Introducción a la programación paralela

Colaboratorio Nacional de Computación Avanzada (CNCA)

2014

Contenidos

Definición

- Estudio, diseño, análisis e implementación de algoritmos que realizan más de una tarea por unidad de tiempo.
- Involucra el uso de múltiples procesadores y, por lo general, mucha memoria.

Definición

- Estudio, diseño, análisis e implementación de algoritmos que realizan más de una tarea por unidad de tiempo.
- Involucra el uso de múltiples procesadores y, por lo general, mucha memoria.
- Paradigma para la modelación computacional, útil para muchas disciplinas científicas.

Definición

- Estudio, diseño, análisis e implementación de algoritmos que realizan más de una tarea por unidad de tiempo.
- Involucra el uso de múltiples procesadores y, por lo general, mucha memoria.
- Paradigma para la modelación computacional, útil para muchas disciplinas científicas.

Justificación

- El mundo natural es paralelo; muchos eventos complejos suceden simultáneamente y se relacionan entre sí. Ej:
  - Formación de galaxias
  - Cambio climático
  - Ensamblaje de automóviles
- Permite trabajar con problemas complejos, imprácticos o imposibles para un solo CPU.

Justificación

- El mundo natural es paralelo; muchos eventos complejos suceden simultáneamente y se relacionan entre sí. Ej:
  - Formación de galaxias
  - Cambio climático
  - Ensamblaje de automóviles
- Permite trabajar con problemas complejos, imprácticos o imposibles para un solo CPU.
- Límites de la computación secuencial:
  - Velocidad de transimisión depende del medio físico y la proximidad de los componentes
  - Miniaturización limitada, aunque sea en escala atómica.
  - Limitaciones económicas: múltiples CPUs de bajo costo vs. 1 CPU extremadamente rápido.

Justificación

- El mundo natural es paralelo; muchos eventos complejos suceden simultáneamente y se relacionan entre sí. Ej:
  - Formación de galaxias
  - Cambio climático
  - Ensamblaje de automóviles
- Permite trabajar con problemas complejos, imprácticos o imposibles para un solo CPU.
- Límites de la computación secuencial:
  - Velocidad de transimisión depende del medio físico y la proximidad de los componentes
  - Miniaturización limitada, aunque sea en escala atómica.
  - Limitaciones económicas: múltiples CPUs de bajo costo vs. 1 CPU extremadamente rápido.

Tipos de problemas

- Trivialmente paralelizables:
  - Pocas o ninguna dependencia funcional
  - No hay dependencias de datos
  - Requieren mínima comunicación
- Fuertemente acoplados:
  - Fuertes dependencias estructurales y/o fucionales.
  - Requieren comunicación y sincronización.

Tipos de problemas

- Trivialmente paralelizables:
  - Pocas o ninguna dependencia funcional
  - No hay dependencias de datos
  - Requieren mínima comunicación
- Fuertemente acoplados:
  - Fuertes dependencias estructurales y/o fucionales.
  - Requieren comunicación y sincronización.
- Híbridos

Tipos de problemas

- Trivialmente paralelizables:
  - Pocas o ninguna dependencia funcional
  - No hay dependencias de datos
  - Requieren mínima comunicación
- Fuertemente acoplados:
  - Fuertes dependencias estructurales y/o fucionales.
  - Requieren comunicación y sincronización.
- Híbridos

Identificación del paralelismo

- El *significado* de un programa se define en términos de su implementación secuencial: ambos producen los mismos resultados.
- Determinar cuándo dos o más cálculos de un programa secuencial pueden ejecutarse en paralelo preservando los resultados finales.

Identificación del paralelismo

- El *significado* de un programa se define en términos de su implementación secuencial: ambos producen los mismos resultados.
- Determinar cuándo dos o más cálculos de un programa secuencial pueden ejecutarse en paralelo preservando los resultados finales.
- Ejemplo 1: paralelizar cálculos que no comparten datos.

Identificación del paralelismo

- El *significado* de un programa se define en términos de su implementación secuencial: ambos producen los mismos resultados.
- Determinar cuándo dos o más cálculos de un programa secuencial pueden ejecutarse en paralelo preservando los resultados finales.
- Ejemplo 1: paralelizar cálculos que no comparten datos.
- Ejemplo 2: paralelizar cálculos con datos compartidos que solamente son leídos (no modificados).

Identificación del paralelismo

- El *significado* de un programa se define en términos de su implementación secuencial: ambos producen los mismos resultados.
- Determinar cuándo dos o más cálculos de un programa secuencial pueden ejecutarse en paralelo preservando los resultados finales.
- Ejemplo 1: paralelizar cálculos que no comparten datos.
- Ejemplo 2: paralelizar cálculos con datos compartidos que solamente son leídos (no modificados).

Identificación del paralelismo

Dos cálculos C_1 y C_2 pueden ser ejecutados en paralelo sin sincronización si y sólo si:

1. C_1 no modifica en una región que es leída posteriormente por C_2. Conocido como Lectura tras escritura (RAW, read-after-write).
2. C_1 no lee una región que luego es modificada por C_2. Conocido como Escritura tras lectura (WAR, write-after-read).

Identificación del paralelismo

Dos cálculos C_1 y C_2 pueden ser ejecutados en paralelo sin sincronización si y sólo si:

1. C_1 no modifica en una región que es leída posteriormente por C_2. Conocido como Lectura tras escritura (RAW, read-after-write).
2. C_1 no lee una región que luego es modificada por C_2. Conocido como Escritura tras lectura (WAR, write-after-read).
3. C_1 no modifica una región que luego es sobreescrita por C_2. Conocido como Escritura tras escritura (WAW, write-after-write).

Identificación del paralelismo

Dos cálculos C_1 y C_2 pueden ser ejecutados en paralelo sin sincronización si y sólo si:

1. C_1 no modifica en una región que es leída posteriormente por C_2. Conocido como Lectura tras escritura (RAW, read-after-write).
2. C_1 no lee una región que luego es modificada por C_2. Conocido como Escritura tras lectura (WAR, write-after-read).
3. C_1 no modifica una región que luego es sobreescrita por C_2. Conocido como Escritura tras escritura (WAW, write-after-write).

Representación de datos

Los datos pueden ser representados en forma homóloga a las arquitecturas:
Memoria compartida:

- Los datos residen en la memoria global
- Múltiples hilos manipulan las estructuras de datos
- La escritura debe ser sincronizada
- Permite utilizar paralelismo de *grano fino*

Memoria distribuida:

- Los datos están distribuidos en espacios de memoria independientes
- Un proceso accede sólo su memoria local
- Requiere comunicación para lectura y escritura de datos remotos, usualmente a través de paso de mensajes
- Por lo general utiliza paralelismo de *grano grueso*

Representación de datos

Los datos pueden ser representados en forma homóloga a las arquitecturas:
Memoria compartida:

- Los datos residen en la memoria global
- Múltiples hilos manipulan las estructuras de datos
- La escritura debe ser sincronizada
- Permite utilizar paralelismo de *grano fino*

Memoria distribuida:

- Los datos están distribuidos en espacios de memoria independientes
- Un proceso accede sólo su memoria local
- Requiere comunicación para lectura y escritura de datos remotos, usualmente a través de paso de mensajes
- Por lo general utiliza paralelismo de *grano grueso*

Granularidad

Grano fino:

- Poco trabajo computacional entre cada evento de sincronización
- Relacionado con memoria compartida, hilos y tecnologías como OpenMP.

Grano grueso:

- Mucho trabajo computacional entre cada sincronización.
- Enfocado en la distribución inicial de datos y procesamiento pesado
- Relacionado con computación cluster, memoria distribuida, procesos y tecnologías como MPI.

Granularidad

Grano fino:

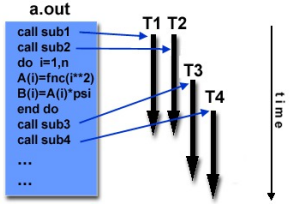
- Poco trabajo computacional entre cada evento de sincronización
- Relacionado con memoria compartida, hilos y tecnologías como OpenMP.

Grano grueso:

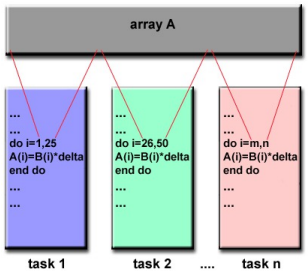
- Mucho trabajo computacional entre cada sincronización.
- Enfocado en la distribución inicial de datos y procesamiento pesado
- Relacionado con computación cluster, memoria distribuida, procesos y tecnologías como MPI.

Modelos de bajo nivel

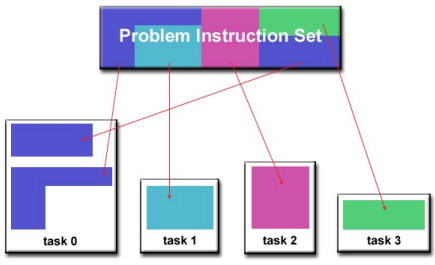
Hilos



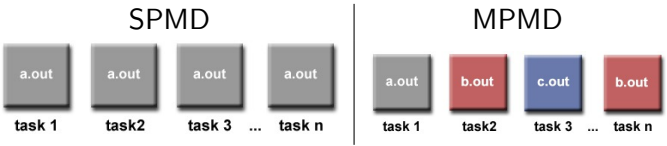
Datos



Tareas



Modelos de alto nivel



Medidas de Rendimiento

Aceleración: tiempo secuencial / tiempo paralelo

- $S = T_s/T_p$
- Amdahl: $S = \frac{T_s}{(1-r)T_s + rT_s/p} = \frac{1}{(1-r)+r/p}$
- Aceleración máxima: $\bar{P} = \frac{T_s}{T_\infty}$, T_∞ ruta crítica. $\bar{P}$ *paralelismo* de un algoritmo.
- Si $S = p$, es decir $T_p = T_s/p$ entonces la aceleración es *lineal*.
- Aceleración *supralineal*: $T_p < T_s/p$. ¿Por qué?

Medidas de Rendimiento

Eficiencia: aceleración / procesadores

- $E = S/p = T_s/pT_p$

Trabajo: tiempo de ejecución total

- $W_s = T_s$
- $W_p = \sum_{i=1}^{p} W_i = pT_p$

Overhead: trabajo paralelo que *no* forma parte del secuencial

- $T_o = pT_p - T_s$

Medidas de rendimiento

Escalabilidad: capacidad del algoritmo (paralelo) para mejorar su rendimiento cuando utiliza más recursos.

Un algoritmo es *escalable* si incrementa su rendimiento en forma proporcional a los recursos disponibles. O bien, si se incrementa el tamaño del problema y los recursis, la eficiencia permanece igual.

Medidas de rendimiento

Escalabilidad: capacidad del algoritmo (paralelo) para mejorar su rendimiento cuando utiliza más recursos.

Un algoritmo es *escalable* si incrementa su rendimiento en forma proporcional a los recursos disponibles. O bien, si se incrementa el tamaño del problema y los recursis, la eficiencia permanece igual.

Es posible redefinir eficiencia en términos de overhead:

- $E(n,p) = \frac{T_s(n,p)}{T_o(n,p)+T_s(n,p)} = \frac{1}{T_o(n,p)/T_s(n,p)+1}$
- Eficiencia está determinada por el overhead.
- Si n aumenta igual que p (1:1), la eficiencia decrece si T_o crece más rápidamente que $n = p$.

Medidas de rendimiento

Escalabilidad: capacidad del algoritmo (paralelo) para mejorar su rendimiento cuando utiliza más recursos.
Un algoritmo es *escalable* si incrementa su rendimiento en forma proporcional a los recursos disponibles. O bien, si se incrementa el tamaño del problema y los recursis, la eficiencia permanece igual.
Es posible redefinir eficiencia en términos de overhead:

- $E(n,p) = \frac{T_s(n,p)}{T_o(n,p)+T_s(n,p)} = \frac{1}{T_o(n,p)/T_s(n,p)+1}$
- Eficiencia está determinada por el overhead.
- Si n aumenta igual que p (1:1), la eficiencia decrece si T_o crece más rápidamente que $n = p$.

Objetivo: encontrar función que determine el crecimiento de n con respecto a p de forma que E sea *constante*.

Medidas de rendimiento

Escalabilidad: capacidad del algoritmo (paralelo) para mejorar su rendimiento cuando utiliza más recursos.
Un algoritmo es *escalable* si incrementa su rendimiento en forma proporcional a los recursos disponibles. O bien, si se incrementa el tamaño del problema y los recursis, la eficiencia permanece igual.
Es posible redefinir eficiencia en términos de overhead:

- $E(n,p) = \frac{T_s(n,p)}{T_o(n,p)+T_s(n,p)} = \frac{1}{T_o(n,p)/T_s(n,p)+1}$
- Eficiencia está determinada por el overhead.
- Si n aumenta igual que p (1:1), la eficiencia decrece si T_o crece más rápidamente que $n = p$.

Objetivo: encontrar función que determine el crecimiento de n con respecto a p de forma que E sea *constante*.

Producto punto (secuencial)

Algorithm 1 Producto punto

1: Sean a y b vectores
2: $N = ||a||$
3: producto $= 0$
4: **for** $i = 0 : N$ **do**
5: producto $+= a_i \cdot b_i$
6: **end for**
7: Retornar producto

Paralelización del Producto punto

- Cada producto es independiente de los demás
- La suma y el producto son operaciones lineales: $\sum_{i=1}^{n} a_i b_i = \sum_{i=1}^{k} a_i b_i + \cdots + \sum_{i=k}^{n} a_i b_i$
- Si tenemos p procesadores y vectores de norma n, podemos asignar a cada procesador n/p elementos, para luego sumar las sumas parciales de cada procesador.

Paralelización del Producto punto

Algorithm 2 Producto punto paralelo

1: $p =$ cantidad de procesadores
2: a y b vectores
3: $N = ||a||$
4: $n = N/p$
5: Declarar nuevos vectores a_local y b_local de norma n
6: Distribuir n elementos de a a cada proceso y guardarlos en a_local
7: Distribuir n elementos de b a cada proceso y guardarlos en b_local
8: $producto\_local = 0$
9: **for** $i = 0 : n$ **do**
10: $\quad producto\_local += a\_local_i \cdot b\_local_i$
11: **end for**
12: Reducir $producto\_local$ en $producto$
13: Retornar producto

Análisis

Sin comunicación:

- Declaración de vectores locales: 1
- Producto punto: $p \cdot n\_local = n$
- $S = \frac{n}{n/p} = p$
- $E = \frac{n}{p \cdot (n/p)} = 1$

Comunicación (Modelo de Hockney):

- Distribución de vectores: $2T_{com}(n) = 2p(\alpha + \beta n)$
- Reducción: $T_{com}(p) = (\alpha + \beta p)$

Análisis

Sin comunicación:

- Declaración de vectores locales: 1
- Producto punto: $p \cdot n\_local = n$
- $S = \frac{n}{n/p} = p$
- $E = \frac{n}{p \cdot (n/p)} = 1$

Comunicación (Modelo de Hockney):

- Distribución de vectores: $2T_{com}(n) = 2p(\alpha + \beta n)$
- Reducción: $T_{com}(p) = (\alpha + \beta p)$

Tiempo computacional:

$$T(n) = n + 2p(\alpha + \beta n) + (\alpha + \beta p) + 1$$

Análisis

Sin comunicación:

- Declaración de vectores locales: 1
- Producto punto: $p \cdot n\_local = n$
- $S = \frac{n}{n/p} = p$
- $E = \frac{n}{p \cdot (n/p)} = 1$

Comunicación (Modelo de Hockney):

- Distribución de vectores: $2T_{com}(n) = 2p(\alpha + \beta n)$
- Reducción: $T_{com}(p) = (\alpha + \beta p)$

Tiempo computacional:

$$T(n) = n + 2p(\alpha + \beta n) + (\alpha + \beta p) + 1$$