Índice general

Capítulo 1

Introducción

1.1. Motivación

Gran cantidad de problemas como simulación del clima, análisis de genómas, investigación en nuevas energía o aprendizaje de máquina [Pac11] requieren gran poder computacional para ser resueltos. Por lo que mayor poder de cómputo disponible aumenta la cantidad de problemas tratables. En el pasado, esto se lograba aumentando la velocidad y densidad de transistores en los procesadores. Sin embargo, esta técnica está llegando a un límite físico.

Así que en los últimos años, para obtener un aumento en el rendimiento se usan sistemas paralelos. Estos pueden variar desde un procesador con varios núcleos hasta redes de computadores. El tener varias unidades de cómputo permite obtener mejoras de rendimiento a pesar de no tener procesadores más veloces.

Pero hacerlo introduce problemas concernientes a la coordinación de todas las partes. Los detalles del problema y las diferentes técnicas y herramientas para lidiar con ellos dependen bastante de la infraestructura particular.

1.2. Objetivo

El objetivo general de este trabajo consiste en analizar que tanto se puede mejorar el rendimiento de una heuristica existente para el problema de la optimización de cadenas de suministros.

Los objetivos particulares consisten en

- Determinar los puntos de paralelización de la técnica existente
- Analizar las mejoras en tiempos de la implementación paralela
- Analizar las mejoras en calidad de la implementación paralela

1.3. Metas

Se establecieron como metas:

Las contribuciones de este trabajo son:

1.4. Organización de la tesis

Este trabajo se desarrolla en seis capítulos, incluyendo esta introducción, y una sección adicional de apéndices que complementan la investigación.

En capítulo 2, se presenta el problema de cadenas de suministros. Se incluye su formalización como problema de optimización, se mencionan las dificultades para obtener soluciones óptimas y las técnicas usadas para resolverlo. En capítulo 3 se presentan los diferentes tipos de paralelismo y se estudian algunas técinas comunes para paralelizar algoritmos secuenciales. En capítulo 4 se analiza la heuristica propuesta en capítulo 2 usando los criterios presentados en capítulo 3 y se propone una versioń paralelizada. En capítulo 5 se realiza una comparación cualitativa entre los recursos y la calidad de las soluciones tanto de la solución secuencial como la solución paralela. Se concluye el trabajo con el capítulo 6, donde se exponen los comentarios finales y se sugiere el trabajo a relizar en próximas investigaciones.

Se anexan además varios apéndices. En el apéndice A se describe la arquitectura de las tarjetas CUDA de Nvidia, así como conceptos básicos de CUDA y `nvcc`. En el apéndice B se introducen conceptos básicos del lenguaje Julia, así como algunas bibliotecas útiles. En el apéndice C se da un breve repaso de programas lineals y del método Simplex.

Capítulo 2

Cadenas de Suministros

La *logística* de un sistema consiste en satisfacer ciertas demandas, de materias primas, de tiempo, bajo ciertas restricciones, normalmente presupuesto, mientras se intenta optimizar una medida de desempeño, normalmente el costo de materiales y transporte.

En el ámbito industrial, la logística se denomina *cadena de suministros*. Esta tiene dos etapas, la *cadena de producción* y la *cadena de distribución* [GLM04].

2.1. Formalización

Su estrucutura depende del caso en particular, pero siguiendo el modelo simplificado planteado por [Lic16], se considera que la cadena de producción tiene entrada fija de proveedores y plantas de ensamblaje. A su vez, la cadena de distribución tiene centros de distribución, proveídos por las plantas de ensamblaje, y puntos de venta. Cada instalación tiene un costo de operación fijo. El transporte entre cada instalación tiene un costo unitario. Cada instalación tiene una capacidad de producción máxima, y los puntos de venta tiene una demanda. Formalmente el problema tiene

- Proveedores S, plantas de ensamplaje P, centros de distribución D y puntos de venta C. Las instalaciones se denominan $I = S \cup P \cup D \cup C$.
- Matrix de costo unitario $U \in \mathbb{M}_{|I|\times|I|}$, donde U_{ij} representa el costo de mover una unida de productos de la instalación i a la instalación j
- Cada instalación tiene un costo fijo $F : I \to \mathbb{R}^+$.
- Cada instalación tiene una capacidad (o demanda) fija $W : I \to \mathbb{R}^+$.
- P_{max} y D_{max} para limitar la cantidad de plantas y centros de distribución a abrir.

El objectivo de la logística es encontrar como repartir la carga entre las plantas de producción y en los centros de distribución para respetar las restricciones y minimizando el costo. Formalmente

- Determinar $O : I \to \{0, 1\}$ que indique que instalaciones hay que abrir. Tanto los proveedores como los puntos de venta tiene que estar abiertos siempre.
- Determinar matrix de cargas X, donde X_{ij} es la cantidad de productos que va a circular entre la instalación i y j.

2.2. Programación lineal

Dado una O fija, encontrar X se puede formular como un problema de programación lineal, por lo que se pueden encontrar los valores óptimos en tiempo polinomial. La formulación sería minimzar

$$z(O) = \sum_{i,j \in I^2} O(i)O(j)X_{i,j}U_{i,j} + \sum_{i \in P \cup D} O(i)F(i) \text{ costo fijo + costo unitario} \tag{2.1}$$

sujeto a

$$\sum_{j \in I} X_{i,j} \leq W(i), \forall i \in I \text{ respetar capacidades}$$
$$\sum_{i \in I} X_{i,c} \geq W(c), \forall c \in C \text{ satisfacer demandas}$$
$$\sum_{i \in I} X_{i,j} = \sum_{k \in I} X_{j,k}, \forall j \in P \cup D \text{ flujo constante}$$
$$\sum_{p \in P} O(p) \leq P_{max}, \sum_{d \in D} O(d) \leq D_{max} \text{ respetar máximo de instalaciones}$$

2.3. Enjambre de partículas

Una de las técnicas presentadas por [Lic16] es una variante de la técnica de optimización por enjambre de partículas (PSO por sus siglas en inglés). Esta es una heuristica bioinspirada en los movimientos de parvadas de pájaros, que carecen de lider pero imitando localemente a las aves cercanas surgen patrones complejos.

La técnica original funciones con espacios continuos, en general $\mathbb{R}^n$, así que la técina presentada por [Lic16] es una variante para trabajar con espacios binarios. En esta técnica se tiene un conjunto X de k partículas, donde cada una representa una solución al problema de optimización.

Cada partícula tiene una velocidad V asociada. En cada paso, la velocidad se actualiza usando la ecuación (2.2).

$$v \leftarrow \phi v + c_1 U(0,1)(x^\star - x) + c_2 U(0,1)(\mathbf{x}^\star - x) \tag{2.2}$$

donde $x^\star$ es la mejor posición alcanzada por$x$, $\mathbf{x}^\star$ es la mejor posición global actual, ϕ es la constante de aceleración, c_i son constantes para determinar cuando influencia la mejor solución personal y global.

Luego, usando la velocidad actualizada se puede actualizar el $i$ésimo componente de la particula x con la ecuación (2.3).

$$x_i \leftarrow \begin{cases} 0, & \text{si } U(0,1) \leq f(v) \\ 1, & \text{en otro caso} \end{cases} \tag{2.3}$$

donde $f(v) = \frac{1}{1+e^{-v}}$. Una secuencia general de los pasos de la optimización se puede ver en el algoritmo 1.

Esta técnica se puede usar para encontrar las instalaciones óptimas para abrir, pues es un vector binario, con un flujo de productos fijo. Para optimizar ese flujo, hay que notar que se tiene una x fija en cada ciclo, así que se puede construir la formulación de la ecuación (2.1) para optimizar los flujos de productos.

Sería una optimización que se alterna. Con BPSO optimizando las instalaciones a abrir y algún método de programación lineal para optimizar los flujos de productos.

Algoritmo 1 Enjambre de particulas binarias

Entrada Función z a optimizar, n número de partículas, k número de rondas, ϕ constante de aceleración, c_1, c_2 constantes de influencia
Salida Mejor solución x encontrada

```
1:  funcion BPSO(z, k, n)
2:      Generar n soluciones aleatorias y guardarlas en X
3:      Inicilizar velocidades V en cero y mejores soluciones X*
4:      Encontrar mejor solución actual x*
5:      para t ∈ 0..k hacer
6:          para i ∈ 0..n hacer
7:              Actualizar V[i] usando la ecuación (2.2)
8:              Actualizar X[i] usando la ecuación (2.3)
9:              Aplicar optimización adicional                ▷ Opcional
10:             Actualizar mejor solución global x* y local X*
11:         fin para
12:     fin para
13:     devolver x*
14: fin funcion
```

Capítulo 3

Paralelismo

Más aún, cierta infraestrucutura o técnica pueden ser más conveniente de usar para ciertos tipos de problemas. En este capítulo se van a explorar diferentes clasificaiones de arquitecturas paralelas y sus peculiaridades, además de dar algunos criterios para clasificar los problemas y decidir la mejor manera de crear una solución paralela a un problema en particular.

3.1. Tipos de arquitectura

Tradicionalmente se dividen los tipos de sistemas en base a como tratan el flujo de datos y de instrucciones. Esto se conoce como taxonomía de Flynn. Están las siguientes categorías

- *SISD* (Single Instruction Single Data): sistemas sin paralelismo.
- *SIMD* (Single Instruction Multiple Data): sistemas que pueden aplicar una operación a vectores o arreglos de datos en una unidad de tiempo
- *MIMD* (Multiple Data Multiple Instructions): sistemas donde cada unidad de procesamiento es independiente.

3.1.1. Sistemas *SISD*

Contrario a su nombres, estos modelos pueden tener ciertos mecanismos paralaelos. Estos son optimizaciones a nivel de hardware que permiten ejecutar instrucciones de forma no secuencial . Esto no se considera paralelismo en la taxonomía de Flynn ya que estos mecanismos no están disponibles para los desarrolladores. Aún así, vale la pena revisar algunos.

- *Pipelining*: Consiste en partir una instrucción en pasos atómicos. Esto es sacar los datos de memoria, revisar el tipo de operación, aplicar la operación, guardar los datos resultantes, entre otros. Cada sección se le asigna a diferentes secciones de la unidad de control. Entonces, cuando un

3.1.2. Sistemas *SIMD*

En estos sistemas hay una única unidad de control con varias unidades aritmético lógicas (*ALU*). Al recibir la instrucción, la unidad de control avisa a todas las *ALUs* para

que apliquen la operación al dato correspondiente. Este tipo de paralelismo se denomina **paralelismo de datos** .

Deben tener registros capaces de guarda vectores y operaciones optimizadas para leer y escribir de los elementos en estos registros. Todas las operaciones son síncronas y totalmente uniformes. Además de operaciones sobre los elementos, suelen haber operaciones que actúan sobre los vectores sin tener que acceder a cada elementos, como obtener la longitud del vector.

A pesar de solo ser factibles en operaciones muy restringidas, estos sistemas no suelen introducir problemas nuevos a los programas y su uso es bastante directo. Además, la mejora en rendimiento al usarlos suele ser aceptable.

Hoy en día, cada núcleo de un *CPUs* cuentan con instrucciones que permiten este tipo de operaciones. No son un sistema exclusivamente *SIMD*, ya que tener varios núcleos le dan características *MIMD*.

Las unidades de procesamiento gráfico (*GPU*) también tienen este tipo de funcionalidad. Además, como las imágenes suelen ocupar mucha memoria, tiene memoria de gran tamaño optimizada para el manejo con hilos a nivel de hardware . Curiosamente, esto hace que tenga mal desempeño en problemas pequeños. Al igual que los *CPUs*, tampoco son exclusivamente un sistema *SIMD*, ya que suelen tener más de un núcleo.

3.1.3. Sistemas *MIMD*

3.2. Tipos de Problemas

Pancake [Pan96] propone algunas reglas que a grandes rasgos permitan determinar si vale la pena dar una solución paralela a un problema.

1. Si el problema es de paralelismo perfecto, probablemente sea fácil de paralelizar y se obtengan ganancia considerable en rendimiento.

2. Si el problema es de paralelismo de *pipeline*, sólo valdría la pena paralelizarlo si se puede balancear la carga de trabajo entre las etapas.

3. Si el problema es de paralelismo totalmente síncrono, valdría la pena paralelizar dependiendo de que tan uniformemente se puede distriubir la carga.

4. Si el problema es de paralelismo vagamente síncrono, solo valdría la pena paralelizarlo si hay muy pocas interacciones entre procesos.

5. Un problema de paralelismo perfecto podría implementarse en un sistema *MIMD*, pero sería problemático hacerlo en un sistema *SIMD*.

6. Un problema de paralelismo de *pipeline* probablemente se desempeñe mejor en un sistema de memoria compartida o en un SMP (siempre y cuando cada etapa queda en una unidad de procesamiento). Podría funcionar decentemente en un sistema distribuido si la conexión entre etapas es lo suficientemente rápida.

7. Un problema totalmente síncrono probablemente se desempeñe mejor en un sistema SIMD. Pero esto requeriría que todas las operaciones sean uniformes. De no ser así, una opción decente sería un sistema de memoria compartida.

8. Para un problema vagamente síncrono la mejor opción sería un sistema de memoria compartida. Un sistema distribuido funcionaría siempre y cuando haya muchas operaciones entre cada interacción de los procesos.

9. Lenguaje

10. El tiempo de ejecución de una versión secuencia del programa se puede usar para estimar el desempeño del programa paralelo.
11. Debido a las restricciones de las secciones secuenciales del programa, probablemente no valga la pena paralelizar un programa con una fracción paralelizable menor a 0.95.
12. Hay que estimarlo con cuidado como cambia el desempeño del programa al cambiar la entrada. Este depende de la naturaleza del problema.
13. El desempeño final siempre será peor al estimado.
14. Aunque se puede intentar resolver pérdidas de tiempo surgidas por la concurrencia, en general dependerá de la naturaleza del problema y del equipo usado.
15. Un problema de baja granuralidad tendrá un desempeño decente en cualquier sistema. Un sistema de granuralidad media/alta probablemente solo sea decente en un sistema SIMD.
16. Para sistemas distribuidos, es útil estimar la equivalencia de mensaje.

3.3. Métricas de desempeño

3.4. Resumen

Capítulo 4

Paralelización de *B-PSO*

Capítulo 5

Resultados

Capítulo 6

Conclusiones y Trabajo Futuro

Apéndice A

Arquitectura de CUDA y CUDA-C

A.1. Motivaciones

A.2. CPUs

A.3. GPUs

Kernels Indices Huesped y devices

A.4. TPUs

A.5. Resumen

Apéndice B

Especificación de Julia

B.1. Sintaxis

B.2. Estructuras básicas

B.3. Sistema de tipos

Apéndice C

Programación Lineal

Un problema de optimización lineal requiere maximizar (o minimizar) una función lineal z bajo m restricciones lineales.

$$a_j \leq k_j \tag{C.1}$$
$$a_j \geq k_j \tag{C.2}$$
$$a_j = k_j \tag{C.3}$$

donde tanto z como a_i son combinaciones lineales de un conjunto $x_i{}_{i=1}^{k}$ de variables y k_j son números reales. Esto puede ser restrictivo en varios casos, pero aún así existen una gran cantidad de problemas que pueden ser formulados de esta forma. Usando la linealidad de la función objetivo y de las restricciones, se han desarrollado varios métodos para encontrar soluciones óptimas en tiempo razonable.

C.1. Forma ecuacional

Un problema está en forma ecuacional si todas sus restricciones (excepto las de no negativiad) son ecuaciones (i.e. $a_j = k_j$) con $k_j \geq 0$ y todas las variables son no negativas. Para todo problema, es posible construir un problema en forma ecuacional con la misma solución óptima. Para hacer esto se puede hacer lo siguiente

- Si $k_j < 0$, se multiplica la restricción por -1 y se usa alguno de los casos siguientes.
- Si se tiene una variable x_j que puede tener valores positivos y nagativos, se usan dos nuevas variables para obtener la ecuación $x_j = x_j^+ - x_j^-$, con $x_j^-, x_j^+ \geq 0$.
- Para las desigualdades de la forma $a_j \leq k_j$, se introduce una variable s_j de *sobra* (*slack* en inglés). La ecuación equivalente sería $a_j + s_j = k_j$ y además se agrega la restricción $s_j \geq 0$
- Para las desigualdades $a_j \geq k_j$, se hace algo análogo al caso anterior. La ecuación resultante es $a_j - s_j = k_j$ y $s_j \geq 0$.

Índice alfabético

Referencias

[GLM04] Gianpaolo Ghiani, Gilbert Laporte y Roberto Musmanno. *Introduction to Logistics Systems Planning and Control*. USA: Halsted Press, 2004. isbn: 0470849177.

[Lic16] Diana Xochitl Canales Licona. «Un Algoritmo de Cúmulo de Partículas para la Optimización del Costo de Operación de una Red de Cadena de Suministro de Múltiples Etapas mediante una Estrategia de Distribución Física». Maestro en Ciencias en Ingeniería Industrial. Ciudad del Conocimiento, Mineral de la Reforma, Hidalgo, México: Universidad Autónoma del Estado de Hidalgo, jul. de 2016.

[Pac11] Peter Pacheco. *An Introduction to Parallel Programming*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. isbn: 9780123742605. doi: 10.1016/c2009-0-18471-4. url: https://doi.org/10.1016/c2009-0-18471-4.

[Pan96] C. Pancake. «Is parallelism for you?» En: *IEEE Computational Science and Engineering* 3.2 (1996), págs. 18-37. doi: 10.1109/99.503307.