A continuació, us presente un problema de programación amb OpenMP resolt d'un examen:
Problema: Se pretende paralelizar el siguiente código en C mediante OMP:

```
for(i=1; i<n; i++)
  for(j=0; j<n-1; j++)
    y[i][j] = y[i-1][j] + a*x[i][j+1];
```

a) Suponiendo datos en coma flotante, obtener el coste en FLOP de este algoritmo, mostrando todos los cálculos intermedios, y el coste asintótico. Suponemos que todas las operaciones tienen el mismo coste de 1 FLOP.
b) Proponer una paralelización de este código utilizando OpenMP. Escribir el código paralelizado y justificar la elección.

a) Para obtener el coste en FLOP (operaciones en coma flotante) de este algoritmo comenzamos analizando el interior del código, donde están las operaciones aritméticas, y después analizamos las veces que estas se realizan a partir del número de iteraciones de los bucles.

```
for(i=1; i<n; i++)                    → n-1 iteraciones
  for(j=0; j<n-1; j++)                → n-1 iteraciones
    y[i][j] = y[i-1][j] + a*x[i][j+1]; → coste de 2 operaciones: 2FLOP
```

Luego del análisis obtenemos que el coste de operaciones será: $2(n-1)^2$ FLOP
Si n tiene un valor muy grande obtenemos el valor asintótico, impuesto por el término con la potencia de mayor orden: $O(n^2)$, aunque también podemos dejar el término escalar $O(2n^2)$
b) Antes de decidir la paralelización de un código hay que analizar sus dependencias para ver si la paralelización es factible. Si nos fijamos en la expresión de la operación del bucle:

```
y[i][j] = y[i-1][j] + a*x[i][j+1];
```

vemos que hay una escritura en y[i][j] y una lectura de y[i-1][j], un término anterior en el bucle i, por lo que habrá un riesgo RAW si hacemos operaciones fuera orden sobre esta variable en el bucle i, por lo que se concluye que el bucle de I no es paralelizable.
En el caso del bucle j vemos que aparece para la variable y, términos de j a ambos lados de la expresión, no existiendo por tanto ningún tipo de dependencia y pudiendo ser paralelizada. También podemos ver que los bucles i y j son intercambiables entre sí, siendo el número de iteraciones es el mismo para ambos, y que el coste de cada iteración es constante, por lo que se propone la siguiente paralelización de OpenMP:

```
#pragma omp parallel for private (j,i) shared(y,x,a,n) schedule (static)
    for(j=0; j<n-1; j++)
      for(i=1; i<n; i++)
        y[i][j] = y[i-1][j] + a*x[i][j+1];
```

PD: Notar que al paralelizar el bucle externo (ahora j) el bucle interno (i) se ejecuta de forma secuencial en cada hilo, por lo que no hay ningún problema con la dependencia de datos, pues ahora las iteraciones se realizan en orden.
La paralelización del bucle j sin el intercambio de bucle sería posible, pero menos eficiente al incrementarse los sobrecostes de creación y sincronización los hilos.

VALENCIÀ

Utilitzant el model vist en el problema anterior, resolgueu el següent problema d'examen:
Problema: Es pretén paral·lelitzar el següent codi en C mitjançant OpenMP:

```
float dist(int n,float *d1,float *d2) {
  float sum=0.0;
  float x1,x2;
  for(int i=0;i<n;i++) {
    x1=d1[i];
    for(int j=i+1;j<n;j++) {
      x2=d2[j];
      sum+=pow((x1-x2),2);
    }
  }
  return sqrt(sum)/(n*n);
}
```

a) (0.5) Obtín el cost en FLOP d'este algoritme, mostrant tots els càlculs intermedis, i el cost asimptòtic. Suposem que les operacions tenen un cost de 1 FLOP, les funcions POW i SQRT un cost de 3 FLOP, i no es té en compte el sobrecost de las sentències "for".
b) (0.5) Indica la dependència de dades del codi i les seues possibilitats de paral·lelització.
c) (1) Proposa una paral·lelització d'este codi utilitzant OpenMP. Escriu el codi paral·lelitzat complet i justifica l'elecció.

CASTELLANO

Utilizando el modelo del problema visto anteriormente, resuelve el siguiente ejercicio de examen:

Problema: Se pretende paralelizar el siguiente código en C utilizando OpenMP:

```
float dist(int n,float *d1,float *d2) {
  float sum=0.0;
  float x1,x2;
  for(int i=0;i<n;i++) {
    x1=d1[i];
    for(int j=i+1;j<n;j++) {
      x2=d2[j];
      sum+=pow((x1-x2),2);
    }
  }
  return sqrt(sum)/(n*n);
}
```

a) (0.5) Obtén el coste en FLOP de este algoritmo mostrando todos los cálculos y el coste asintótico. Supondremos que las operaciones tienen un coste de 1 FLOP, las funciones PW y SQRT un coste de 3 FLOP, y no tendremos en cuenta el sobrecoste de las instrucciones "for".
b) (0.5) Indica las dependencias de datos del código y sus posibilidades de paralelización.
c) (1) Propón una paralelización de este código utilizando OpenMP. Escribe el código paralelizado completo y justifica la elección realizada.