Are there any functional or issues in the given code? Document your findings.

if (!(arr)[i]) PERROR\_GOTO(label); \

the paranthesis dont really make are necessary here, it should be arr[i] instead of (arr)[i]

(arr)[i] = malloc(sizeof(\*\*(arr)) \* n); \

use calloc:

* the cleanup logic doesnt work with malloc

should use calloc to check for null later, if 1d pointers have not been malloced

* c and local\_res are not initialized to 0s
* if (!local\_res) PERROR\_GOTO(error\_c); -> wrong error message and wrong cleanup

-> if (!local\_res) PERROR\_GOTO(error\_local\_res);

* too much #pragma omp parallel, only once for each loop structure
* collapse should be tried used to really use parallelism for the nested loops also maybe reduction is more efficient
* wrong datatype for \*\* pointers; must be long instead of int
* #define INIT\_ARRAY(arr, label) \

do { \

if (!(arr)) PERROR\_GOTO(label); \

for (long i = 0; i < n; ++i) { \

arr[i] = calloc(n, sizeof(\*\*(arr))); \

if (!arr[i]) PERROR\_GOTO(label); \

} \

} while (0)

* Change type to long long here, because RAND\_MAX \*RAND\_MAX can be bigger than LONG\_MAX if summed up

Improve the performance of the given source code, fix any functional bugs you find.

Benchmark your modification of the code and the original implementation with 1, 2, 4, 6, and 12 threads on LCC3 using n=1,500. Has your effort paid off?  
  
  
 unsigned long long res = 0;

#pragma omp parallel for reduction(+ : res) default(none) shared(n, c)

for (long i = 0; i < n; ++i) {

for (long j = 0; j < n; ++j) {

res += c[i][j];

}

}

tried to use a reduction, but seems like I always overflowed (I get the same super high value all of the time

* Collapse didn’t speed up the program

Original times:

A graph with colorful lines

AI-generated content may be incorrect.

Enter your shortest wall clock time for 12 threads and n=1,500 on LCC3 to the comparison spreadsheet linked on Discord.