CS3P01 Análisis de escalabilidad Evolución Galáctica

Fabrizio Vásquez
Departamento de Computer Science
Universidad de Ingeniería y Tecnología
Lima, Perú
fabrizio.vasquez@utec.edu.pe

Johan Tanta
Departamento de Computer Science
Universidad de Ingeniería y Tecnología
Lima, Perú
johan.tanta@utec.edu.pe

I. INTRODUCCION

Las simulaciones de colisiones galácticas son importantes debido a que permiten predecir de forma teórica el comportamiento de dos galaxias en cercanía y determinar las probabilidades de colisiones entre sí a futuro. El modelado y simulaciones de galaxias y cuerpos celestes presenta retos numéricos y computacionales, esto es causado a la gran cantidad de cuerpos que se tiene que usar y las diferentes interacciones que estos manejan. Una de las prioridades de las simulaciones se aproximen lo mejor posible a un escenario realista y que el tiempo de ejecución no sea tan alto.

II. OBJETIVOS

En el presente proyecto se plantearon los siguientes objetivos:

- Diseño del PRAM del problema de N-cuerpos.
- Ejecutar pruebas con diferentes cantidad de cuerpos y procesadores.
- Medir el tiempo de ejecución, cálculo de la fuerza, tiempo de comunicación entre procesos y los GFlops.
- Realizar comparaciones de los resultados obtenidos del *speedup* y eficiencia.
- Determinar la escalabilidad del algoritmo.

III. MARCO TEÓRICO

En el desarrollo e implementación del software se utilizaron dos conceptos para determinar cuan escalable son los códigos, estos son: Escalamiento fuerte, escalamiento débil , eficiencia de escalamiento fuerte, eficiencia de escalamiento débil, curva de isoeficiencia.

A. ESCALAMIENTO FUERTE (Strong Scaling)

En el caso de los resultados provistos, el tamaño del problema permanece fijo pero aumenta el número de elementos de procesamiento. Esto se usa como justificación para los programas que tardan mucho en ejecutarse. En una escala fuerte, se considera que un programa escala linealmente si el speed-up es igual al número de elementos de procesamiento utilizados [3].

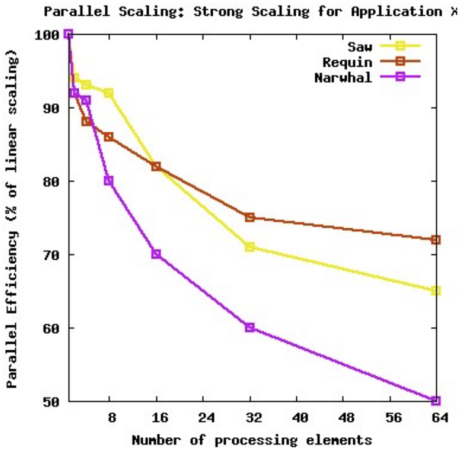

Fig. 1. Escalado fuerte: escalamiento fuerte para la aplicación x[4].

B. EFICIENCIA DE ESCALAMIENTO FUERTE

Ruby Lee en 1980 definió varios parámetros para evaluar el cálculo paralelo. A continuación se muestra la definición de dichos algunos parámetros.

Sea O(n) el número total de operaciones elementales realizadas por un sistema con n elementos de proceso, y T(n) el tiempo de ejecución en pasos unitarios de tiempo. En general, T(n) ¡ O(n) si los n procesadores realizan más de una operación por unidad de tiempo, donde $n \geq 2$. Supongamos que T(1) = O(1) en un sistema mono-procesador, lo cual supone que IPC = 1 (IPC= Instrucciones Por Ciclo). El factor de mejora del rendimiento (speed-up, aceleración o rapidez) se define como:

$$S_n = \frac{T_1}{T_n} \tag{1}$$

La eficiencia del sistema para un sistema con n procesadores se define como:

$$E_n = \frac{S_n}{n} = \frac{T_1}{p * T_n} \tag{2}$$

IV. METODOLOGÍA

En primer lugar para realizar nuestro cálculos de manera teórica diseñamos el siguiente PRAM al cual se le añadió el

costo de comunicación.

Algorithm 1: PRAM N-cuerpos

Entrada: Todos las partículas con masa,posición,velocidad,aceleración
Salida: Un dataset con el resultado de las interacciones con los diferentes cuerpos.

for $i \leftarrow 1$ **to** N **do**
  $\zeta_i \leftarrow$ read(datos)
end
$broadcast(N)\ AllScatter(...\zeta_{\frac{n}{p}})$
while $i < N$ **do**
  pardo
  $my_x := \zeta.rx$;
  $my_y := \zeta.ry$;
  $my_z := \zeta.rz$;
  while $j < N$ **do**
    if $i \neq j$ **then**
      $d_x := \zeta_{x_j} - my_x$;
      $d_y := \zeta_{y_j} - my_y$;
      $d_z := \zeta_{z_j} - my_z$;
      send(i)—receive(j);
      $a_x := \frac{G*my_i my_j}{d_x^2}$;
      $a_y := \frac{G*my_i my_j}{d_y^2}$;
      $a_z := \frac{G*my_i my_j}{d_z^2}$;
    end
  end
  while $l < N$ **do**
    pardo
    $\zeta.vxi := a_x \triangle ts$;
    $\zeta.vyi := a_y \triangle ts$;
    $\zeta.vzi := a_z \triangle ts$;
    $\zeta.rxi := v_z \triangle ts$;
    $\zeta.ryi := v_z \triangle ts$;
    $\zeta.rzi := v_z \triangle ts$;
  end
  while $k < N$ **do**
    $AllGather(...\zeta_{\frac{n}{p}})$
  end
end

Para el desarrollo del análisis se utilizo el generador N-cuerpos gen-plum como primer componente para obtener información de nuestra utilidad, dicho generador nos brindo un conjunto de datos que variaba según la modificación del parámetro N o cantidad de partículas. Luego utilizamos dicha información para ejecutar el programa cpu-4th que nos brindaba multiples estadísticas en ellas que usaríamos más adelante como: speed,tiempo de comunicación, tiempo total de ejecución, tiempo en el cálculo de la fuerza entre otros. Una vez terminada la ejecución según la cantidad de cuerpos y cantidad de procesadores empleados, derivamos las múltiples salidas a archivos .txt con nombre del numero de partículas empleadas seguida del número de procesadores empleados en esa prueba. Después, se utiliza cada una de estas salidas como insumo para un programa .sh para generar archivos .txt pero esta vez con la información relevante obtenida a través de lenguaje *bash* el cual realiza una consulta al usuario de que información quisiera obtener y capturarla.

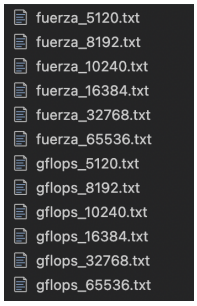

Fig. 2. Documentos retornados luego de ejecutar el script

Luego creamos un archivo .gnu y empleamos gnuplot a partir de la información devuelta por el script para graficar las diferentes variables que desarrollaremos en nuestro análisis. Finalmente se desarrollo un software en lenguaje python para poder obtener las estadísticas y datos relevantes de manera mucho más interactiva.

Como último detalle, utilizamos Khipu, supercomputador de UTEC. Específicamente el procesador Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz.

V. RESULTADOS

Ejecutamos el programa principal modificando el tamaño de los datos, N = 5k, 8k, 10k, 16k, 32k, 64k y del archivo de salida que obtenemos, se extrae los datos que necesitamos.

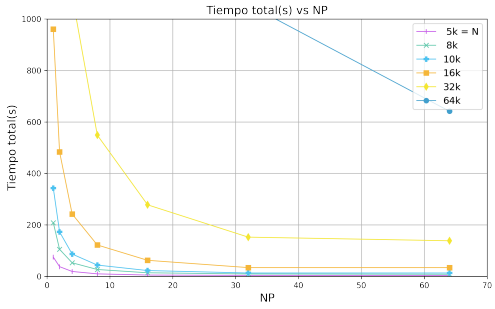

Fig. 3. Tiempo total de ejecución (s) vs NP

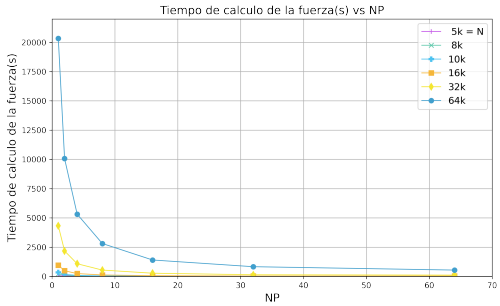

Fig. 4. Tiempo del cálculo de la función calc_force (s) vs NP

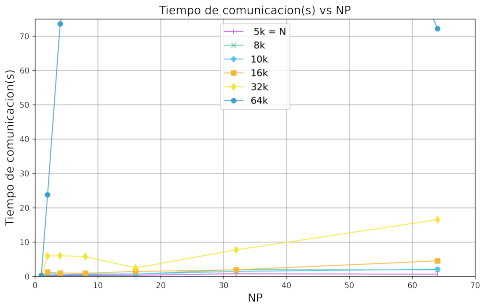

Fig. 5. Tiempo de Comunicación (s) vs NP

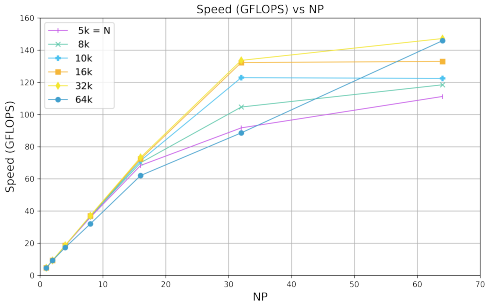

Fig. 6. Speed(GFlops) vs NP

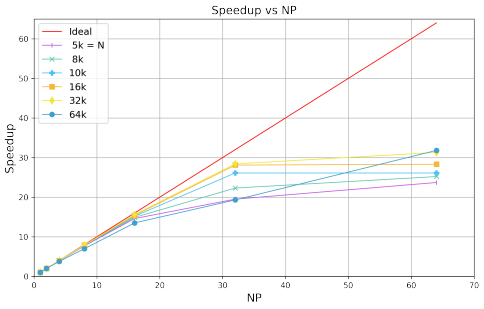

Fig. 7. Speedup vs NP

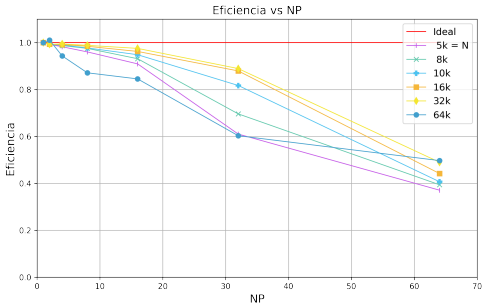

Fig. 8. Eficiencia vs NP

VI. ANÁLISIS DE RESULTADOS

En primer lugar, se analiza el comportamiento que presenta el algoritmo secuencial ($np = 1$), por lo que extraemos los datos del tiempo del cálculo de la fuerza. De manera visual, observamos en la figura 9 una complejidad $O(n^2)$ que concuerda con la teórica. Asimismo, los resultados son respaldados con el $r - squared$ el cual indica que la curva experimental se aproxima a la curva teórica.

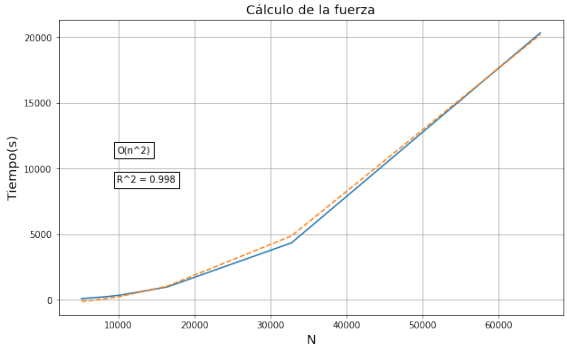

Fig. 9. Complejidad algorítmica

En cuanto, a los tiempos de ejecución que observamos en la figura 3, a medida que se utiliza más procesadores, el problema original se resuelve cada vez más rápido. Este mismo patrón se observa en el tiempo del cálculo de la fuerza, vista en

la figura 4. Sin embargo, cuando observamos la figura 5, donde se presenta el tiempo de comunicación entre procesos, al utilizar mayor cantidad de procesos, se aprecia que el tiempo que se emplea es mucho mayor, lo cual es lo esperado.

Al observa la gráfica del speedup, en la figura 7, observamos que la curva $N = 32k$ presenta un mejor speedup a comparación de los demás. Asimismo, los resultados se acercan a un speedup lineal cuando utilizamos hasta 16 procesos. Cada vez que vamos aumentando la cantidad de procesos, se van alejando de la curva ideal, presentando una convergencia.

Finalmente, cuando observamos la gráfica de eficiencia, en la figura 8, observamos que la curva $N = 32k$ presenta una mejor eficiencia con respecto a los demás. Y, la eficiencia va disminuyendo cada vez que aumentamos la cantidad de procesos, pero, al aumentar el N aumentamos la eficiencia. Es decir, la carga promedio de cada procesador se encuentra más balanceada con respecto a su poder de cómputo.

VII. CONCLUSIÓN

- Según los datos obtenidos, podemos concluir que el algoritmo N-cuerpos optimizado para MPI (Message Passing Interface) que interviene en el cálculo de fuerzas entre partículas nos muestra una gran similitud con lo desarrollado en la parte teórica siguiendo la complejidad de $O(\frac{n^2}{p})$
- Creemos que haber realizado una mayor cantidad de experimentos con una mayor cantidad de partículas seguiría respaldado hallado.
- En un caso específico, cuando la cantidad de partículas es $N = 32K$ encontramos que su eficiencia es mayor al resto de pruebas con un N diferente.
- El tiempo de cálculo de la fuerza nos muestra como disminuye el tiempo de ejecución en relación con el aumento en la cantidad de procesos, ya que este representa el mayor tiempo de cálculo junto con el tiempo de comunicación. Por lo antes expuesto podemos concluir que el algoritmo N-cuerpos paralelizado con MPI que utiliza el proyecto Evolución galáctica es escalable.

VIII. CÓDIGO FUENTE

Proyecto Evolución Galáctica, disponible en GitHub. (https://github.com/FabrizioVasquez/CPD_Proyecto_Evolucion_Galactica)

REFERENCES

[1] Kai Hwang. *Advanced computer architecture*: Parallelism, scalability,programmability. McGraw-Hill, 1993.

[2] John Gurland and Ram C. Tripathi (1971), *A Simple Approximation for Unbiased Estimation of the Standard Deviation*: The American Statistician 25 (4): 30-32 .

[3] Measuring Parallel Scaling Performance(2016)[Internet].Disponible en https://www.sharcnet.ca/help/index.php/MeasuringParallelScalingPerformanceStrongScaling .

[4] Introduction to HPC: Scaling tests and profiling for efficient HPC[Internet].Disponible en https://pdc-support.github.io/hpc-intro/10-scaling/ .