Sistemas Concurrente y Distribuidos: Tema 3
10 de Octubre del 2020
A year to Forget

Daniel Monjas Miguélez

Índice

Mecanismos básicos en sistemas multiprocesadores: procesadores individuales que ejecutan sus instrucciones independientemente (modelo MIMD). Estos utilizan variables en memoria compartida a los procesadores. Su principal problema es la falta de escalabilidad (competición por recursos, colisiones, etc.). Se necesitan crossbar switches para acceder a posiciones de memoria externas a 1 procesador.

Multiplrocesamiento:
Utilización de varios procesadores o núcleos para ejecutar los programas de una misma aplicación. Desde el punto de vista dle sistema es la capacidad para gestionar más de 1 procesador y re-asignar tareas entre tales procesadores durante la ejecución de los programas.

Los procesadores pueden ejecutar 1 sola secuenci o varias secuencias de instrucciones, que se ejecutaráne n contextos múltiples y simultáneos.

- **SISD Single-Instruction Single-Data:** una instrucción se ejecuta sobre un solo conjunto de datos.
- **SIMD Single-Instruction Multiple-Data:** una instrucción se ejecuta sobre varios conjuntos de datos.
- **MISD Multiple-Instruction Single-Data:** se realizan varias intrucciones sobre un único conjunto de datos.
- **MIMD Multiple-Instruction Multiple-Data:** se realizan varias instrucciones sobre varios conjuntos de datos.

Mecanismos básicos en sistemas multicomputadores: en estos no existe memoria común, toda la comunicación y sincronización se realiza por medio de una red de interconexión. Son más difíciles de programar, aunque a diferencia de los multiprocesadores son escalabres hasta un número máximo de procesadores individuales. Las primitivas concurrentes clásicas que suponen memoria compartidas no se pueden utilizar.

Estilo de programación SPMD (Single-Program Multiple-Data): se utiliza programación distribuida y paralela, el código que ejecutan los procesos es idéntico pero tal código actúa sobre diferentes datos.

- Variante del modelo MIMD de la taxonomía de Flynn, que se aproxima al SIMD.
- Cone ste modelo los procesos se pueden ejecutar en procesadores de propósito general.

- Los procesadores ejecutan el mismo programa pero de forma independiente: el programa puede contener ramas condicionales que se ejecutan dependiendo del número de procesador.
- A cada proceso paralelo del programa se le asignará un valor distinto de id en una etapa de configuración posterior a la compilación de dicho programa.
- Cada proceso peude ejecutar una parte distinta del programa común, que seleccionará dependiendo del valor del identificador que se le haya asignado.

Semántica de las operaciones de paso de mensajes: Sinificado(semántica) de las operaciones de comunicación ({send(),receive()}) entre dos procesos depende de, el cumplimiento (o no) de la propiedad de seguridad y del modo de comunicación.

1. Propiedad de seguridad: si no se cumple se permite alterar un dato despué de que la operación de envío se ejecute y vuelva, pero antes de que termine la transmisión del valor enviado. Si se cumple el valor del dato enviado coincide con el del dato recibido posteriormente.
2. Modo de comunicación de las operaciones de paso de mensajes: estas pueden ser bloqueantes o no bloqueantes (con asincronicidad).

Operaciones bloqueantes de pasos de mensajes: la operación de envío solo vuelve cuando se garantice la propiedad de seguridad durante las transmisión de los datos enviados. Estas operaciones se pueden implementar con hardware especializado, con búfer y con sincronización o sin ellas.

Si se implementa sin búffer se denominaría como sincronización por citas, es decir, bloquea tanto al emisor como al receptor. Si admite capacidad de bufferización, pero no hardware especializado entonces se requerirá sincronización en la parte receptora. Si se dispusiese de hardware especializado la sincronización sería relajada.

Mecanismos de citas: Se trata de una operación de comunicación bloqueante y sin buffer. En este antes de que comience la transmisión de los datos, ambos procesos han de estar preparados para la comunicación. El estado del proceso emisor se mantiene hasta que vuelve la operación de recepción en el otro procesos. Los procesos receptor o emisor pueden sufrir espera ociosa que determina la mencionada cita.

Paso de mensajes bloqueantes con bufer: el proceso emisor del mensaje no se bloquea al ejecutar la operación send()... salvo que el bufer

se desborde. La operación receive(...) no vuelve hasta que dicha operación se completa.

Operaciones no-bloqueantes: el cumplimiento de la propiedad de seguridad durante el paso de mensajes se convierte en responsabilidad del programador.

- Las operacioens de envío se ejecutan y devuelven inmediatamente el control al programa (no suspenden), antes incluso de que sea seguro modificar los datos que están en transmisión.
- Existe operaciones de comprobación de estado de los datos transmitidos para determinar si pueden ser modificados o no en un momento dado de la ejecución del programa.
- La operación receive(...) vuelve (o no) antes de terminar la transmisión dependiendo de si existe hardware especializado de apoyo a esta operación de paso de mensajes.

Paso de mensajes no bloqueante: en este caso se reduce el tiempo de espera respecto del caso anterior porque la operación receive() provoca la transferencia inmediata de datos del búfer a la memoria propia del proceso receptor.

Bibliiotecas de paso de mensajes y patrones de interacción:
Modelo SPMD: Message Passing Interface (MPI). Tiene diferentes implementaciones (bingdings).

- **Órdenes para compilación:** mpicc, mpif77: versioes de las órdenes habituales.
- **Órdenes específicas para ejecución de aplicaciones paralelas:** mpirun.
- **Herramientas para monitorización y depuración de programas paralelos.**

No es la única biblioteca para la programación de aplicaciones paralelas y distribuidas, pero sí la más utilizada en la actualidad. Se dispone de funciones de comunicación punto-a-punto, así como operaciones colectivas para involucrar a un grupo de procesos. Los procesos pueden agruparse y fromar comunicadores para permitir la definición del ámbito de las operaciones colectivas y un diseño modular de los programas distribuidos.

Operaciones de paso de mensajes utilizando MPI:

Un mensaje de MPI es un bloque de datos transferido entre procesadores y consiste en: envoltorio de mensaje (compuesto de destino/orignen, etiqueta y comunicador), un cuerpo del mensaje (compuesto de un bufer, un contador y tipo de datos).

Semántica de las operacioens de paso de mensajes bloqueantes: La envoltura del mensajes envido ha de coincidir con la envoltura del mensajes recibido. Las operaciones: MPI_Send, MPI_Ssend, MPI_Recv se suspenden (no vuelven) hasta que se completan.

- MPI_Send: el mensaje se termina de copiar en el búfer RMI.
- MPI_Ssend: se produce la cita con el proceso que llama a la operación MPI_Recv y hay concordancia entre las envolturas en cada parte de la comunicación.
- MPI_Recv: existe ya un mensaje pendiente conforme a la declaración de parámetros de esta operación.

Situaciones de error: las situaciones más comunes son cuando el tamaño del mensaje recibido es mayor que el esperado (el programa aborta) y cuando los tipos de datos declarados en el emisor y en el receptor son incompatibles (resultado indefinido).

Sustitución de comodines: se puede sustituir MPI_ANY_SOURCE en el campo origen y MPI_ANY_TAG en el campo etiqueta sin que se produzca error en la operación de comunicación. Sin embargo, el campo comunicador carece de comodín.

Obtener información de los mensajes recibidos:

- estado/tamaño: MPI_Get_Count(status,t_datos,cont)
- estado: campo MPI_Status (parámetro de MPI_Recv)

Comunicación no bloqueante en MPI: La idea fundamental es evitar situaciones de interbloqueo de procesos debidar a una incorrecta secuenciación en el orden de ejecución de las operaciones de envío y recepción por parte de los procesos que intervienen en una comunicación. Impedir el bloqueo de los procesos emisores de mensajes debido al desbordamiento del búfer interno al recibir.

Operaciones de MPI de envío/recepción no bloqueantes

- MPI_Isend: inicia envío, pero vuelve de la llamada antes de comenzar a copiar el mensaje en el buffer.

- MPI_Irecv: inicia recepción pero vuelve de la llamada antes de comenzar a recibir ningún mensaje.
- MPI_Test (MPI_Request *r, int *flag, MPI_Status *s), chequea si la operación no bloqueante (identificada por el argumento r) ha finalizado → argumento flag ¿0

Sondeo de estado de un mensaje:
MPI_Iprobe (int origen, int etiqueta, MPI_Comm comunicador, int * flag, MPI_Status * estado). Si hay mensaje pendiente (flag ¿0), entonces hay que recibirlo con una llamada MPI_Recv. Comprobación no bloqueante.

MPI_Porbe (int origen, int etiqueta, MPI_Comm comunicador, MPI_Status * estado). Comprobación bloqueante. Esperar un mensaje sin conocer su procedencia, etiqueta o tamaño.

Esperar la completación de una operación:
MPI_Wait(MPI_Request * solicitud, MPI_Status * estado). Bloquea al proceso que la llama hasta que la operación identificada por 'solicitud' termina de forma segura.

MPI_Request_free (MPI_Request * solicitud). Libera el objeto 'solicitud' de forma explícita.

Modelos y lenguajes de programación distribuida: Programación de procesos servidores con paso de mensaje síncrono:

- Inadecuación de las órdenes condicionales deterministas (if, switch, ...) de los lenguajes secuenciales para implementar servidores.
- Sentencias no-deterministas en los lenguajes de programación facilitan la implementación de sistemas que reaccionan frente a estímulos procedentes de su entrono.

Paradigma cliente/servidor de programación distribuida: las ideas fundamentales son: comunicación muchos-a-uno, cada comunicación es un par (datos de entrada, resultados) y selección no-determinista entre varias comunicaciones posibles, en lado del servidor. Características:

- Proceso cliente: solicita un servicio enviando un mensaje servidor. Los procesos clientes tiene un carácter activo, ya que envían mensajes solicitando un servicio.
- Proceso servidor: tiene un carácter pasivo; recibe una petición de servicio de los clientes, devuelve un mensaje con los posibles resultados.