Computación de Altas Prestaciones (propuesta examen)

I. Un programa de cálculo numérico se ha instrumentalizado mediante la herramienta *gprof* y el resultado obtenido después de ejecutarlo ha sido el siguiente:

Flat profile:

Each sample counts as 0.04 seconds

% time	acumulative seconds	self seconds	calls	self ms/call	total ms/call	name
54,19	170,52	170,52	12	14210	14210	suma
30,79	267,42	96,9	45	2153,44	2153,44	multi
13,87	311,08	43,66	1	43655	43655	coseno
1,15	314,69	3,61	66	54,71	54,71	resto

Se pide:

1. ¿Qué procedimientos elegiría para mejorar el rendimiento? Justifique su respuesta.
2. ¿Qué sería más beneficioso a priori suponiendo aceleraciones ideales los casos propuestos?

   a) Usar las instrucciones SSE para optimizar la rutina *suma* si se utilizasen datos flotantes sencillos

   b) Usar las instrucciones vectoriales AVX para optimizar la rutina *multi* si se utilizasen datos en punto flotante sencillos

   c) Paralelizar las rutinas *suma* y *multi* con OpenMP en un multicore de 2 cores

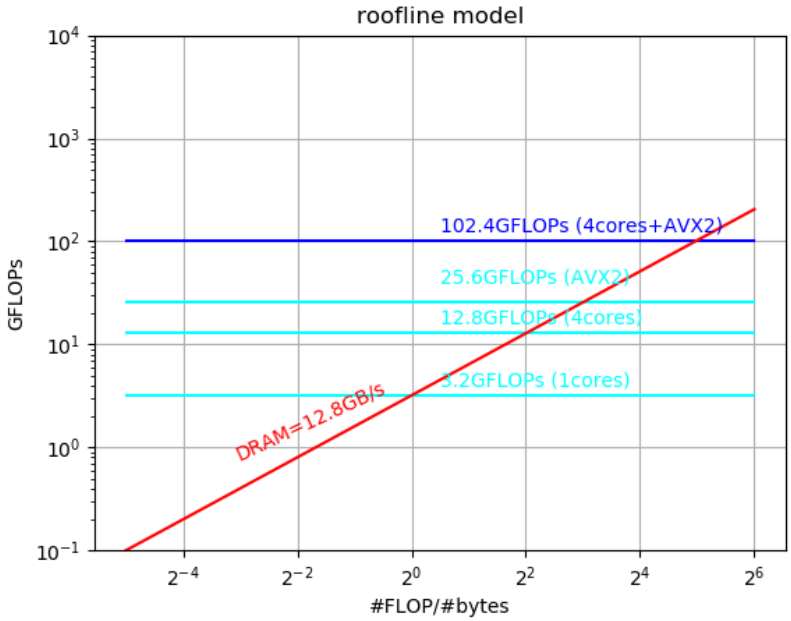
II. De acuerdo a los siguientes códigos y suponiendo que los *arrays* se encuentran almacenados en la memoria principal y la variable ***s*** puede almacenarse en un registro, responder a las siguientes cuestiones

1A	1B	1C	1D

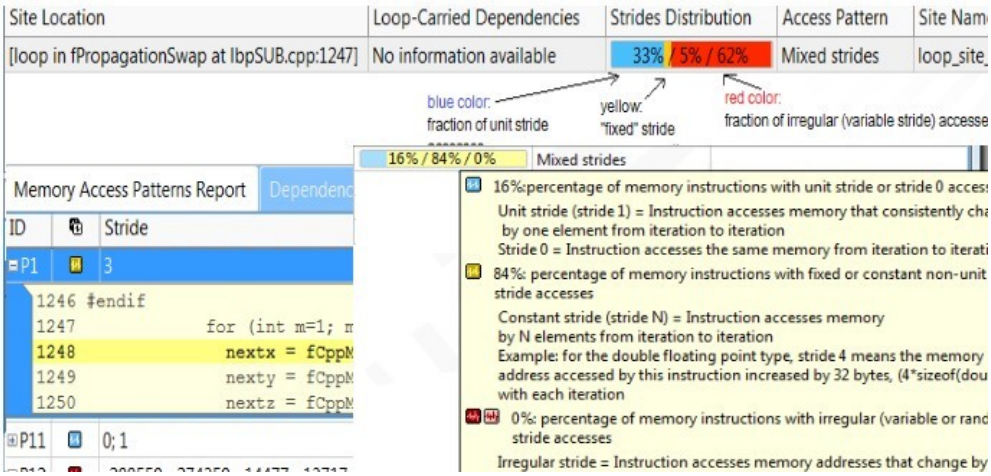
`double a[], b[];`
`for(i=0; i<N; ++i)`
`  a[i] = a[i] + b[i];`
	`double a[], b[];`
	`for(i=0; i<N; ++i)`
	`  a[i] = a[i]+ s * b[i];`
		`float s=0, a[];`
		`for(i=0; i<N; ++i)`
		`  s = s + a[i] * a[i];`
			`float s=0, a[], b[];`
			`for(i=0; i<N; ++i)`
			`  s = s + a[i] * b[i];`

1. ¿Cual es la intensidad aritmética de cada uno de los códigos? ¿Alguno de los códigos es *memory-bound* o *compute-bound*?

2. De acuerdo al modelo *roofline* de la máquina disponible dibujar el rendimiento máximo esperable para los códigos 1A, 1B, 1C y 1D con un único *core* y sin explotar las capacidades SIMD.



III. Evaluando un código desarrollado observamos que no escala como esperamos cuando se hace uso del paralelismo SIMD. Por este motivo, se evalúa su rendimiento mediante la herramienta del Intel Advisor, y poniendo el foco en los patrones de acceso a memoria se observa el siguiente resultados para la rutina *fPropagationSwap*.



El código de colores de la barra *Strides Distribution*, corresponde al acceso con *stride=1* (color azul), *stride* fijo (amarillo) y *stride* variable (barra roja).

De acuerdo a este análisis, explicar razonadamente porque el rendimiento de la aplicación no alcanza la aceleración esperada motivada por los patrones de acceso a memoria. Indicar el motivo del *overhead* o sobrecoste producido de acuerdo a la medida observada en el Intel Advisor.

IV. Añadir la líneas de código correspondientes a las directivas para desarrollar una implementación de OpenMP (target) de la multiplicación de matrices

V. Se desea implementar el producto escalar en un sistema multiprocesador mediante el paradigma de programación MPI. Siendo el código secuencial el que se muestra a continuación:

```
double dotProduct(double *x, double *y, int n) {
  int i;
  double prod = 0.0;
  for (i = 0; i < n; i++) {
    prod += x[i]*y[i];
  }
  return prod;
}


int main(int argc, char *argv[]) {
  double x[N];
  double y[N];
  int i;
  for(i = 0; i < N; i++) {
    x[i] = 0.01 * i;
    y[i] = 0.03 * i;
  }

  double prod;
  prod = dotProduct(x,y,N);
  printf("dotProduct = %f\n", prod);

  return 0;
}
```

VI. La versión secuencial del código siguiente calcula la multiplicación de matrices de A y B. Se desea realizar una implementación paralela con el paradigma de programación paralelo OpenMP

```
for (i=0; i<Ndim; i++){
  for (j=0; j<Mdim; j++){
    tmp = 0.0;
    for(k=0;k<Pdim;k++){
      /* C(i,j) = sum(over k) A(i,k) * B(k,j) */
      tmp += *(A+(i*Ndim+k)) * *(B+(k*Pdim+j));
    }
    *(C+(i*Ndim+j)) = tmp;
  }
}
```