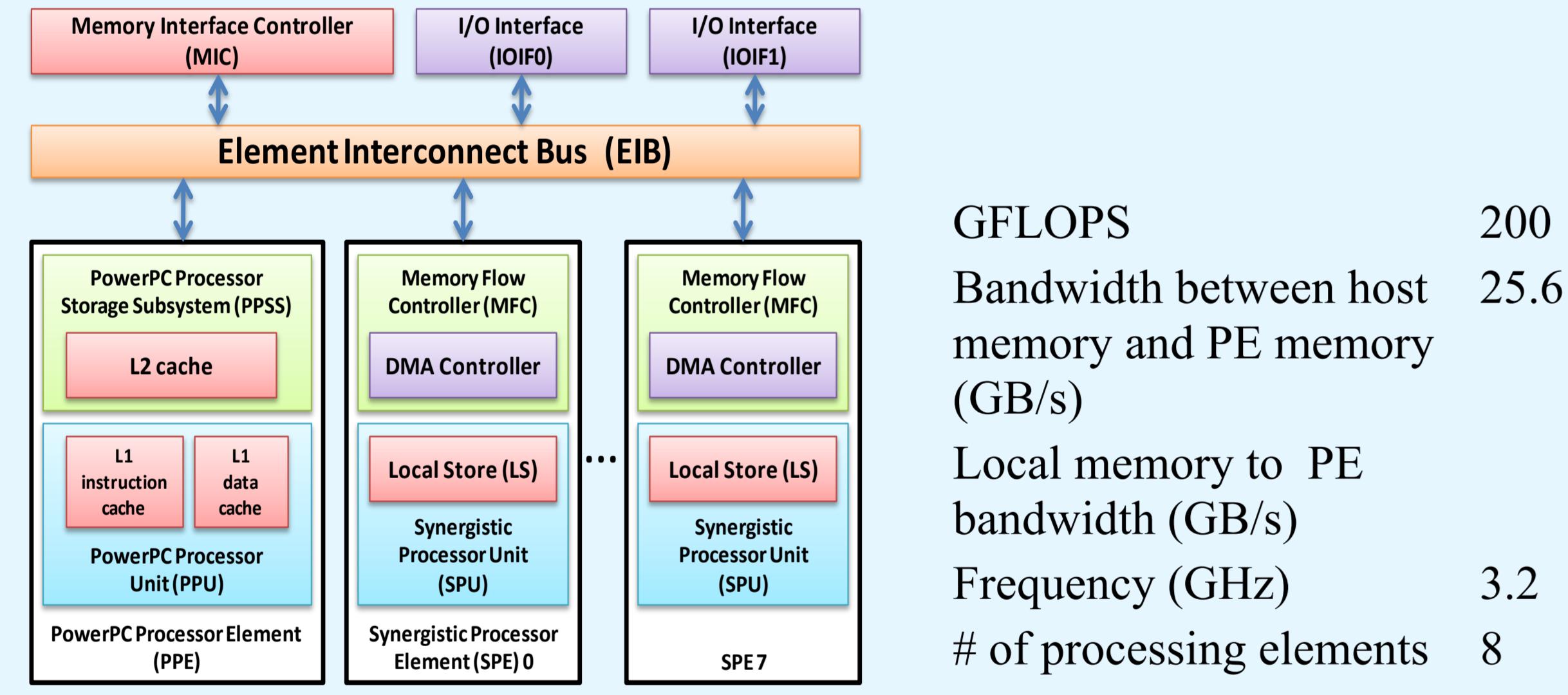
- random:  $\omega(\theta)=0$  on all scales
- observed:  $\omega(\theta)$  is larger on smaller scales

TPACF is computed using modified Landy & Szalay estimator

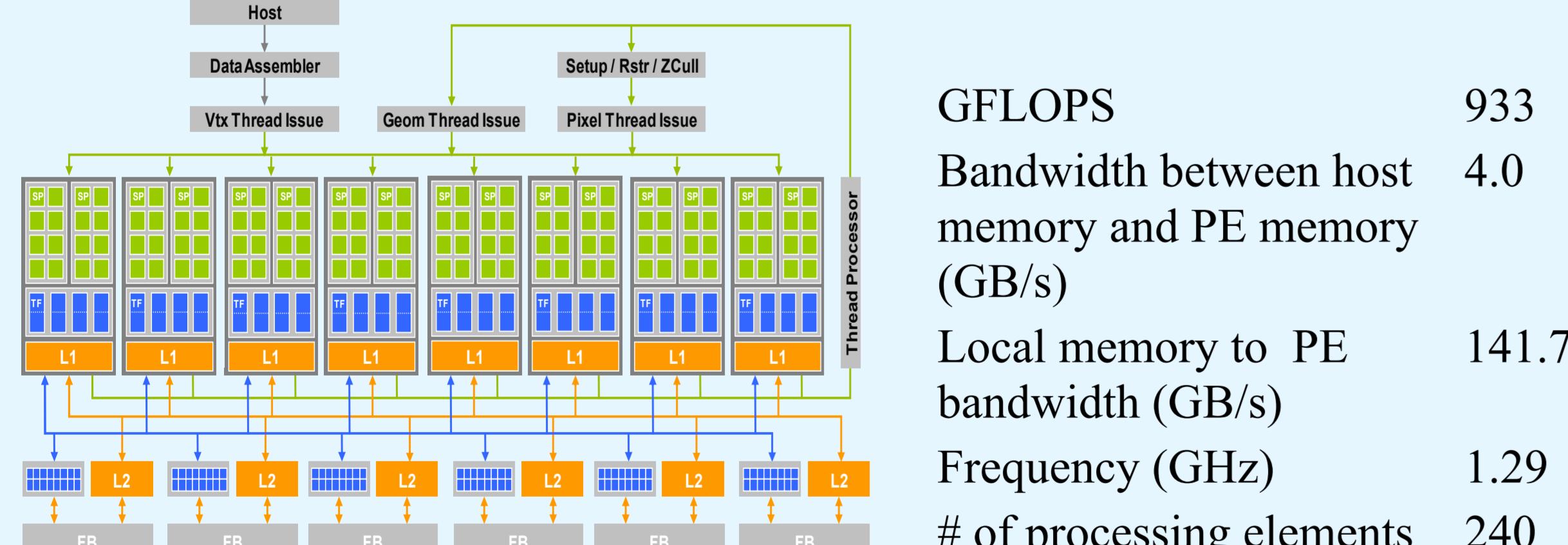
$$\omega(\theta) = \frac{n_R \cdot DD(\theta) - 2 \sum_{i=0}^{n_R-1} DR_i(\theta)}{\sum_{i=0}^{n_R-1} RR_i(\theta)} + 1$$

## Systems

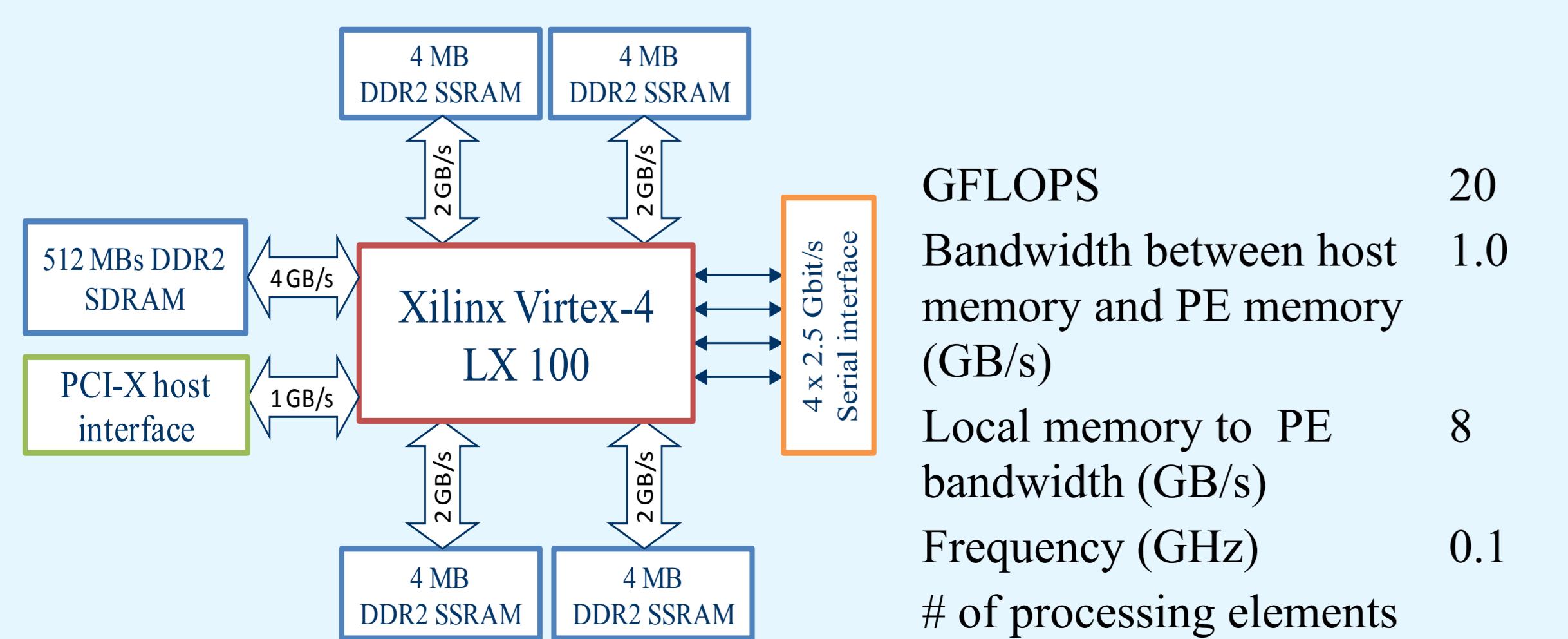
### Cell processor



### NVIDIA GeForce GTX 280 GPU



### Nallatech H101



## Algorithm

*Algorithm 1: autocorrelation (DD and RR bin counts)*

**Input:** 1) set of Cartesian coordinates of  $N$  points  $x_1, \dots, x_N$  on the celestial sphere; 2) for each point  $x_i$ , jackknife sample number,  $1 \leq s_i \leq K$ , from which  $x_i$  is removed; 3) set of  $M$  bins given by their boundaries:  $[\theta_0, \theta_1], [\theta_1, \theta_2], \dots, [\theta_{M-1}, \theta_M]$ .

**Output:** for each bin in every jackknife sample, the number of unique pairs of points  $(x_i, x_j)$  for which the angular separation  $\theta(x_i, x_j)$  is in the respective bin.

```

1 for i=1,..., N-1 do
2   for j=i+1,..., N do
3     l←binmap(θ(xi,xj), θ)
4     for k=1,..., K do
5       if si ≠ k do Bk,l← Bk,l + 1

```

*Algorithm 2: cross-correlation (DR bin counts)*

**Input:** 1) 2 sets of Cartesian coordinates of  $N$  points  $x_1, \dots, x_N$  and  $y_1, \dots, y_N$  on the celestial sphere; 2) for each point  $x_i$ , jackknife sample number,  $0 \leq s_i \leq K-1$ , from which  $x_i$  is removed; 3) set of  $M$  bins given by their boundaries:  $[\theta_0, \theta_1], [\theta_1, \theta_2], \dots, [\theta_{M-1}, \theta_M]$ .

**Output:** for each bin in every jackknife sample, the number of unique pairs of points  $(x_i, y_j)$  for which the angular separation  $\theta(x_i, y_j)$  is in the respective bin.

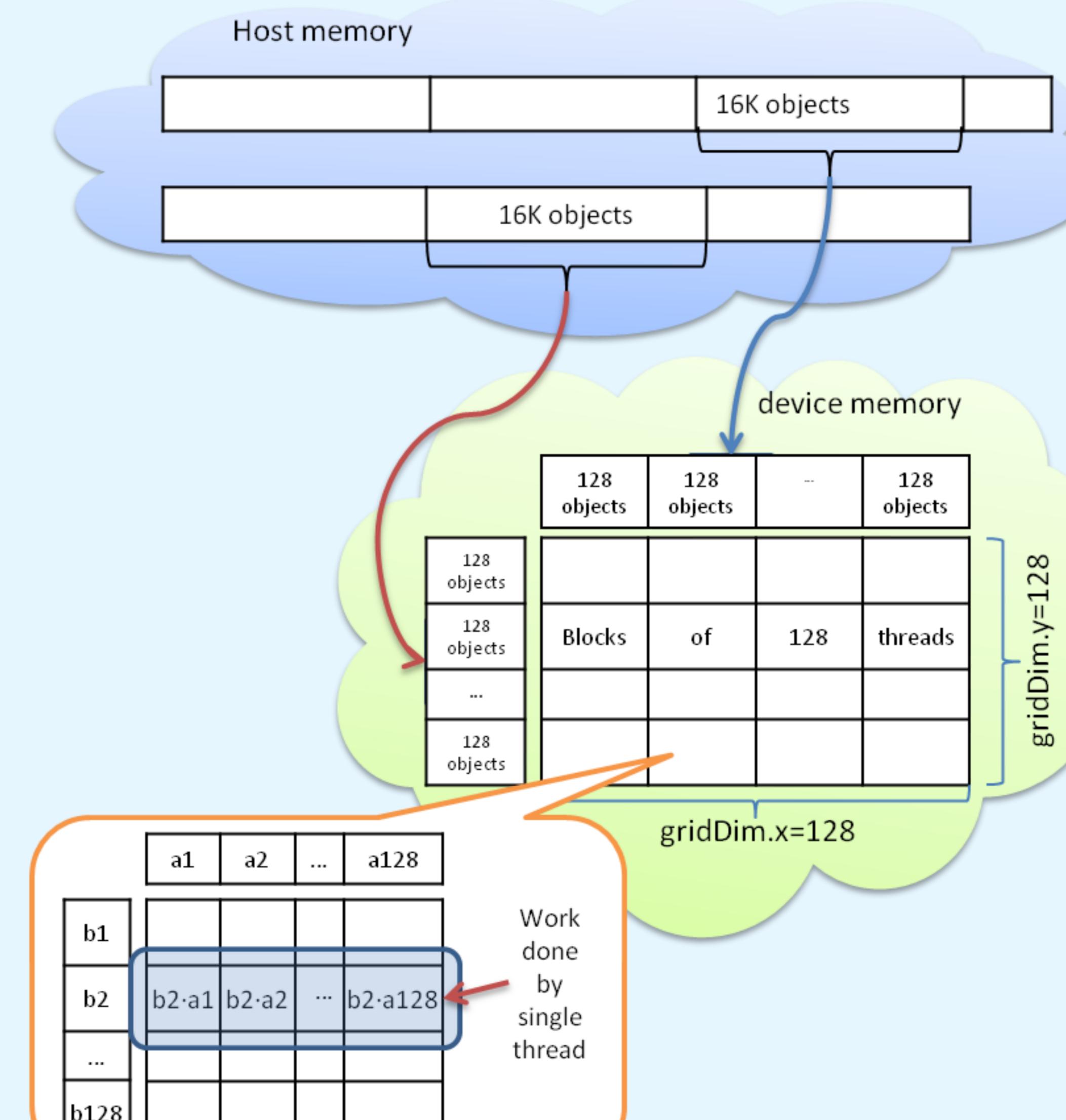
```

1 for i=1,..., N do
2   for j= 1,..., N do
3     l←binmap(θ(xi,yj), θ)
4     for k=1,..., K do
5       if si ≠ k do Bk,l← Bk,l + 1

```

## GPU (CUDA) Implementation

### Bin assignment kernel



### Histogram kernel

Adaptation of Podlozhnyuk, V. "64-bin Histogram." 2007.  
<http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/histogram64/doc/histogram64.pdf>.

## FPGA (Dime-C) Implementation

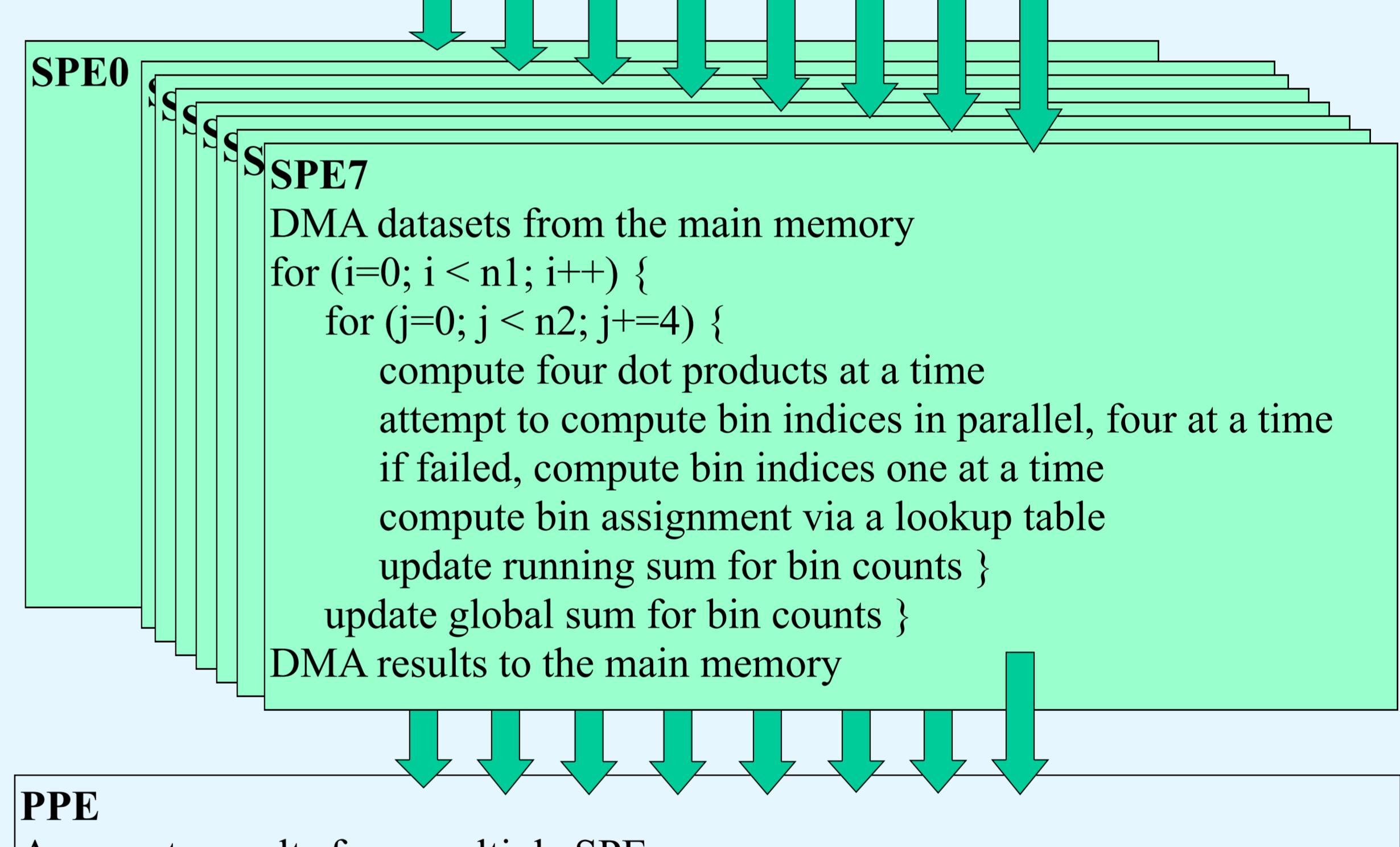
```

53 for (i = 0; i < i_end; i++) { // loop thru one dataset
54   x1 = X[i]; y1 = Y1[i]; z1 = Z1[i]; jk = (int)JK[i];
55   if (doSelf) j_start = i+1; else j_start = 0;
56   for (j = j_start; j < n2; j++) { // loop thru second dataset
57     if (doSelf) offset = j; else offset = n1 + j;
58     x2 = X2[offset]; y2 = Y2[offset]; z2 = Z2[offset];
59     dot = x1 * x2 + y1 * y2 + z1 * z2; // dot product
60     // unrolled binary search
61     if (dot < binb15) { if (dot < binb23) { if (dot < binb27) { if (dot < binb30) indx = 31;
62       else indx = 30; } else { if (dot < binb28) indx = 29; else indx = 28; } } else { if (dot < binb25) {
63       if (doSelf) offset = j; else offset = n1 + j;
64       if (dot < binb26) indx = 27; else indx = 26; } else { if (dot < binb24) indx = 25; else indx = 24; } }
65     else { if (dot < binb19) { if (dot < binb21) { if (dot < binb22) indx = 23; else indx = 22; } else {
66       if (dot < binb20) indx = 21; else indx = 20; } } else { if (dot < binb17) { if (dot < binb18) indx = 19;
67       else indx = 18; } else { if (dot < binb16) indx = 17; else indx = 16; } } }
68     else { if (dot < binb07) { if (dot < binb11) { if (dot < binb14) indx = 15; else indx = 14; } else {
69       if (dot < binb12) indx = 13; else indx = 12; } } else { if (dot < binb09) { if (dot < binb10) indx = 11;
70       else indx = 10; } else { if (dot < binb08) indx = 9; else indx = 8; } } } else { if (dot < binb03) { if (dot < binb05) {
71       if (dot < binb06) indx = 7; else indx = 6; } else { if (dot < binb04) indx = 5; else indx = 4; } } else {
72       if (dot < binb01) { if (dot < binb02) indx = 3; else indx = 2; } else { if (dot < binb00) indx = 1; else indx = 0; } }
73     bin_bank = j % 4; // update bin values
74     if ((jk != 0) val0 = 1; else val0 = 0; // update corresponding bin for jk=0
75     if (bin_bank == 0) binv1_00[indx] = binv1_00[indx] + val0;
76     else if (bin_bank == 1) binv2_00[indx] = binv2_00[indx] + val0;
77     else if (bin_bank == 2) binv3_00[indx] = binv3_00[indx] + val0;
78     else binv4_00[indx] = binv4_00[indx] + val0;
79   }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
599 }
```

## Cell B/E. Implementation

### PPE

Convert data layout from array of structures to vectors of coordinates  
Divide work between 8 SPEs



## Results

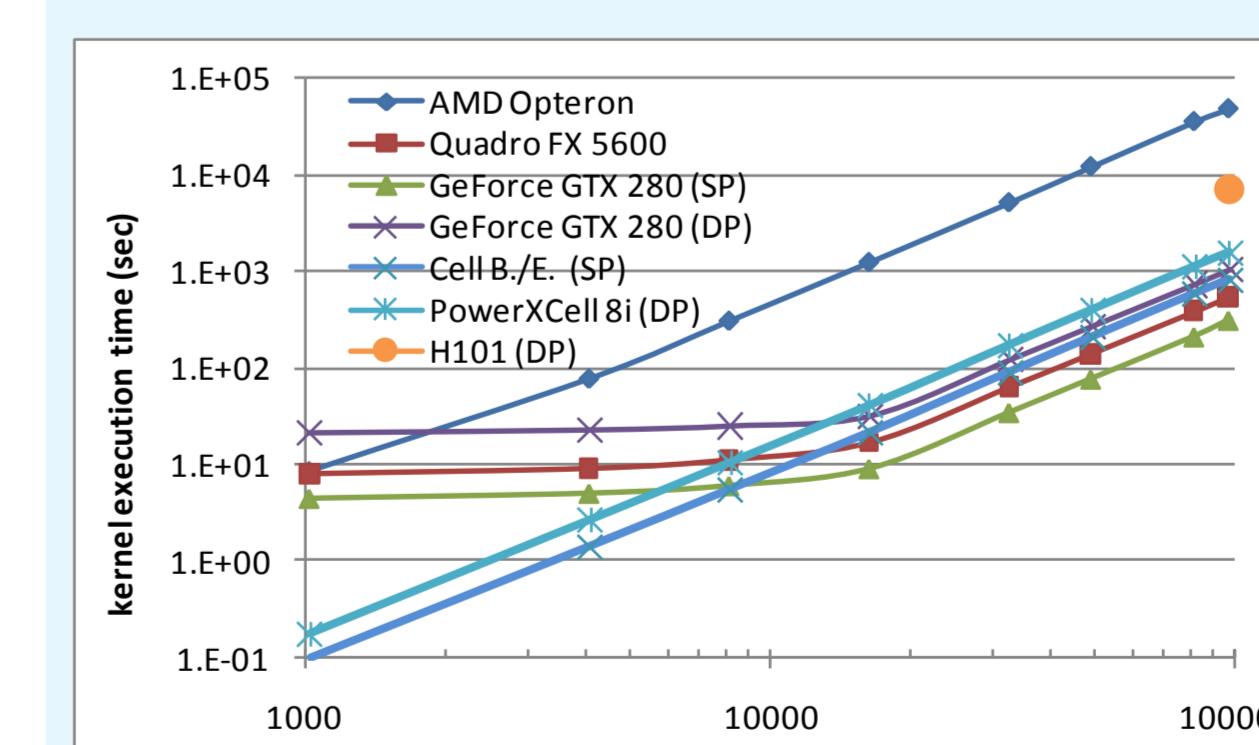


Fig 1. Execution time vs. dataset size

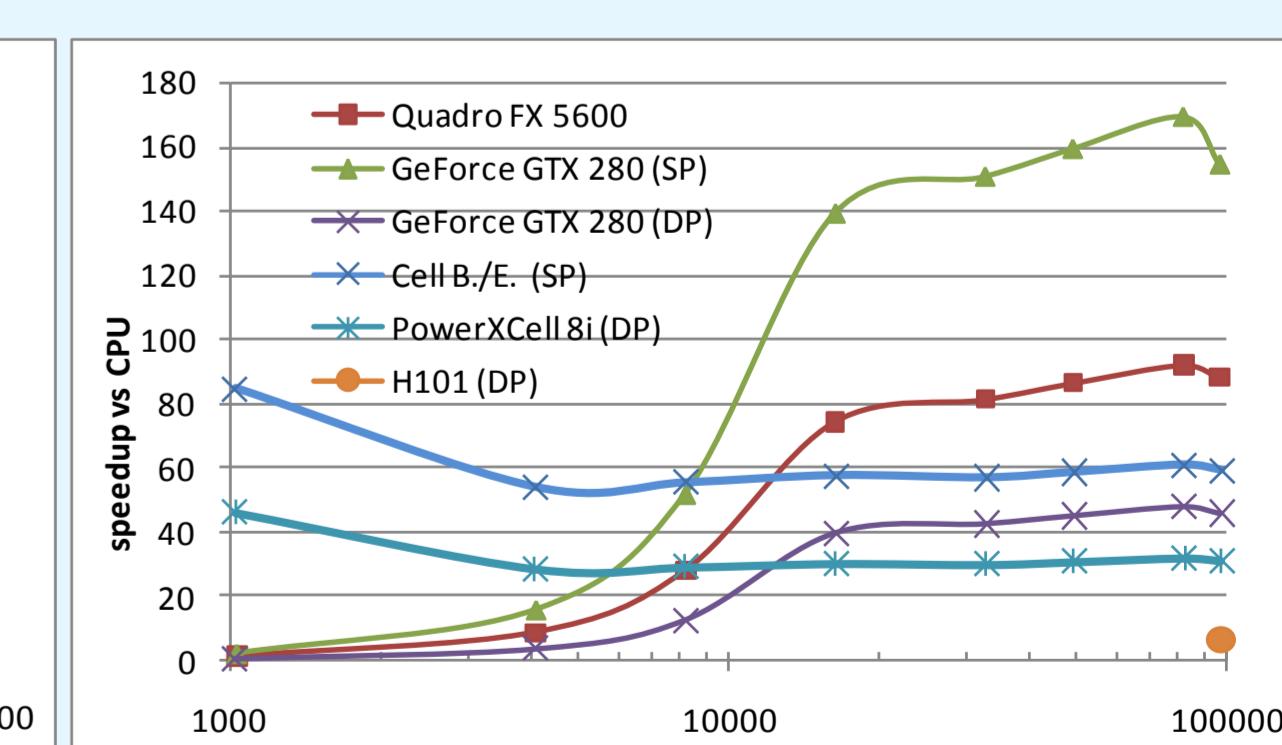


Fig 2. Speedup vs. dataset size

H101 (DP) speedup: 6.8×  
PowerXCell 8i (DP) speedup: 30.9×  
GeForce GTX 280 (DP) speedup: 54.5×  
GeForce GTX 280 (SP) speedup: 154.1×  
Quadro FX 5600 (SP) speedup: 88.2×  
Cell B/E- (SP) speedup: 59.1×