Paralelización de procesos con Multiprocessing

En este trabajo se presentan una forma de paralelizar la simulación de Ising dos dimensional desarrollada en [mitrabajo], para ello se hace uso de la clase Pool del módulo de Python multiprocessing para la ejecución en paralelo del algoritmo de metropolisHasting. Acá se muestra una grafica comparativa de los tiempos de ejecución en serial y en paralelo del programa en función del tamaño del sistema de partículas y muestra que con esta simple optimización del código se logra acelerar el proceso en un factor de 3 aproximadamente usando los cuatro núcleos de procesamiento.

Introducción

La computación paralela es una técnica fundamental en el ámbito de investigación científica, especialmente en el campo de la simulación, donde se llevan a cabo cálculos y operaciones complejas que requieren de una gran capacidad de procesamiento. Aunque los conceptos de la computación en paralelo pueden ser bastante amplios y diversos, en el ámbito de la simulación científica es común enfocarse en la paralelización a nivel de tareas, en la cual la arquitectura de Von neumann se mantiene, pero se explota utilizando múltiples unidades de procesamiento que ejecuten las diferentes tareas que componen un proceso completo. [https://www.teldat.com/blog/es/computacion-paralela-capacidad-procesamiento/]

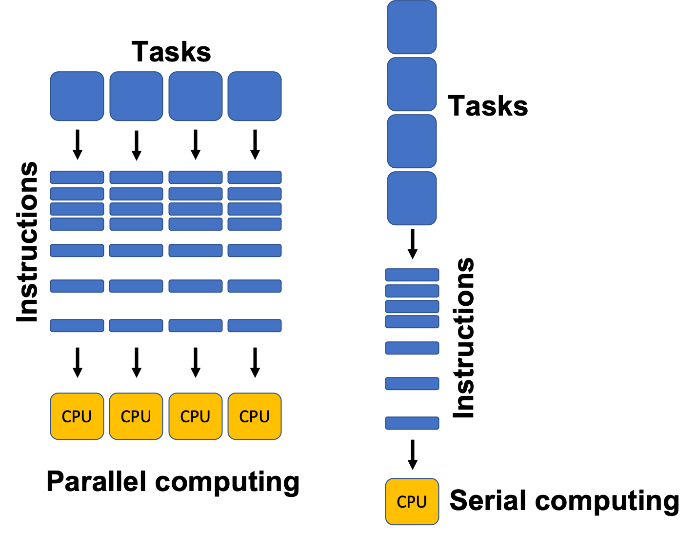
De forma particular, en este trabajo, que corresponde al segundo proyecto del curso Computación Avanazada I, se mostrará el poder de la computación en paralelo con una simulación de un modelo de Ising dos dimensional, que se caracteriza por requerir de tiempos de cómputo mayores cuanto más grande es el sistema. Para esto, se medirán los tiempos de ejecución para mayas cuadradas de diferentes tamaños y se analizará la mejora de rendimiento con respecto al desarrollo secuencial típico de la simulación. Se hará uso de la librería de Python multiprocessing, que representa una forma muy simple y poderosa de tener un primer acercamiento a la paralización de tareas por medio de la creación de procesos que pueden desarrollarse de forma simultánea.

En [mi trabajo] se pudo presentar simulaciones para mayas de dimisiones 10x10 y 30x30, omitiendo sistemas más grandes por la gran cantidad de tiempo necesaria para correr cada una, sin embargo, en este trabajo se pretende usar la mejora de rendimiento alcanzada con la paralelización del programa para poder analizas mayas con tamaños más significativos.

Marco teórico

Como ya se planteó en el primer proyecto de este curso [], el modelo de Ising es propuesto para estudiar el comportamiento de materiales ferromagnéticos, y representa un problema paradigmático para la física estadística en el sentido de que permite ilustrar los fenómenos de transición de fase de sistemas macroscópicos.

En [mi trabajo] además de plantear las bases conceptuales importantes sobre el modelo de Ising dos dimensional, se mostró, cómo a partir de la simulación de mayas cuadradas de espines evolucionadas bajo un algoritmo de Metrópolis-Hastings, se podía observar una transición de fase calculando el parámetro de orden M (magnetización del sistema) en función del parámetro $\beta = \frac{1}{k\_B T}$ por medio de un estimado de Montecarlo. Pero como se mencionó antes, los tiempos de computo deben mejorarse si se quieren analizar sistema con más partículas, por lo que, a continuación, se da una breve idea de en lo que consiste la paralización a nivel de tareas para aplicarlo a la simulación del modelo de Ising.

Paralelización de tareas:

Para resolver un problema de forma computacional, típicamente, se plantean una serie de tareas o procesos que deben ser desarrollados. Cada una de estas tareas está compuesta de un conjunto de instrucciones que son entregadas a las unidades de procesamiento en el orden en el que el programador lo estableció, esta situación se ilustra en la parte derecha de la figura (). La solución al problema se obtiene al finalizar la ejecución de cada tarea y para esto se invierte cierta cantidad de tiempo de computo que se denotará como Ts. Ahora bien, los computadores modernos, por lo general y por fortuna, cuentan con más de una unidad de procesamiento, lo cual brinda al programador la posibilidad de repartir la carga de trabajo del problema, y, por ende, estará de acuerdo con esta afirmación, la posibilidad de reducir el tiempo de ejecución (Tp < Ts). Cuando las tareas de un problema se ejecutan por unidades de procesamiento distintas, se puede decir entonces que se está desarrollando un proceso de computación en paralelo.

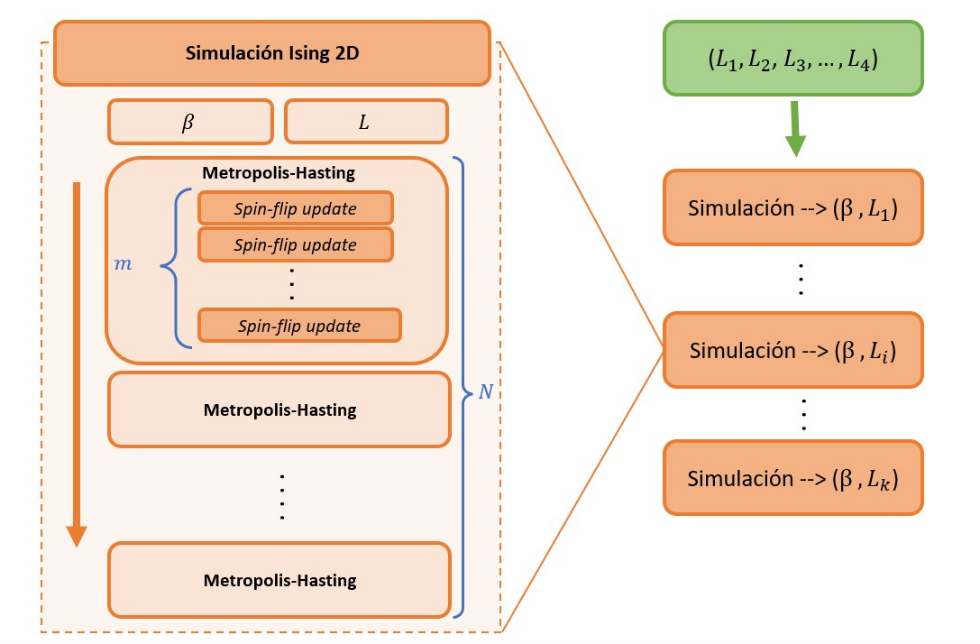
Idealmente, la ejecución sincrónica, como se ve la figura izquierda, es la que permite el tiempo más optimo de procesamiento y es el objetivo general cuando se decide paralelizar un problema, sin embargo, no todas las tareas de un programa se pueden paralelizar, y esto dependerá de la correlación o dependencia entre los procesos, al final, solo cierto porcentaje del programa podrá ser paralelizado, pero será suficiente para mejorar en gran medida el rendimiento de un programa, simulación o analisis científico.

Multiprocessing

La forma de implementar tareas en paralelo en un programa está lejos de ser única y depende en gran medida del lenguaje que se quiera usar. Una primera aproximación a este paradigma de la programación puede hacerse en Python a través del módulo multiprocessing, una librería bastante robusta que permite aprovechar al máximo los múltiples procesadores de una maquina usando para esto procesos. En particular, en este modulo se puede encontrar un objeto, conocido como Pool, que ofrece una forma de paralelizar la ejecución de funciones a través de la distribución simultanea de estas tareas con diferentes argumentos de entrada en los módulos de procesamiento disponibles. Para la simulación de este trabajo entonces, se hace uso de esta clase para optimizar el tiempo de ejecución del programa como se explicará en la metodología.

Metodología

La simulación del Modelo de Ising dos dimensional consiste en tomar una maya cuadrada de partículas y evolucionar la configuración de espines de acuerdo con la teoría de colectivos canónicos de física estadística siguiendo para esto un algoritmo de MCMC (Markov chain Monte Carlo), en específico el algoritmo de Metropolis-Hasting. Los detalles de esto ya fueron detallados en [mi trabajo], pero se puede resumir las tareas del programa como se muestra en el recuadro titulado “Simulación Ising 2D” de la figura.



Como se puede ver, un proceso de simulación consiste básicamente en: Dado un tamaño de maya (L) y la temperatura del sistema (beta), se corren N veces la tarea etiquetada como Metropolis-Hasting, cuya ejecución lleva una configuración inicial de espines (que puede ser ordenada o desordenada) en m pasos del algoritmo a una final de las L^2 posibles configuraciones del sistema de acuerdo a la distribución de probabilidad del modelo.

Como se mencionó al comienzo, uno de los objetivos es analizar el tiempo gastado en la ejecución de las simulaciones, entonces, la idea es correr para los tamaños diferentes tamaños y ver el comportamiento de Ts vs L, donde Ts es el tiempo que toma la ejecución de “Simulación Ising 2d” de forma serial, ósea sin ser paralelizada. Por otro lado, también se analizará Tp vs L, donde Tp es el tiempo de ejecución pera de la versión paralelizada del problema, que consiste en las tareas que se representan gráficamente en la figura.

Diagrama

Descripción generada automáticamente

Se resalta el concepto de paralelización de tareas del que se hablaba antes, note de la figura que el problema se optimizo ejecutando simultáneamente los N procesos de Metropolis-Hasting que no tienen correlación entre ellos.

Para el otro propósito de este trabajo, correr una simulación de mayor tamaño, se plantea entonces la ejecución de un programa paralelizado con la estructura que se plasma en el diagrama de la figura

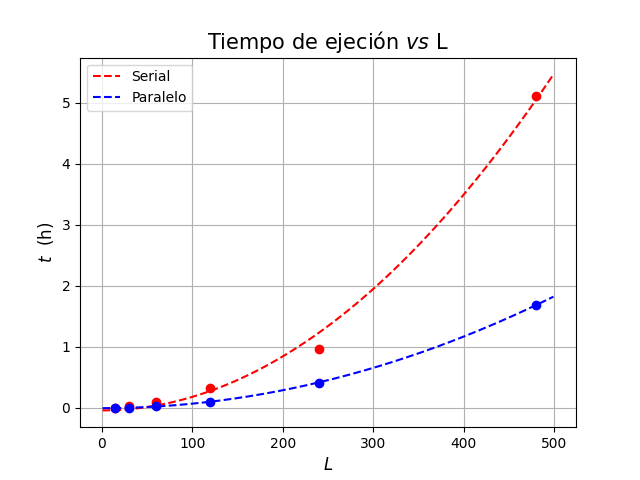
Diagrama

Descripción generada automáticamente

Para lo cual se corre en paralelo diferentes simulaciones para varios valores de temperatura, con lo cual se espera obtener un diagrama que muestre la transición de fase a través de la magnetización del sistema. Cabe resaltar que en este ultimo esquema de paralelización, el proceso de simulación no corresponde con el proceso paralelizado que se planteó en la figura (2), sino con el proceso secuencial de ejecutar N veces el algoritmo de Metrópolis.

Resultados

La paralelización del proceso “Simulación Ising 2d” que se planteo en la figura 2, represento una mejora bastante buena en el tiempo de ejecución, pudiendo llevar una tarea que tardaba alrededor de 5 horas para una maya grande de L=480, a un poco menos de un par de horas. En la figura se presenta gráficamente este resultado, en donde la curva roja representa el incremento en el tiempo de ejecución para el programa sin paralelizar y en azul se presentan los tiempos del programa ya optimizado.



De los datos de tiempo se pudo promediar las fracciones Ts(L)/Tp(L), obteniendo un valor de 3.5+-0.7, que representa una cantidad conocida como Aceleración observada (OBSERVED SPEEDUP)[link]

Ahora, el propósito de correr mayas grandes usando la paralelización planteada en la figura 4 no presenta el rendimiento esperado, aunque si se mejora con respecto a tratar de correrlo en serial. Si se analiza con detalle el proceso que ese programa conlleva, podrá estar de acuerdo con que el tiempo de cómputo sigue siento alto, esto porque, con base en la figura 5, una simulación de 250x250 partículas tardaría alrededor de 1h30, y con un esquema ideal como el de la figura 4, podríamos correr para cada beta en simultaneo en menos de 2 horas, sin embargo, la situación está lejos de ser la idea, no es posible que todos los procesos para cada beta diferente se ejecuten al tiempo, pues el numero de procesadores es finito y no lo suficiente para abarcar los casi 100 procesos deseados (0.1, 1, 0.01), en su lugar, con una maquita que tiene a su disposición 4 núcleos de procesamiento, tomaría 100/4 \* 1h30 terminar el proceso de simulación deseado. Por tanto, sin más tiempo para correr esta simulación, se decide presentar en este trabajo el análisis respecto a los procesos de paralelización solamente y se omite el análisis de transición de fase del sistema.

A pesar de no haber podido generar una simulación de una maya grande, se opto por correr los mismo tamaños del [mitrabajo] con lo que se obtuvo una mejora en el valor del parámetro crítico, recuperando los valores 0.56 y 0.66 para las mayas de L = 10 y L=30 respectivamente, que representa un reducción de error con respecto al presentado en [mitrabajo] de casi 17% para la medida de 0.56 por ejemplo.

Conclusiones