1. Resumen.

Los sistemas computacionales actuales están formados por nodos que comprenden CPUs multicore junto con uno o varios coprocesadores (normalmente GPUs pero ocasionalmente también MICs), de forma que se dispone de un sistema híbrido y heterogéneo.

Por otro lado, el desarrollo de metaheurísticas híbridas y de hiperheurísticas que trabajan sobre ellas también sigue una estructura híbrida, en la que se puede explotar paralelismo multinivel para llevar a cabo computaciones con distinto volumen de cómputo (heterogéneas). En este trabajo analizamos la combinación de paralelismo híbrido y heterogéneo a los dos niveles, de software y de hardware, en la aplicación de metaheurísticas híbridas. Se realizan experimentos con la aplicación de metaheurísticas a dos problemas, uno de docking de moléculas y otro de modelos.

***Artículo publicado por XIII Congreso Español en Metaheurísticas y Algoritmos Evolutivos y Bioinspirados autores: J.M. Cecilia, J. Cuenca y J. M. Cutillas-Lozano “Explotación de Paralelismo Multinivel e Híbrido en Metaheurísticas Híbridas” 2018.***

1. Introducción.

Las metaheurísticas se utilizan para la aproximación de soluciones de problemas de gran diﬁcultad computacional. Hay variedad de métodos metaheurísticos, básicamente agrupados en dos clases: distribuidos o basados en poblaciones, y de búsqueda local, y también métodos híbridos, que combinan las características de diferentes metaheurísticas para una mejor adaptación al problema con el que se trabaja.

Así, el software es híbrido al combinar distintas técnicas, y heterogéneo en el sentido de que cada uno de sus componentes tiene un coste computacional distinto. Con la aparición de la computación paralela se han desarrollado versiones paralelas de las metaheurísticas, o nuevas metaheurísticas paralelas. Los nodos de la mayoría de los sistemas computacionales actuales están formados por sistemas multinúcleo junto con coprocesadores, que son principalmente GPUs (Graphics Processing Unit)) y MICs (Many Integrated Core), por lo que se está trabajando en la adaptación y optimización de software para este tipo de sistemas híbridos, y, en particular, hay trabajos sobre metaheurísticas para GPU [9], [10]. Además, el hardware sobre el que se trabaja es híbrido y heterogéneo, con elementos computacionales con arquitecturas distintas, en distinto número y distinta velocidad, y está organizado en muchos casos de forma jerárquica (un cluster con nodos híbridos multicore+GPU, GPUs organizadas en grids y bloques...).

Esta situación propicia la línea de trabajo que se presenta: análisis de la combinación de las características híbridas, heterogéneas y jerárquicas de software y hardware para explotar el paralelismo y obtener beneﬁcios en cuanto a reducir el tiempo de ejecución o mejorar las soluciones obtenidas. Trabajamos con metaheurísticas híbridas desarrolladas a partir de un esquema parametrizado , que combina características de metaheurísticas distribuidas y de búsqueda local, y se ha aplicado a diversidad de problemas: p-hub, asignación de tareas a procesos, y modelado de ecuaciones simultáneas; consumo de electricidad en explotación de pozos de agua; determinación de constantes en ecuaciones cinéticas, etc. Además, al estar el esquema parametrizado, se puede utilizar para diseñar hiperheurísticas que buscan en el espacio de los parámetros que determinan las metaheurísticas.

El esquema parametrizado se puede ampliar a esquemas paralelos que incluyen parámetros de paralelismo dependiendo del sistema computacional destino (memoria compartida o paso de mensajes), con una paralelización uniﬁcada para las distintas metaheurísticas híbridas del esquema. Analizamos la adaptación del esquema metaheurístico parametrizado a sistemas más complejos: nodos multicore+multiGPU, posiblemente con GPUs de distintas características. Y los problemas a los que se aplican las metaheurísticas tienen una característica que facilita el uso eﬁciente de la alta capacidad computacional de los coprocesadores: alto coste de cálculo del ﬁtness, que puede delegarse a las GPUs. Se utilizan dos problemas: docking de moléculas y modelado de autoregresión vectorial; y un tercer problema es la búsqueda por medio de hiperheurísticas (programadas como metaheurísticas) de metaheurísticas apropiadas para estos dos problemas básicos.

El mayor coste computacional reside en el cálculo del ﬁtness, y en las hiperheurísticas el cálculo del ﬁtness se hace a su vez con la aplicación de metaheurísticas al problema básico. El resto del trabajo está estructurado de la siguiente manera. La Sección II muestra el esquema metaheurístico parametrizado que se utiliza, y se comentan las características de las hiperheurísticas sobre este esquema.

***Artículo publicado por XIII Congreso Español en Metaheurísticas y Algoritmos Evolutivos y Bioinspirados autores: J.M. Cecilia, J. Cuenca y J. M. Cutillas-Lozano “Explotación de Paralelismo Multinivel e Híbrido en Metaheurísticas Híbridas” 2018.***

1. Cribado virtual.

La principal idea que subyace en este método de cribado virtual es la de una técnica de detección sobre la superficie de proteínas, ejecutada en paralelo sobre GPU. Esencialmente, los métodos de CV procesan una gran base de datos de moléculas con el fin de encontrar cuál encaja mejor con algunos criterios establecidos. En el caso del descubrimiento de nuevos fármacos potenciales, optimización de compuestos, valoración de la toxicidad y las etapas adicionales del proceso de descubrimiento de fármacos, examinamos una gran base de datos de compuestos para encontrar una pequeña molécula, que interactúa de una manera deseada con uno o varios receptores. Entre los muchos métodos de CV disponibles para este propósito hemos decidido utilizar el acoplamiento proteína ligando. Estos métodos tratan de obtener predicciones rápidas y exactas de la conformación 3D que adopta un ligando cuando interacciona con una determinada proteína objetivo, y también la fuerza de esta unión, en términos del valor de su función de afinidad. Normalmente, las simulaciones de acoplamientos se llevan a cabo en una parte muy concreta de la superficie de la proteína en métodos como Autodock, Glide y Dock, por nombrar algunos. Esta región se deriva comúnmente desde la posición de un ligando particular en la estructura cristalina, o de la estructura cristalina de la proteína sin ligando. El primero se puede realizar cuando la proteína se ha cocristalizado con el ligando, pero puede ocurrir que no haya una estructura cristalina del par ligando proteína que se encuentre disponible.

Sin embargo, el principal problema es suponer, una vez que se especifica el sitio de unión, que muchos ligandos diferentes van a interactuar con la proteína en la misma región, descartando completamente las otras áreas de la proteína. Ante este problema, se propone la división de toda la superficie de la proteína en regiones definidas. Posteriormente, se llevan a cabo simulaciones de acoplamiento para cada ligando en todos los puntos especificados de la proteína simultáneamente. Siguiendo este enfoque, los nuevos puntos de acceso se pueden localizar tras un examen de la distribución de la función de afinidad en toda la superficie de la proteína. Esta información podría conducir al descubrimiento de nuevos lugares de unión. Si comparamos este enfoque con una simulación de acoplamiento típico, realizado solo en una región de la superficie , el principal inconveniente de este enfoque radica en el aumento de su coste computacional. Decidimos continuar en esta dirección y mostrar cómo esta limitación puede solucionarse gracias al hardware de las GPU y los nuevos diseños algorítmicos.

En esencia, en una simulación de acoplamientos se calcula la energía de interacción ligando-proteína para una configuración de partida dada del sistema, que está representado por una función de afinidad. En BINDSURF se calcula la función de puntuación, carga electrostática, VanderWaals y los términos de los átomos de hidrógeno(hbond). Por otra parte, en los métodos de acoplamiento normalmente se supone que los mínimos de la función de afinidad, entre todas las conformaciones ligando-proteína, representan con exactitud la conformación que adopta el sistema cuando el ligando se une a la proteína. Por lo tanto, cuando se inicia la simulación, tratamos de minimizar el valor de la función de afinidad mediante la introducción continua de perturbaciones aleatorias o predefinidas en el sistema, calculando para cada paso el nuevo valor de la función de afinidad, así como aceptando o no los diferentes enfoques siguiendo el método de reducción de MonteCarlo u otros. Las simulaciones se realizan siempre con un total de 500 pasos de MonteCarlo.

***Artículo publicado por ESEVIER autores: J.M. Cecilia, G. Cano, G. García Rodríguez y H. Pérez Sánchez “Descubrimiento de fármacos basado en cribado virtual con enfoques neuronales paralelos” 2015.***

4. Paralelización del término electrostático.

a. Descripción del código secuencial.

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <math.h>

#include "energy\_common.h"

#include "definitions.h"

#include "solver.h"

#include "energy\_struct.h"

using namespace std;

int main (int argc, char \*\*argv){

// c recoge argumento introducido por teclado

char c;

// mode recoge si vamos a trabajar en modo secuencial, o paralelo.

int mode;

char \*file\_l, \*file\_p, \*file\_e, \*file\_a;

// nconformations variable que indicará el número de conformaciones.

int nconformations;

// en energy\_desolv\_CPU vamos a guardar el valor de cada una de las conformaciones.

float \*energy\_desolv\_CPU;

float \*conformations\_x, \*conformations\_y, \*conformations\_z;

struct ligand\_t ligando;

struct receptor\_t proteina;

// system ("clear");

file\_l = (char \*)malloc(sizeof(char)\*100);

file\_p = (char \*)malloc(sizeof(char)\*100);

file\_e = (char \*)malloc(sizeof(char)\*100);

file\_a = (char \*)malloc(sizeof(char)\*100);

nconformations = 1;

// El while va cogiendo los parámetros de la línea de comandos.

while ((c = getopt (argc, argv, "vr:m:l:a:r:x:g:y:z:s:c:e:n:h")) != -1) {

switch (c) {

case 'v':

printf("Energy atomic Ligand-Protein Calculation v.1.0\n\n");

return 0;

case 'm':

mode = atoi(optarg);

break;

\\ Indicamos con “–r” el fichero de la proteína.

case 'r':

strcpy(file\_p,optarg);

break;

\\ Indicamos con “-l” dónde está guardado el ligando.

case 'l':

strcpy(file\_l,optarg);

break;

\\ le pasamos a la varialbe mediante argumentos el numero de conformaciones

case 'c':

nconformations = atoi(optarg);

break;

case 'h':

case '?':

\\ idicamos a modo de leyenda cómo introducir los parámetros.

printf("Usage:\tenergy -r fichero\_protein.mol2 -l fichero\_ligand.mol2 [-c numero\_conformaciones] [-h | -? HELP] \n");

printf("\t<Params>\n");

printf("\t\t-v\t\tOutput version information and exit\n");

return 0;

}

}

\\ llamada a función archivos de proteína y ligando según los datos previamente almacenados.

readLigand (file\_l,&ligando);

readProtein (file\_p, proteina);

//CREATE CONFORMATIONS

\\ Reservamos memoria para las conformaciones que hemos pasado como parámetros.

conformations\_x = (float \*)malloc(sizeof(float)\*nconformations\*ligando.nlig);

conformations\_y = (float \*)malloc(sizeof(float)\*nconformations\*ligando.nlig);

conformations\_z = (float \*)malloc(sizeof(float)\*nconformations\*ligando.nlig);

//FILL CONFORMATIONS.

fill\_conformations(nconformations,conformations\_x,conformations\_y,conformations\_z,ligando);

//ENERGY FOR EACH ATOM

\\ Nueva reserva de memoria

energy\_desolv\_CPU = (float \*)calloc(sizeof(float),nconformations);

\\ Llamada al procedimiento solver

solver\_AU(mode, proteina.atoms, ligando.atoms, ligando.nlig,proteina.rec\_x, proteina.rec\_y, proteina.rec\_z, conformations\_x, conformations\_y,conformations\_z,ligando.ql,proteina.qr,energy\_desolv\_CPU,nconformations);

return 0;

}

\\ Fichero energy-struct.h: incluye las estructuras, struct ligand\_t ligando y struct receptor\_t proteina

b. Paralelización en procesadores multinúcleo.

Las moléculas y proteínas reales están en la carpeta input. Se trata de paralelizar el cálculo electrostático ya implementado.

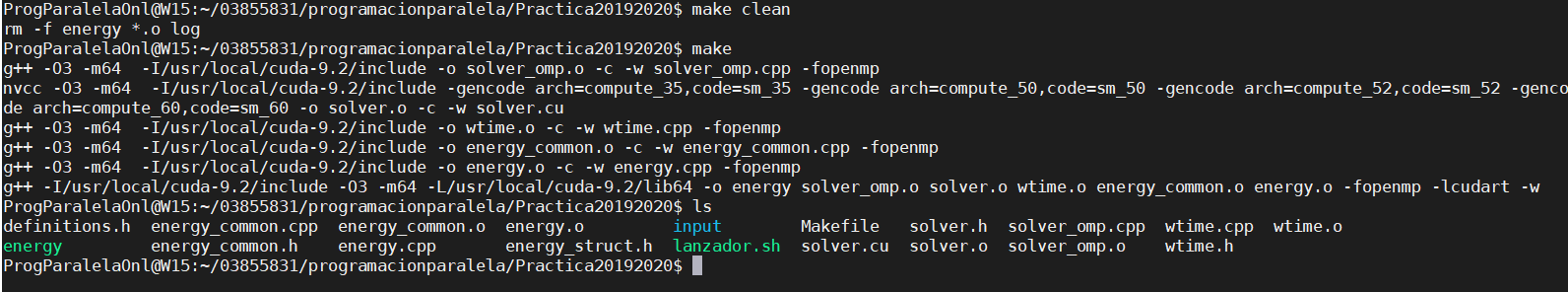
1. Compilación. (make clean, make)

- Para evitar hacer las compilaciones una a una usamos el fichero de reglas make y genera el ejecutable energy.

- Lanzador.sh: ejecuta el programa en CPU, GPU, con todos los ficheros que hay en input.

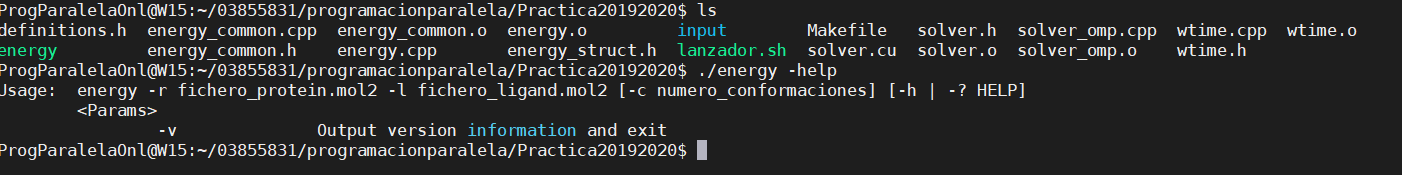
- En solver.q , es el punto de entrada al código en función de los parámetros usará cod. Secuencial (0), openmp (1) o cuda (2).

- Para paralelizar en openmp : Tenemos la función forces\_OMP\_AU que se encuentra en el fichero solver\_omp.ccp (openomp y cuda no pueden estar en el mismo fichero. (tenemos que paralelizar este código).

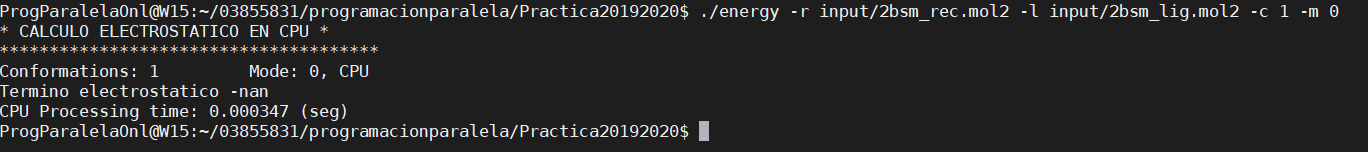


Una vez compilado comprobamos con el parámetro –help los argumentos requeridos para ejecutar el programa en sus diferentes

modalidades de paralelización:



Los parámetros según serían –r para indicar dónde está el fichero receptor, -l indica dónde está el ligando –c número de conformaciones –m la modalidad, siendo 0 en secuencial.

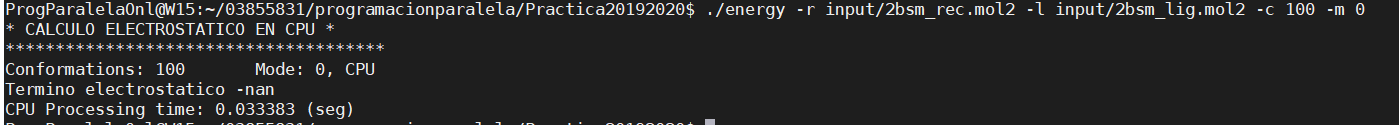


Vamos a hacer el análisis en función de diferentes conformaciones en las diferentes modalidades.

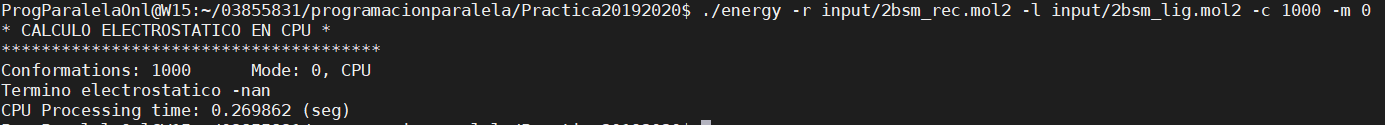
1. ANÁLISIS SECUENCIAL:

Pruebas en función del nº de conformaciones:

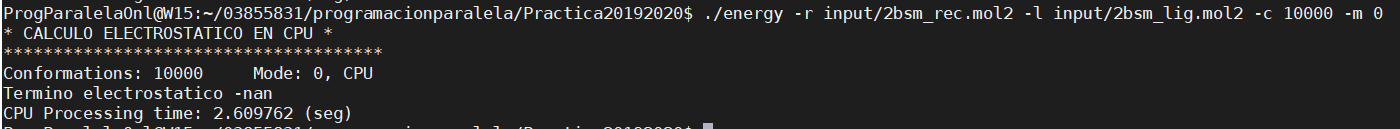
100 CONFORMACIONES:



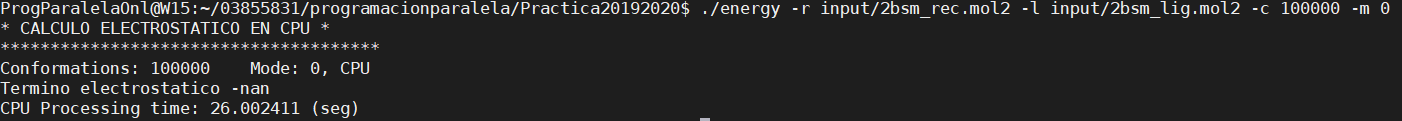
1000 CONFORMACIONES:



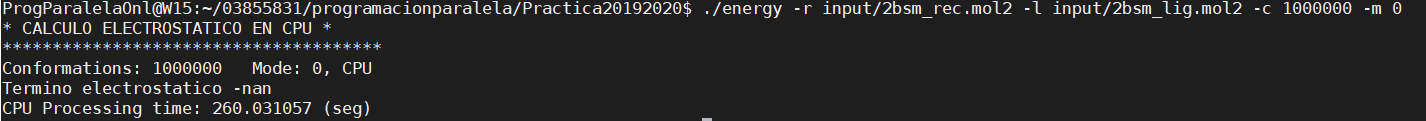
10000 CONFORMACIONES:



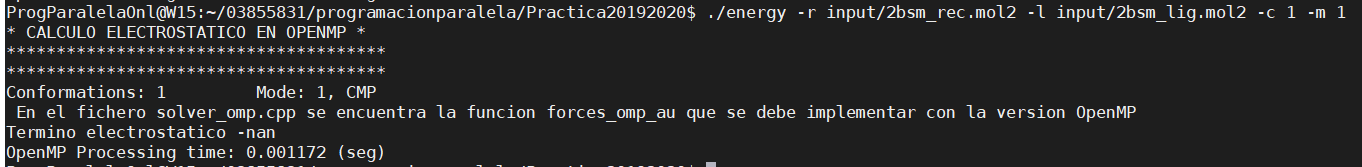
100000 CONFORMACIONES:



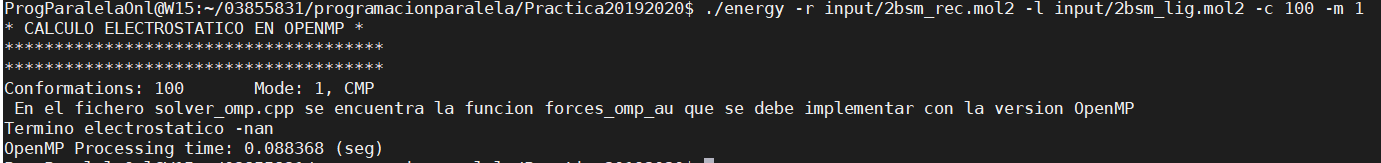
1000000 CONFORMACIONES:



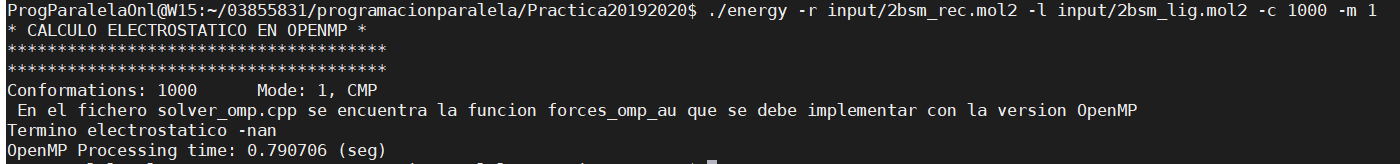
OPENOMP:



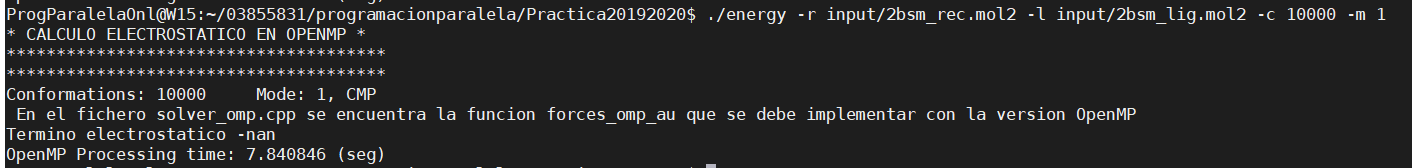
100 CONFORMACIONES:



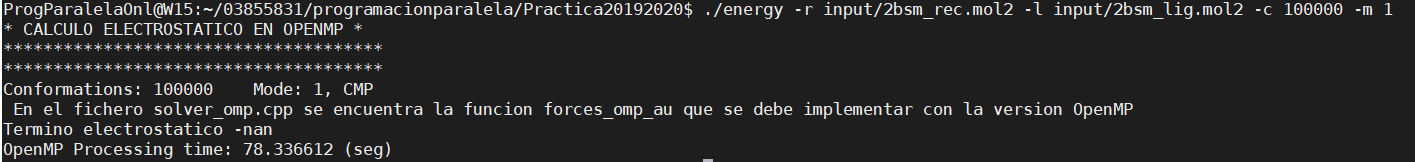
1000 CONFORMACIONES:



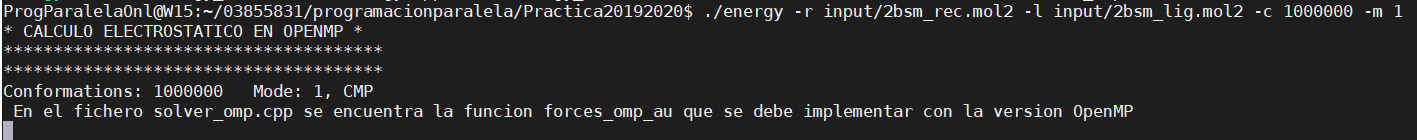
10000 CONFORMACIONES:



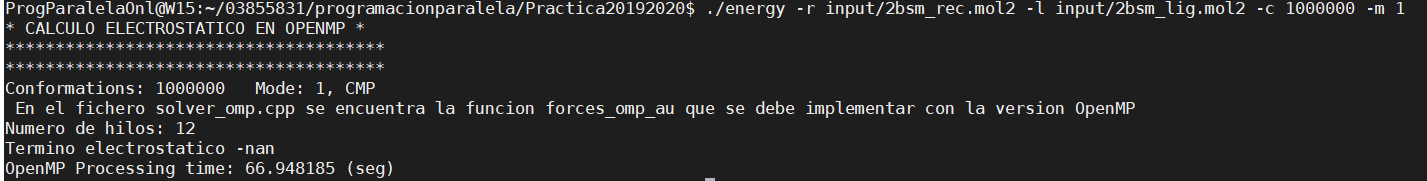
100000 CONFORMACIONES:



1000000 CONFORMACIONES (Ni termina)

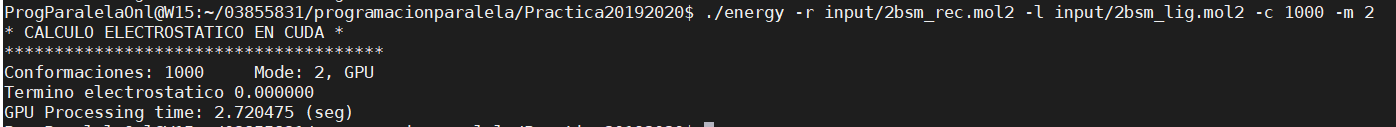


1000000 CONFORMACIONES CON MODIFICACIONES Con modificaciones:

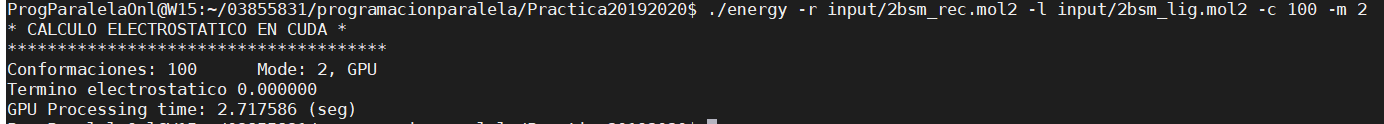


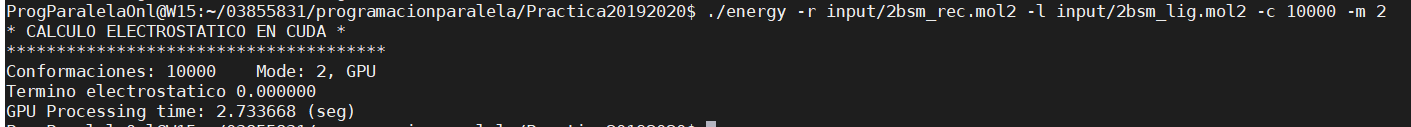
CUDA:

1000 CONFORMACIONES

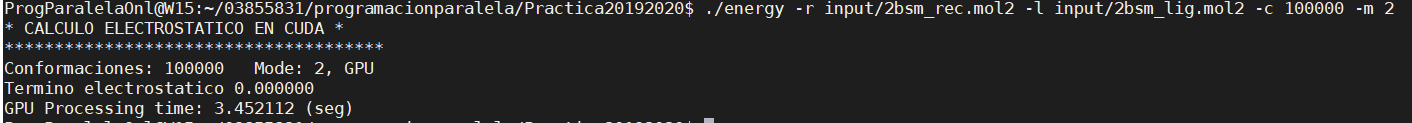


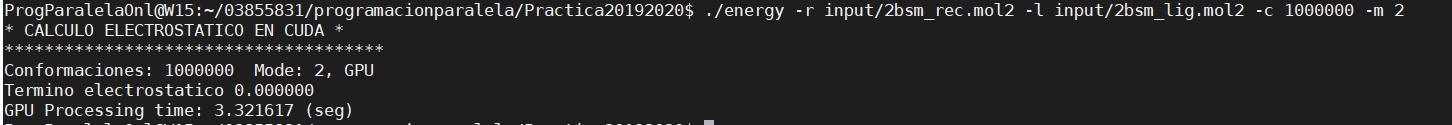
100 CONFORMACIONES



10000 CONFORMACIONES

100000 CONFORMACIONES:



1000000 CONFORMACIONES:

5. Resultados experimentales.

a. Entorno de simulación.

El sistema utilizado para las pruebas tiene las siguientes características:

Intel® Xeon® CPU E52603 V4 @ 1,70Ghz

32 gb RAM

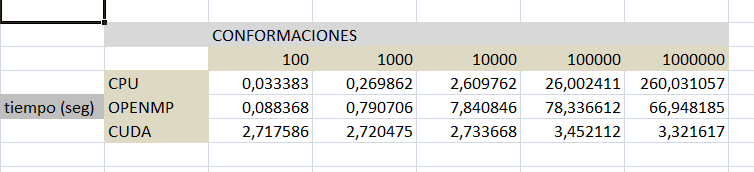
Nvidia Corporation GK110b gl k40

Nvidia Corporation GK110b gl k20

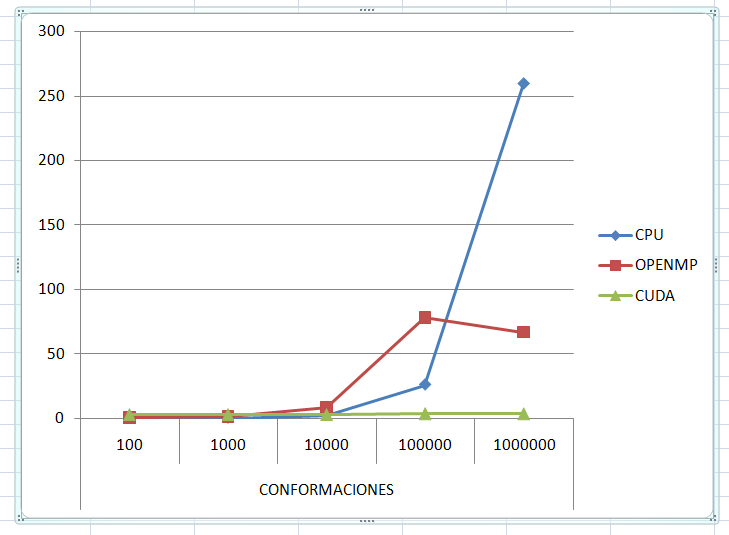
2 procesadores 12 núcleos.

b. Resultados.

Tabulación de datos obtenidos:



Grafica de rendimientos



6. Conclusiones.

Tras los experimentos realizados en CPU, OpenMP y Cuda, éstas son las conclusiones:

- Hasta 10.000 conformaciones los rendimientos son similares en torno a : . Esto es por que la carga computacional es relaivamente baja y no aprovechamos la paralelización, entre 10000 y 100000 comformaciones los tiempos en CPU se disparan y el OpenMP gestiona los hilos mejor que cuda, no hay que olvidar que los procesadores de la tarjeta gráfica son multiples pero simples esto supone que necedsitamos un paralesilmo de datos sobre el mismo proceso muy alto para obrener resultados, esto se confirma cuando experimentamos con 1.000.000, los resultados son, claramente mejores en Cuda que en las otras dos modalidades.

7. Bibliografía

***J.M. Cecilia, J. Cuenca y J. M. Cutillas-Lozano “Explotación de Paralelismo Multinivel e Híbrido en Metaheurísticas Híbridas” 2018.***

***J.M. Cecilia, G. Cano, G. García Rodríguez y H. Pérez Sánchez “Descubrimiento de fármacos basado en cribado virtual con enfoques neuronales paralelos” 2015.***