Fundamentos de la Computación Paralela

Comparación con la Computación Secuencial y Herramientas Clave

27 de mayo de 2025

¿Qué es la Computación Paralela?

- La **computación paralela** es un tipo de computación en el que se ejecutan muchas **cálculos o procesos simultáneamente**.
- El objetivo es resolver problemas grandes que se pueden dividir en partes más pequeñas, las cuales se resuelven *al mismo tiempo*.
- No es exactamente los mismo que la **computación de alto rendimiento (HPC)**.
- Ha ganado interés debido a las **restricciones físicas** que limitan el aumento de la frecuencia del procesador (frequency scaling) y la preocupación por el **consumo de energía y calor**.

Secuencial vs. Paralela

- **Computación Secuencial:**
  - Los programas se escriben para ejecutarse como un **flujo en serial de instrucciones**.
  - Solo se ejecuta **una instrucción a la vez** en una sola CPU.
- **Computación Paralela:**
  - Utiliza **múltiples elementos de procesamiento simultáneamente** para resolver un problema.
  - Esto se logra dividiendo el problema en **partes independientes**.
  - El aumento del rendimiento ahora proviene de **incrementar el número de procesadores (nucleos o nodos)** en lugar de hacer que un solo core sea más rápido.

Arquitecturas Paralelas

- Principios de **arquitecturas paralelas y distribuidas**.
- Se pueden clasificar según el **nivel de soporte hardware al paralelismo**:
  - **Computadoras con múltiples elementos de procesamiento en una sola máquina:**
    - **Multi-core:** Múltiples unidades de procesamiento (cores) en el mismo chip. Cada core es independiente.
    - **Symmetric Multiprocessing (SMP):** Múltiples procesadores idénticos que comparten memoria y se conectan vía bus.
  - **Sistemas que usan múltiples computadoras:**
    - **Clusters:** Grupo de computadoras acopladas laxamente que trabajan juntas.
    - **Massively Parallel Processors (MPP):** Computadoras con muchos procesadores interconectados con redes especializadas.
    - **Grid computing:** Utiliza computadoras a través de Internet.

Desafíos de la Programación Paralela

- Los algoritmos explícitamente paralelos son **más difíciles de escribir** que los secuenciales.
- La **concurrencia** introduce nuevas clases de **errores de software**, siendo las **race conditions** las más comunes.
- Es necesaria la **sincronización** para acceder a recursos compartidos.
- Posibles problemas como **deadlock** al usar locks.
- La **comunicación y sincronización** entre subtareas son grandes obstáculos para el rendimiento óptimo.
- Las **dependencias** de datos (flujo, anti, salida) restringen la paralelización.
- **Parallel slowdown:** Aumento del tiempo de ejecución por el overhead de comunicación/espera al aumentar la paralelización.

Modelos de Memoria y Comunicación

- **Memoria Compartida (Shared Memory):**
  - Todos los elementos de procesamiento comparten un **espacio de direcciones único**.
  - Acceso a memoria con latencia y ancho de banda uniformes (UMA) o no uniformes (NUMA).
  - La escalabilidad está limitada en comparación con sistemas de memoria distribuida.
- **Memoria Distribuida (Distributed Memory):**
  - Cada elemento de procesamiento tiene su **propio espacio de direcciones local**.
  - La comunicación entre procesos se realiza mediante **paso de mensajes**.
  - Sistemas altamente **escalables**.
  - Ejemplos: Clusters, MPPs.

Leyes de Rendimiento Paralelo

- **Ley de Amdahl:**
  - Establece un **límite superior teórico** a la mejora de velocidad (speedup) de un programa debido a la paralelización.
  - El speedup está limitado por la **fracción de tiempo que la paralelización puede ser utilizada** (la parte secuencial del programa).
  - Muestra que el aumento del número de procesadores genera *rendimientos decrecientes*.
  - La mejora óptima se logra balanceando las partes paralelizables y no paralelizables. Tiene limitaciones.
- **Ley de Gustafson:**
  - Ofrece una **evaluación más realista** del rendimiento paralelo al considerar que el tamaño del problema puede escalar con el número de procesadores.

Herramientas Clave (Libraries y APIs)

- Se han creado lenguajes, librerías, APIs y modelos de programación para computadoras paralelas.
- **OpenMP:**
  - Librería para **programación paralela en sistemas de memoria compartida**.
  - Proporciona directivas y funciones para crear regiones paralelas, gestionar threads y sincronización.
- **MPI (Message Passing Interface):**
  - Librería para **programación paralela en sistemas de memoria distribuida**.
  - Permite la comunicación explícita (paso de mensajes) entre procesos.
  - Es la **API de paso de mensajes más utilizada**.
- Otras librerías para procesamiento numérico a gran escala: **BLAS, LAPACK, ScaLAPACK, ARPACK**.

Aplicaciones Típicas de la Computación Paralela

- Históricamente, **computación científica y simulación** (ej: meteorología).
- Ahora se utiliza en campos variados como **omics** y **economía**.
- Tipos comunes de problemas adecuados para paralelizar:
  - Álgebra Lineal Densa y no-Densa.
  - Métodos Espectrales (como Fast Fourier Transform).
  - Problemas de N-cuerpos.
  - Problemas de Malla Estructurada y No Estructurada (ej: análisis de elementos finitos).
  - Método de Monte Carlo.
  - Lógica Combinacional (ej: criptografía de fuerza bruta).
  - Recorrido de Grafos (ej: algoritmos de ordenación).
  - Programación Dinámica.

Conclusión

- La **computación paralela** es fundamental para lograr aumentos de rendimiento en la era post-frequency scaling.
- Implica un cambio de paradigma de programación, requiriendo manejar la *concurrencia, comunicación y sincronización*.
- Las arquitecturas varían desde **multi-core** en desktops hasta *clusters y MPPs* en supercomputadoras.
- Herramientas como **OpenMP** (memoria compartida) y **MPI** (memoria distribuida) son esenciales para desarrollar programas paralelos.
- Permite abordar y resolver **problemas complejos y de gran escala** que antes eran intratables.
- La investigación en **paralelización automática** continúa, pero la programación explícita es el enfoque dominante.