1. La solució habitualment més eficient és fer servir 1 thread per cada nucli, en el cas del cluster de la UOC els nodes tenen 4 nuclis. En cas de crear més d'un thread per nucli el sobrecost de creació dels threads i els canvis de context pot penalitzar el rendiment.

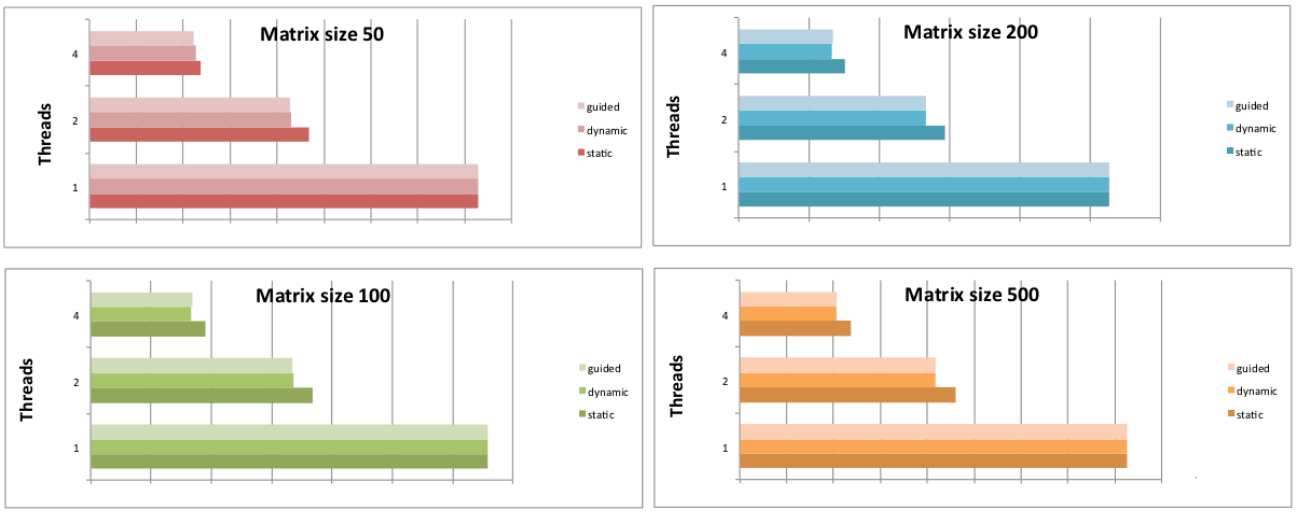
2.

```
#pragma omp parallel for reduction (+:result)
for(i=0; i<N; i++){
      result = result + a[i];
}
```

3. Diverses implementacions són possibles, es pot incloure la paral.lelització de la inicialització de les matrius. Es poden fer servir les polítiques concretes (`static, dynamic, guided`) o mitjançant "runtime" com es mostra a continuació. La principal necessitat és la privatització dels índex utilitzats. També es pot afegir la clàusula `collapse(3)`.

```
#pragma omp parallel for private (j,k) schedule(runtime)
for (i=0;i<SIZE;i++)
   for(j=0;j<SIZE;j++)
    for(k=0;k<SIZE;k++)
      mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];
```

Exemples de comportament amb les diferents polítiques es poden veure a continuació com a referència:



Preguntes d'avaluació de rendiment:

Al realitzar execucions de les dues diferents versions de la multiplicació de matrius (tamany 1000) podeu veure variabilitat i que la segona versió obté pitjors resultats.

#execució	mm (s)	mm2 (s)
1	11,23	15,78
2	11.25	12,77
3	11,24	15,76
4	11,27	12,87

L'execució de flops i flops2 mostra diferències com es mostra a continuació:

	flops	flops2
real_time	11,2617	12,7162
proc_time	11,2351	12,6854
MFLOPS	178,2516	157,9695

La primera versió proporciona més MFLOPS ja que la segona ha de dedicar més temps al moviment de dades a través de la jerarquia de memòria. Com que les matrius s'emmagatzemen de forma seqüencial de forma unidimensional, és d'esperar que l'accés a posicions de memòria de forma ordenada proporcionaran millor rendiment.

Alguns comptadors significatius son aquells relacionats amb la utilització de la jerarquia de memòria com ara la fallades de cache. Tres exemples son:

PAPI_LST_INS → per a mesurar el número total d'instruccions de lectura/escriptura
PAPI_L1_DCM → per a mesurar les fallades de cache de nivell 1
PAPI_L2_DCM → per a mesurar les fallades de cache de nivell 2

A continuació es mostren possibles resultats on es veu que la variabilitat principal es troba en les fallades de cache:

	counters	counters2
PAPI_TOT_INS	33.074.987.583	33.074.989.019
PAPI_FP_OPS	9219	9219
PAPI_L1_DCM	1.873.629.237	3.043.753.161
PAPI_L2_DCM	73.446.810	232.259.159
PAPI_LST_INS	15.043.028.868	15.043.031.965

Amb la versió OpenMP, uns possibles resultats son els següents, on s'observen els mateixos patrons:

	counters (omp)	counters2 (omp)
exec time	4,48	6,88
PAPI_TOT_INS	11.022.923.445	11.023.486.554
PAPI_FP_OPS	9219	9219
PAPI_L1_DCM	477.951.858	868.433.699
PAPI_L2_DCM	19.341.759	307.680.785
PAPI_LST_INS	5.763.670.624	5.763.788.544