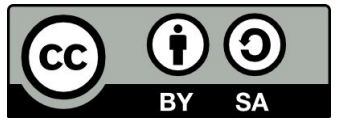
Computación de Altas Prestaciones

Visión global de modelos de programación

José Luis Risco Martín
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid



Índice

Introducción

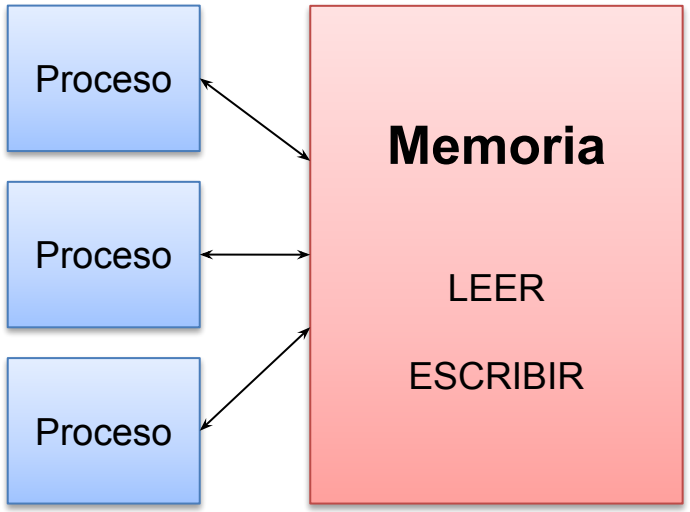
- Existen una variedad de modelos de programación paralela de uso común
  - Memoria compartida (sin threads)
  - Memoria compartida (con threads)
  - Memoria distribuida / Paso de mensajes
  - Datos paralelos
  - Híbrido
  - SPMD
  - MPMD
- Estos modelos no pertenecen a un tipo particular de máquina o arquitectura de memoria.
- ¿Qué modelo usar?
  - No existe el "mejor" modelo, aunque dependiendo del problema a resolver, algunos modelos responden mejores que otros.

Características

- En este modelo de programación, los procesos/tareas comparten un espacio de direcciones común
- Mecanismos como cerrojos o semáforos se usan para controlar el acceso a la memoria compartida, resolver condiciones de carrera e interbloqueos
- Este es quizá el modelo de programación más simple
- **Ventaja:**
  - No existe noción de propiedad de los datos. Todos los procesos ven y tienen igual acceso a la memoria compartida.
- **Desventaja:**
  - En términos de rendimiento, es difícil comprender y gestionar la localidad de los datos.

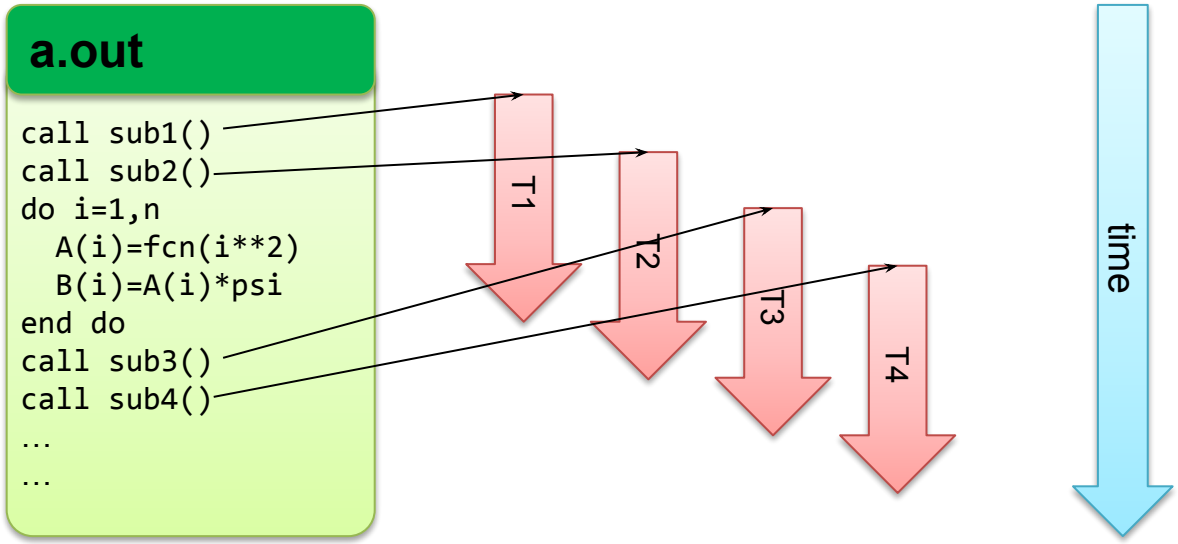
Implementaciones

- Máquinas con memoria compartida
  - POSIX
  - UNIX: shmget, shmat, shmctl, etc.
- Máquinas con memoria distribuida
  - SHMEM (http://en.wikipedia.org/wiki/SHMEM)



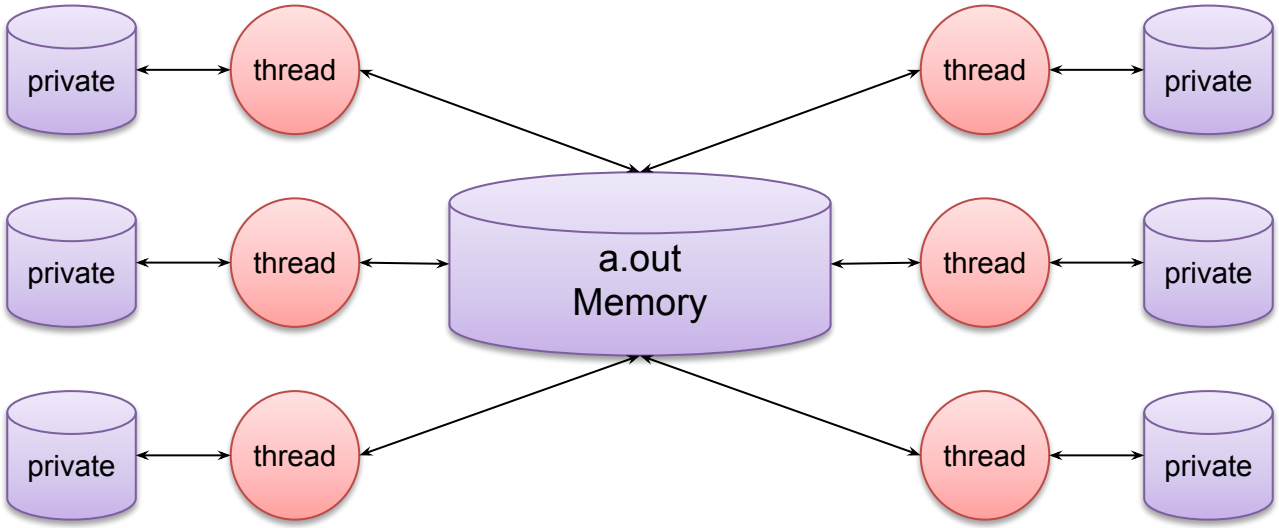
Modelo de memoria compartida (con hebras)

- En este caso, un proceso “pesado” puede tener múltiples vías “ligeras” de ejecución concurrentes
- **Ejemplo:**
  - Se planifica la ejecución del programa principal **a.out**. **a.out** carga todos los recursos necesarios para ejecutarse (es el proceso “pesado”)
  - **a.out** realiza trabajo serie, y entonces genera una serie de tareas que pueden ejecutarse concurrentemente



Modelo de memoria compartida (con hebras)

- **Ejemplo (cont):**
  - Cada hebra dispone de datos locales, pero también comparte los recursos de **a.out**. Cada hebra se beneficia de una visión global de la memoria porque comparte el espacio de **a.out**.
  - Las hebras se comunican entre sí a través de la memoria global. Esto requiere de sincronización.
  - Las hebras pueden ir o venir, pero **a.out** permanece presente hasta que la aplicación finaliza.



Implementación (1/2)

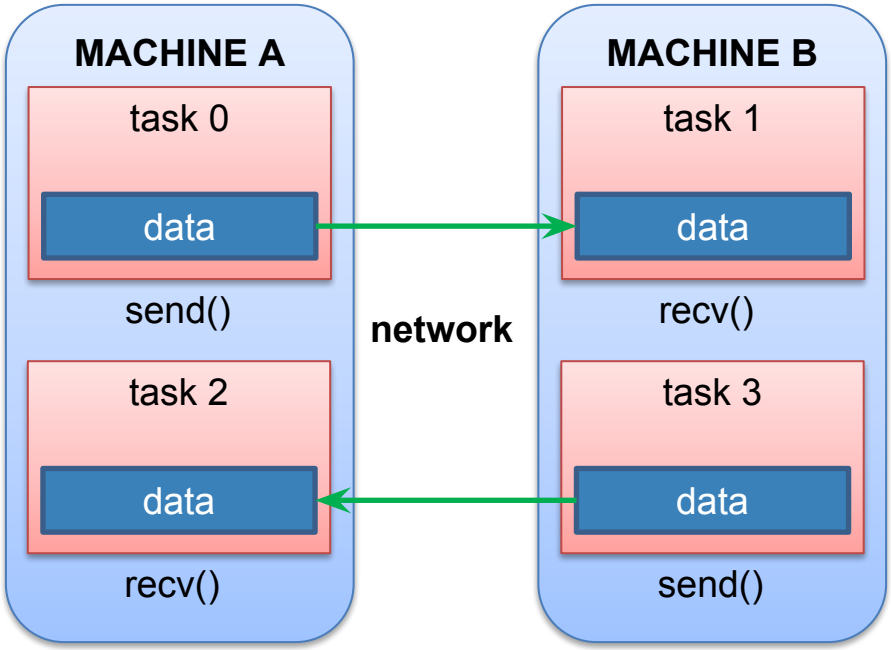
- Desde un punto de vista de programación, las hebras se pueden implementar:
  - Como funciones de biblioteca invocadas desde un código fuente paralelo
  - Como directivas de compilación, incorporadas en el código serie o paralelo
- Ciertos esfuerzos de estandarización han desembocado en implementaciones diferentes: **POSIX threads** y **OpenMP**

Implementación (2/2)

- POSIX Threads
  - IEEE POSIX 1003.1c standard (1995)
  - Parte de SO Unix/Linux
  - Basado en biblioteca
  - Disponible solo en C
- OpenMP
  - Estándar de la industria: englobe grupos de fabricantes hardware y distribuidores de software
  - Basado en directivas de compilador
  - Portable / multi-plataforma (Unix, Windows incluidos)
  - Disponible en C/C++ y Fortran
  - Sencillo de usar con paralelización incremental, pues se puede comenzar con el código serie
- Existen otras alternativas, como la implementación de Microsoft

Características

- Conjunto de tareas que usan su propia memoria local. Pueden residir en la misma máquina física o distribuidas entre varias máquinas
- Las tareas intercambian datos enviando y recibiendo mensajes
- Una operación de envío tiene que tener una operación de recepción

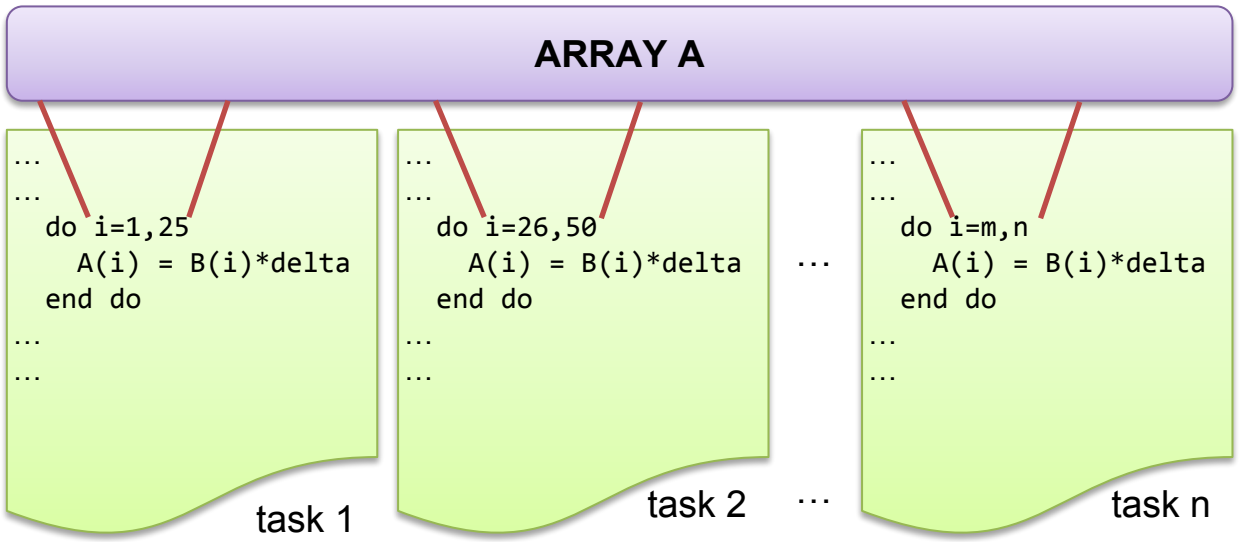


Implementación

- En 1992, se creó el foro MPI con el propósito de crear una interfaz estándar para la programación por paso de mensajes
- MPI-1 se liberó en 1994. MPI-2 en 1996 y MPI-3 en 2012
  - http://www.mpi-forum.org/docs
- MPI es el estándar “de facto” de la industria para el paso de mensajes

Características

- También se conoce como PGAS (Partitioned Global Address Space)
- Las tareas realizan la misma operación sobre una porción del dataset global
  - En memoria compartida, todas las tareas acceden al dataset a través de la memoria global
  - En memoria distribuida, cada tarea tiene una copia local de su porción del dataset

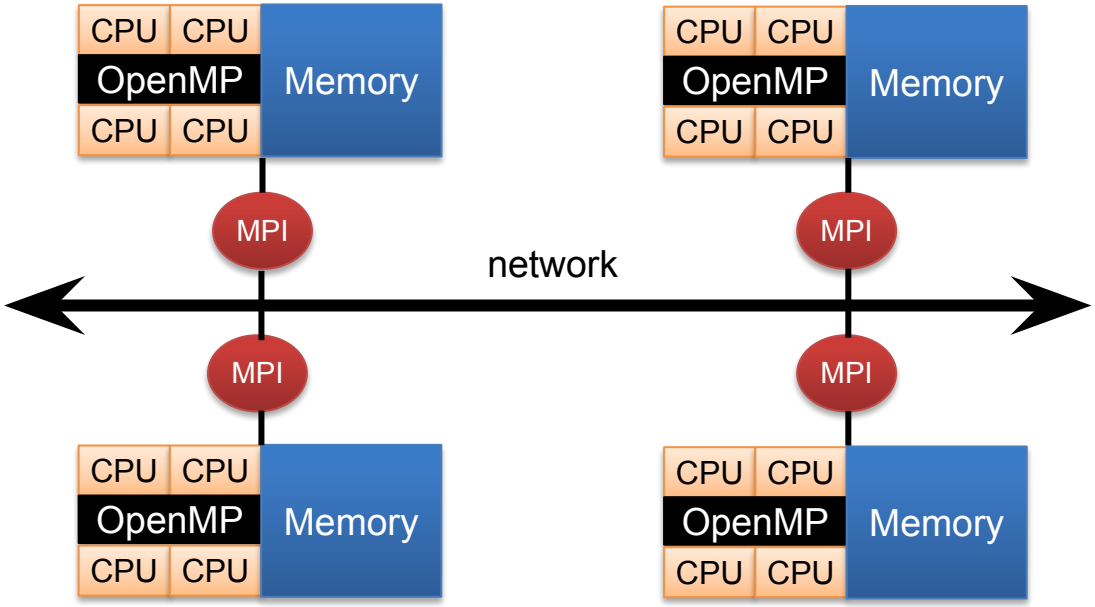


Implementaciones

- Coarray Fortran: extensiones de Fortran 95 para programación paralela en SPMD (ver más adelante)
- Unified Parallel C (UPC): Una extensión de C para SPMD
- Global Arrays: entorno de programación paralela, fundamentalmente para arrays. Bibliotecas C y Fortran77
- X10: Un lenguaje de programación paralelo en datos desarrollado por IBM
- Chapel: Desarrollado por Cray, un lenguaje de programación paralelo de código abierto

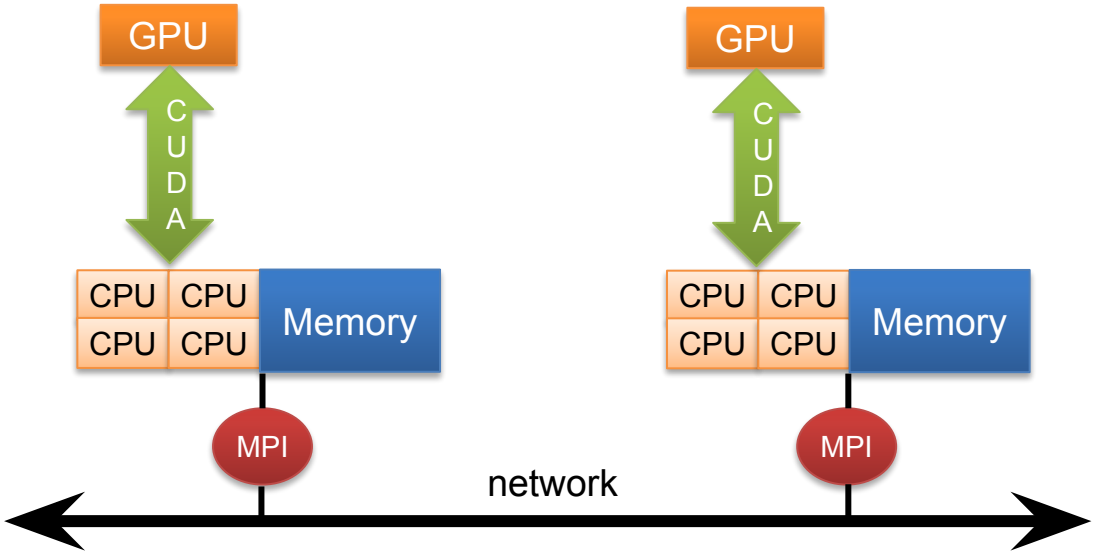
Características (1/2)

- El modelo híbrido combina más de uno de los modelos vistos previamente.
- Actualmente, uno de los modelos más usados combina MPI y OpenMP
  - Las hebras OpenMP se encargan de la computación más intensiva
  - La sincronización entre procesos se resuelve con MPI
- Este modelo encaja muy bien en la naturaleza heterogénea actual de un clúster



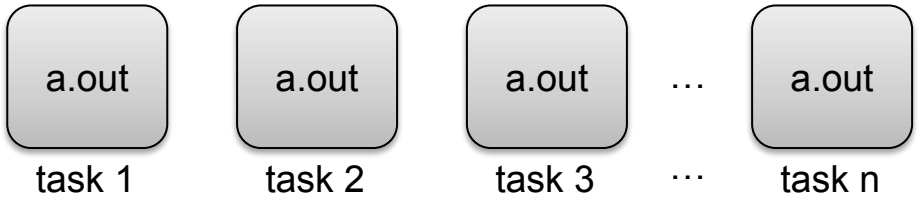
Características (2/2)

- Otro uso popular de este modelo consiste en combinar MPI con CPU-GPU
- Otros modelos híbridos:
  - MPI con pthreads
  - MPI con aceleradores no-GPU



SPMD

- SPMD: Single Program Multiple Data
  - Es un modelo de alto nivel que se puede construir con cualquier combinación de los modelos anteriores
  - Single Program: Todas las tareas ejecutan su copia del mismo programa simultáneamente (hebras, paso de mensajes, paralelo en datos o híbrido)
  - Multiple Data: Todas las tareas pueden usar diferentes datos
  - Los programas SPMD tienen lógica interna necesaria para seleccionar la porción de código con la que deben trabajar ➔ una tarea no la define el programa, sino una porción de él



MPMD

- MPMD: Multiple Program Multiple Data
  - También es un modelo de alto nivel que se puede construir con cualquier combinación de los modelos anteriores
  - Multiple Program: Las tareas pueden ejecutan programas diferentes de forma simultánea (hebras, paso de mensajes, paralelo en datos o híbrido)
  - Multiple Data: Todas las tareas pueden usar diferentes datos
  - Los programas MPMD no son tan comunes como los SPMD. Se ajustan mejor a una partición funcional (ya visto) del programa paralelo