Introducción:

El hardware paralelo ha estado disponible durante décadas y se está volviendo cada vez más común. Sin embargo, el software paralelo que explota completamente el hardware es mucho más raro y en su mayoría se limita al área especializada de la supercomputación.

Creemos que parte de la razón de esta situación es que la mayoría de los entornos de programación paralela, que se centran en la implementación de la concurrencia en lugar de en cuestiones de diseño de alto nivel, son simplemente demasiado difíciles para que la mayoría de los programadores se arriesguen a usarlos.

Estamos involucrados en un esfuerzo continuo para diseñar un lenguaje de patrones para programas de aplicaciones paralelas. El objetivo del lenguaje de patrones es reducir la barrera de la programación paralela guiando al programador a través de todo el proceso de desarrollo de un programa paralelo. En nuestra visión del desarrollo de programas paralelos, el programador aporta al proceso una buena comprensión del problema real que se debe resolver, luego trabaja a través del lenguaje de patrones, obteniendo finalmente un diseño detallado o incluso un código funcional. El lenguaje de patrones está organizado en cuatro espacios de diseño, que se visitan en orden.

• *El espacio de diseño FindingConcurrency* incluye patrones de alto nivel que ayudan a encontrar la concurrencia en un problema y descomponerla en una colección de tareas.

• *El espacio de diseño AlgorithmStructure* contiene patrones que ayudan a encontrar una estructura de algoritmo adecuada para explotar la concurrencia que se ha identificado.

• *El espacio de diseño SupportingStructures* incluye patrones que describen tipos de datos abstractos útiles y otras estructuras de soporte.

• *El espacio de diseño ImplementationMechanisms* contiene patrones que describen problemas de implementación de nivel inferior.

Los dos últimos espacios de diseño (que amplían ligeramente la noción típica de un patrón) podrían incluso incluir bibliotecas de código reutilizables o marcos de trabajo. Usamos un formato de patrón para los cuatro niveles para poder abordar una variedad de cuestiones de una manera unificada. La versión actual, incompleta, del lenguaje de patrones se puede ver en http://www.cise.ufl.

edu/research/ParallelPatterns. Consiste en una colección de documentos con muchos hipervínculos, de modo que el diseñador puede comenzar en el nivel superior y trabajar a través del lenguaje de patrones siguiendo los vínculos.

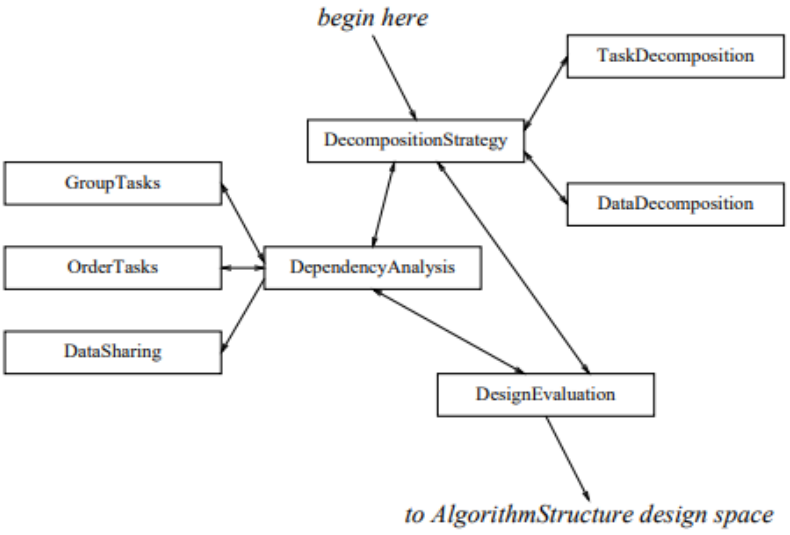
En este artículo, presentamos patrones del espacio de diseño FindingConcurrency. Estos patrones se usan al principio del proceso de diseño, después de que se haya analizado el problema utilizando técnicas de ingeniería de software estándar y se hayan comprendido las

estructuras de datos y los cálculos clave. Ayudan a los programadores a comprender cómo exponer la concurrencia explotable en sus problemas. Más específicamente, estos patrones ayudan al programador:

- Identificar las entidades en las que se descompondrá el problema.
- Determinar cómo las entidades dependen unas de otras.
- Construir un marco de coordinación para gestionar la ejecución paralela de las entidades.

Estos patrones colaboran estrechamente con los patrones AlgorithmStructure, y una de sus principales funciones es ayudar al programador a seleccionar un patrón apropiado en el espacio de diseño AlgorithmStructure. Los diseñadores experimentados pueden saber cómo hacer esto inmediatamente, en cuyo caso podrían pasar directamente a los patrones en el espacio de diseño AlgorithmStructure.

Organización del espacio de diseño de FindingConcurrency:



1.2 Estructura del espacio de diseño FindingConcurrency

Los patrones en este espacio de diseño están organizados como se ilustra en la Figura 1. La ruta principal a través de los patrones pasa por tres patrones principales:

• *DecompositionStrategy*: este patrón ayuda al programador a decidir si el problema debe descomponerse en función de una descomposición de datos, una descomposición de tareas o una combinación de las dos.

• *DependencyAnalysis*: una vez que se han identificado las entidades en las que se descompondrá el problema, este patrón ayuda al programador a entender cómo dependen entre sí.

• *DesignEvaluation*: este patrón es un patrón de consolidación. Se utiliza para evaluar los resultados de los otros patrones en este espacio de diseño y preparar al programador para el siguiente espacio de diseño, el espacio de diseño AlgorithmStructure.

Los patrones DecompositionStrategy y DependencyAnalysis se derivan de grupos de patrones que ayudan con la descomposición de problemas y el análisis de dependencias.
Usamos flechas de dos puntas para la mayoría de las rutas en la figura para indicar que es posible que sea necesario moverse de un patrón a otro repetidamente a medida que avanza el análisis. Por ejemplo, en un análisis de dependencia, el programador puede agrupar las tareas de una manera y luego determinar cómo esta agrupación afecta los datos que deben compartirse entre los grupos. Esta compartición puede implicar una forma diferente de agrupar las tareas, lo que lleva al programador a revisar la agrupación de tareas. En general, se puede esperar que trabajar con estos patrones sea un proceso iterativo.

2 El patrón DecompositionStrategy

Intento:

Este patrón aborda la pregunta "¿Cómo se descompone el problema en partes que se pueden ejecutar simultáneamente?"

Motivación:

Los programas paralelos permiten resolver problemas más grandes en menos tiempo. Lo hacen al resolver simultáneamente diferentes partes del problema en diferentes procesadores. Esto solo puede funcionar si el problema contiene concurrencia explotable, es decir, múltiples actividades o tareas que pueden realizarse al mismo tiempo.
Exponer la concurrencia requiere descomponer el problema en dos dimensiones diferentes:

• *Descomposición de tareas.*

Dividir el flujo de instrucciones en múltiples fragmentos llamados tareas que se pueden ejecutar simultáneamente. Para lograr un rendimiento de tiempo de ejecución razonable, las tareas deben ejecutarse con una necesidad mínima de interacción; es decir, la sobrecarga asociada con la gestión de dependencias debe ser pequeña en comparación con el tiempo de ejecución total del programa.

• *Descomposición de datos*.

Determinar cómo interactúan los datos con las tareas. Algunos de los datos se modificarán sólo dentro de una tarea; es decir, son locales para cada tarea. Para estos datos, el diseñador del algoritmo debe descubrir cómo dividirlos y asociarlos adecuadamente con las tareas correctas. Otros datos son modificados por múltiples tareas; es decir, los datos son globales o compartidos entre tareas. Para los datos compartidos, el objetivo es diseñar el

algoritmo de modo que las tareas no se interpongan entre sí mientras trabajan con los datos.

El equilibrio de las fuerzas opuestas de la descomposición de los datos y las tareas se produce en el contexto de dos factores adicionales: eficiencia y flexibilidad. El programa final debe utilizar eficazmente los recursos proporcionados por la computadora paralela. Al mismo tiempo, las computadoras paralelas vienen en una variedad de arquitecturas y necesita suficiente flexibilidad para manejar todas las computadoras paralelas que le interesan.

Tenga en cuenta que en algunos casos la descomposición adecuada será obvia; en otros necesitará profundizar en el problema para exponer la concurrencia. A veces, puede que incluso necesite reformular por completo el problema y reestructurar su forma de pensar sobre su solución.

Aplicabilidad

Use este patrón cuando:

- Ha determinado que su problema es lo suficientemente grande y significativo como para que valga la pena dedicar el esfuerzo de crear un programa paralelo.
- Entiende las estructuras de datos clave del problema y cómo se utilizan.
- Entiende qué partes del problema requieren más cómputo. Es en estas partes en las que concentrará sus esfuerzos.

Implementación:

El objetivo es descomponer su problema en entidades relativamente independientes que puedan ejecutarse simultáneamente. Como se mencionó anteriormente, hay dos dimensiones a considerar:

• **La dimensión de descomposición de tareas** se centra en las operaciones que se llevarán a cabo dentro de entidades que se ejecutan simultáneamente. Nos referimos a un conjunto de operaciones que están agrupadas lógicamente como una tarea. Para que una tarea sea útil, las operaciones que la componen deben ser en gran medida independientes de las operaciones que se llevan a cabo dentro de otras tareas.

• **La dimensión de descomposición de datos** se centra en los datos. Debe descomponer los datos del problema en fragmentos que se puedan operar de manera relativamente independiente.

Si bien la descomposición debe abordar tanto las tareas como los datos, la naturaleza del problema generalmente (pero no siempre) sugiere una descomposición u otra como la descomposición principal, y es más fácil comenzar con esa.

• Una **descomposición basada en datos** es un buen punto de partida si:

– La parte del problema que requiere más recursos computacionales manipula una gran estructura de datos.

– Se aplican operaciones similares a diferentes partes de la estructura de datos,

de tal manera que las diferentes partes pueden operar de manera relativamente independiente.

Por ejemplo, muchos problemas pueden formularse en términos de la multiplicación de grandes matrices. Matemáticamente, cada elemento de la matriz producto se calcula utilizando el mismo conjunto de operaciones. Esto sugiere que una forma efectiva de pensar en este problema es en términos de la descomposición de las matrices. Hablamos de este enfoque con más detalle en el patrón DataDecomposition.

Las descomposiciones basadas en datos tienden a ser más escalables (es decir, su rendimiento se escala con la cantidad de elementos de procesamiento), ya que se descompone la memoria.

• Una **descomposición basada en tareas** es un buen punto de partida si:

– Es natural pensar en el problema en términos de una colección de tareas independientes (o casi independientes).

Por ejemplo, muchos problemas pueden considerarse en términos de una función que se evalúa repetidamente, con un conjunto de condiciones ligeramente diferente cada vez. Podemos asociar cada evaluación de función con una tarea. Si este es el caso, debe comenzar con una descomposición basada en tareas, que es el tema del patrón TaskDecomposition.

.

Si hay muchas tareas casi independientes, las descomposiciones basadas en tareas tienden a producir un diseño con mucha flexibilidad, lo que es una ventaja al decidir más adelante cómo asignar tareas a los elementos de procesamiento.

En algunos casos, puede ver el problema de ambas formas. Por ejemplo, anteriormente describimos una descomposición de datos de un problema de multiplicación de matrices. También puede verlo como una descomposición basada en tareas, por ejemplo, asociando la actualización de cada columna de la matriz con una tarea. En los casos en los que no se puede tomar una decisión clara, el mejor enfoque es probar cada descomposición y ver cuál es más eficaz para exponer mucha concurrencia.

Durante el proceso de diseño, también debe tener en cuenta las siguientes fuerzas en competencia:

• Flexibilidad

¿Su diseño es lo suficientemente abstracto como para tener la flexibilidad suficiente para adaptarse a diferentes requisitos de implementación? Por ejemplo, no desea limitar sus opciones a un solo sistema informático o estilo de programación en esta etapa del diseño.

• Eficiencia

Un programa paralelo solo es útil si escala de manera eficiente con el tamaño de la computadora paralela (en términos de tiempo de ejecución reducido y/o utilización de memoria).

Para la descomposición del problema, esto significa que necesita suficientes tareas para mantener todos los elementos de procesamiento (PE) ocupados con suficiente trabajo por tarea para compensar la sobrecarga incurrida para administrar las dependencias. La búsqueda de eficiencia puede conducir a descomposiciones complejas que carecen de flexibilidad.

• Simplicidad

La descomposición debe ser lo suficientemente compleja para realizar el trabajo,pero lo suficientemente simple para permitirle depurar y mantener su programa en un tiempo razonable.

Equilibrar estas fuerzas en competencia a medida que descompone su problema es difícil y con toda probabilidad no lo hará bien la primera vez.

Por lo tanto, utilice una estrategia de descomposición iterativa en la que descomponga por el método más obvio (tarea o datos) y luego por el otro método (datos o tarea).

Ejemplos:

Imágenes médicas.

Aquí definiremos un único problema, tomado del campo de las imágenes médicas, y luego lo descomponemos de dos maneras diferentes: en términos de tareas y en términos de datos. Las descomposiciones se presentarán en los patrones DataDecomposition y TaskDecomposition; la discusión aquí servirá para definir el problema y luego describir la manera en que las dos soluciones interactúan.

Una herramienta de diagnóstico importante es administrarle a un paciente una sustancia radiactiva y luego observar cómo esa sustancia se propaga a través del cuerpo observando la distribución de la radiación emitida. Desafortunadamente, las imágenes son de baja resolución, debido en parte a la dispersión de la radiación a medida que pasa a través del

cuerpo. También es difícil razonar a partir de las intensidades absolutas de radiación, ya que las diferentes vías a través del cuerpo atenúan la radiación de manera diferente.

Para resolver este problema, los especialistas en imágenes médicas construyen modelos de cómo la radiación se propaga a través del cuerpo y usan estos modelos para corregir las imágenes. Un enfoque común es construir un modelo de Monte Carlo. Se supone que puntos seleccionados al azar dentro del cuerpo emiten radiación (normalmente un rayo gamma) y se sigue la trayectoria de cada rayo. A medida que una partícula (rayo) pasa a través del cuerpo, se atenúa por los diferentes órganos que atraviesa, y continúa hasta que la partícula sale del cuerpo y golpea un modelo de cámara, definiendo así una trayectoria completa. Para crear una simulación estadísticamente significativa, se siguen miles, si no millones, de trayectorias.

El problema se puede paralelizar de dos maneras. Dado que cada trayectoria es independiente, sería posible paralelizar la aplicación asociando cada trayectoria con una tarea. Este enfoque se analiza en la sección “Ejemplos” del patrón TaskDecomposition.

Otro enfoque sería dividir el cuerpo en secciones y asignar diferentes secciones a diferentes elementos de procesamiento. Este enfoque se analiza en la sección “Ejemplos” del patrón DataDecomposition.

Como en muchos códigos de trazado de rayos, no hay dependencias entre trayectorias, lo que hace que la descomposición basada en tareas sea la opción natural. Al eliminar la necesidad de gestionar dependencias, el algoritmo basado en tareas también ofrece al programador mucha flexibilidad más adelante en el proceso de diseño, cuando la forma de programar el trabajo en diferentes elementos de procesamiento se vuelve importante.

Sin embargo, la descomposición de datos es mucho más eficaz para gestionar el uso de la memoria. Esto suele suceder con una descomposición de datos en comparación con una descomposición de tareas.

Como la memoria se descompone, los algoritmos de descomposición de datos también tienden a ser más escalables. Estas cuestiones son importantes y apuntan a la necesidad de al menos considerar los tipos de plataformas que serán compatibles con el programa final. La necesidad de portabilidad lleva a uno a tomar decisiones sobre las plataformas de destino lo más tarde posible.

Sin embargo, hay momentos en los que retrasar la consideración de cuestiones dependientes de la plataforma puede llevar a uno a elegir un algoritmo deficiente.

Base de datos paralela.

Como otro ejemplo de un problema único que se puede descomponer de múltiples maneras, consideremos una base de datos paralela. Un enfoque es dividir la base de datos en múltiples fragmentos. Múltiples procesos de trabajo manejarían las operaciones de

búsqueda reales, cada uno en el fragmento de la base de datos que "posee" y un solo administrador recibiría las solicitudes de búsqueda y las reenviaría al trabajador relevante para realizar la búsqueda.
Un segundo enfoque para este problema de base de datos paralela también utilizaría un administrador y múltiples trabajadores, pero mantendría la base de datos intacta en una ubicación lógica. Los trabajadores serían esencialmente idénticos y cada uno podría trabajar en cualquier parte de la base de datos. Observe que los problemas planteados en este ejemplo son similares a los del ejemplo de imágenes médicas.

Iterative algorithms

Muchos problemas de álgebra lineal pueden resolverse aplicando repetidamente alguna operación a una matriz grande u otro arreglo. Las paralelizaciones efectivas de estos algoritmos suelen basarse en paralelizar cada iteración (en lugar de, por ejemplo, intentar realizar las iteraciones simultáneamente). Por ejemplo, consideremos un algoritmo que resuelve un sistema de ecuaciones lineales Ax=bAx = bAx=b (donde AAA es una matriz y xxx y bbb son vectores) calculando una secuencia de aproximaciones x(0),x(1),x(2)x^{(0)}, x^{(1)}, x^{(2)}x(0),x(1),x(2), y así sucesivamente, donde para alguna función fff, x(k+1)=f(x(k))x^{(k+1)} = f(x^{(k)})x(k+1)=f(x(k)).

Una paralelización típica se estructuraría como una iteración secuencial (calculando los x(k)x^{(k)}x(k) en secuencia), con cada iteración (calculando x(k+1)=f(x(k))x^{(k+1)} = f(x^{(k)})x(k+1)=f(x(k)) para algún valor de kkk) siendo computada de una manera que explote la concurrencia potencial. Por ejemplo, si cada iteración requiere una multiplicación de matrices, esta operación puede paralelizar utilizando una descomposición basada en tareas (como se discute en la sección "Ejemplos" del patrón **TaskDecomposition**) o una descomposición basada en datos (como se describe en la sección "Ejemplos" del patrón **DataDecomposition**).

3 El patrón TaskDecomposition

Intento:

Este patrón aborda la pregunta “¿Cómo se descompone un problema en tareas que puedan ejecutarse simultáneamente?”

También conocido como:

• Descomposición funcional.

• Paralelismo de tareas.

Motivación:

Este patrón aborda los problemas que surgen durante una descomposición basada principalmente en tareas.

La clave para una descomposición de tareas eficaz es garantizar que las tareas sean lo suficientemente independientes como para que mantener las dependencias ocupe solo una pequeña fracción del tiempo de ejecución general del programa. También es importante garantizar que la ejecución de las tareas se pueda compartir equitativamente entre el conjunto de elementos de procesamiento (el llamado problema de equilibrio de carga).

Aplicabilidad:

Use este patrón cuando

• Ha revisado el patrón DecompositionStrategy y ha decidido probar una descomposición basada en tareas de su problema.

Implementación:

Es muy raro que una descomposición basada en tareas pueda llevarse a cabo automáticamente. Debe hacerlo a mano basándose en su conocimiento del problema y el código necesario para implementarlo.

En una descomposición basada en tareas, observa su problema como una colección de tareas distintas, prestando especial atención a

• Las acciones que se llevan a cabo para resolver su problema.

• Si estas acciones son distintas y relativamente independientes.

Como primer paso, intente identificar tantas tareas como sea posible; es mucho más fácil comenzar con demasiadas tareas y fusionarlas más tarde que comenzar con muy pocas tareas y luego intentar dividirlas. La clave es que las tareas individuales se ejecuten en su mayor parte de manera independiente unas de otras.

Puede encontrar tareas en muchos lugares diferentes:

• En algunos casos, cada tarea corresponde a una llamada distinta a una función en su programa. Estas llamadas de función se pueden asociar con las tareas, lo que lleva a lo que a veces se denomina una "descomposición funcional".

• Otro lugar para encontrar tareas es en iteraciones distintas dentro de un algoritmo. Si las iteraciones son independientes y hay suficientes, entonces se asignan muy bien a las tareas en una descomposición basada en tareas. Este estilo de descomposición basada en tareas conduce a lo que a veces se denomina algoritmos de "división de bucle".

• Las tareas también desempeñan un papel clave en las descomposiciones basadas en datos. En este caso, se descompone una gran estructura de datos y varias unidades de ejecución actualizan simultáneamente diferentes fragmentos de la estructura de datos. En este caso, las tareas son esas actualizaciones en fragmentos individuales.

Esta es solo una lista parcial de dónde puede encontrar tareas. Tenga en cuenta las fuerzas en competencia mencionadas en el patrón DecompositionStrategy.

Flexibilidad

El diseño debe ser flexible en cuanto a la cantidad de tareas generadas.

Esto permitirá que el diseño se adapte a una amplia gama de computadoras paralelas. Se obtiene una flexibilidad

adicional si la definición de las tareas es independiente de cómo se programan para su ejecución.

Eficiencia

Hay dos cuestiones de eficiencia importantes que se deben tener en cuenta en la descomposición de tareas. En primer lugar, cada tarea debe incluir suficiente trabajo para compensar la sobrecarga incurrida al crear las tareas y administrar sus dependencias. En segundo lugar, la cantidad de tareas debe ser lo suficientemente grande como para que todas las unidades de ejecución estén ocupadas con trabajo útil durante todo el cómputo.

Simplicidad

Defina tareas para simplificar la depuración y el mantenimiento. Cuando sea posible, defina tareas de manera que reutilicen el código de programas secuenciales existentes que resuelvan problemas relacionados.

Una vez que tenga las tareas, debe observar la descomposición de datos implícita en las tareas. El patrón DataDecomposition puede ayudar con este análisis.

Ejemplos:

Imágenes médicas.

Consideremos el problema de imágenes médicas descrito anteriormente (en la sección “Ejemplos” del patrón DecompositionStrategy). En esta aplicación, se selecciona aleatoriamente un punto dentro de un modelo del cuerpo, se permite que ocurra una desintegración radiactiva en ese punto y se sigue la trayectoria de la partícula emitida. Para crear una simulación estadísticamente significativa, se siguen miles, si no millones, de trayectorias.

Es natural asociar cada trayectoria con una tarea. Estas tareas son particularmente simples de gestionar simultáneamente, ya que son completamente independientes. Otro punto importante es asegurarse de que haya suficientes para que podamos admitir una variedad de sistemas informáticos, desde aquellos con una pequeña cantidad de elementos de procesamiento hasta computadoras masivamente paralelas.

Con las tareas básicas en la mano, ahora podemos considerar la descomposición de datos correspondiente y definir qué partes del espacio del problema deben asociarse con cada tarea. En este caso, cada tarea debe contener la información que define la trayectoria en cualquier punto, pero eso es todo. El problema más complicado es el modelo del cuerpo. No se puede deducir esto a partir de la descripción del problema que hemos dado, pero resulta que el modelo del cuerpo puede ser enorme. Dado que es un modelo de solo lectura, no hay problema si hay un sistema de memoria compartida eficaz, ya que cada tarea puede leer datos según sea necesario. Sin embargo, si el sistema de destino utiliza memoria distribuida, puede ser necesario replicar el modelo del cuerpo en cada elemento de procesamiento. Esto puede consumir mucho tiempo y puede desperdiciar una gran cantidad de memoria. Por lo tanto, para tales sistemas de destino, una descomposición basada en datos, descrita en la sección "Ejemplos" del patrón DataDecomposition, puede funcionar mejor.

Multiplicación de matrices.

Considere la multiplicación estándar de dos matrices ($C = A \cdot B$). Podemos producir una descomposición basada en tareas de este problema considerando el cálculo de cada elemento de la matriz del producto como una tarea separada. Cada tarea necesita acceso a una fila de A y una columna de B. Esta descomposición tiene la ventaja de que todas las tareas son independientes y, dado que todos los datos que se comparten entre las tareas (A y B) son de solo lectura, probablemente será fácil de implementar en un entorno de memoria compartida.

Sin embargo, en un entorno de memoria distribuida, el requisito de que cada tarea tenga acceso a una fila de A y una columna de B puede generar un uso excesivo de la memoria y/o comunicación, por lo que esta descomposición puede no ser efectiva. Consulte la sección “Ejemplos” del patrón DataDecomposition para obtener una discusión de otras descomposiciones más adecuadas para entornos de memoria distribuida. (De hecho, una descomposición basada en datos puede ser más efectiva para la mayoría de las plataformas actuales; la descomposición basada en tareas anterior funciona bien solo si el acceso a los elementos de la memoria es aproximadamente uniforme, lo que no es el caso de las computadoras basadas en caché).

Usos conocidos

Las descomposiciones basadas en tareas se utilizan en muchas aplicaciones. El código de geometría de distancia
(DGEOM) descrito en [Mattson96][2] utiliza una descomposición basada en tareas.

4 El patrón DataDecomposition

Intento

Este patrón aborda la pregunta “¿Cómo se descomponen los datos de un problema en unidades que se puedan operar de manera relativamente independiente?”

También conocido como:

• Paralelismo de datos.

Motivación

Este patrón analiza los problemas involucrados en la descomposición de datos en unidades que se puedan actualizar simultáneamente.

Por ejemplo, la mayoría de los problemas de álgebra lineal actualizan matrices grandes, aplicando un conjunto similar de operaciones a cada elemento de la matriz. En estos casos, es sencillo impulsar el diseño del algoritmo paralelo observando cómo se puede dividir la matriz en bloques que se actualizan simultáneamente. Las definiciones de tareas luego se derivan de cómo se definen los bloques y se asignan a los elementos de procesamiento de la computadora paralela.

Aplicabilidad

Use este patrón cuando

• Ha revisado el patrón DecompositionStrategy y decidió probar una descomposición de su problema en base de datos.

Implementación

Los compiladores son buenos para analizar las dependencias de datos y, en algunos casos, pueden deducir automáticamente una descomposición de datos. En la mayoría de los casos, sin embargo, debe realizar la descomposición a mano.

Si ya ha realizado una descomposición basada en tareas, la descomposición de datos está impulsada por las necesidades de cada tarea. Si se pueden asociar datos bien definidos y distintos con cada tarea, la descomposición debería ser sencilla.

Sin embargo, si está comenzando con una descomposición de datos, debe observar no las tareas sino las estructuras de datos centrales que definen su problema, considerando si se pueden dividir en fragmentos que se puedan operar simultáneamente. Algunos ejemplos comunes son:

• **Cálculos basados en matrices**: la concurrencia se puede definir en términos de actualizaciones de

diferentes segmentos de la matriz. Si la matriz es multidimensional, observe que se puede descomponer de diversas maneras (filas, columnas o bloques de formas variables).

• **Estructuras de datos recursivas**: podemos pensar, por ejemplo, en descomponer la actualización paralela de una gran estructura de datos en forma de árbol descomponiendo la estructura de datos en subárboles que se puedan actualizar simultáneamente.

Independientemente de la naturaleza de la estructura de datos subyacente, la descomposición de los datos sirve como principio organizador de su algoritmo paralelo.

Al considerar cómo descomponer las estructuras de datos del problema, tenga en cuenta las fuerzas en competencia mencionadas en el patrón DecompositionStrategy:

• **Flexibilidad**. El tamaño y la cantidad de fragmentos de datos deben ser flexibles para admitir la gama más amplia de sistemas paralelos. Considere parametrizar la descomposición para que pueda ajustarse sin problemas a la granularidad de la mayoría de las computadoras paralelas.

Si es posible, debe definir fragmentos cuyo tamaño y cantidad estén controlados por una pequeña cantidad de parámetros. Estos parámetros definen los llamados "perillas de granularidad" que puede variar para modificar la descomposición de datos para que coincida con las necesidades del hardware subyacente. (Tenga en cuenta, sin embargo, que muchos diseños no son infinitamente adaptables con respecto a la granularidad).

El lugar más fácil para ver el impacto de la granularidad en la descomposición de datos es en la sobrecarga requerida para administrar dependencias entre fragmentos. El tiempo requerido para administrar dependencias debe ser pequeño en comparación con el tiempo de ejecución general.

En una buena descomposición de datos, las dependencias escalan en una dimensión menor que el esfuerzo computacional asociado con cada fragmento. Por ejemplo, en muchos códigos de diferencia finita, la mitad del ancho de la plantilla de diferencia finita define una región de celdas a lo largo de la superficie de cada fragmento descompuesto. Esta región de superficie define la dependencia entre fragmentos. El tamaño del conjunto de celdas dependientes escala como el área de superficie, mientras que el esfuerzo requerido

en el cálculo escala como el volumen del fragmento. Esto significa que puede escalar el esfuerzo computacional (basado en el volumen del fragmento) para compensar las sobrecargas asociadas con las dependencias de datos
(basadas en el área de superficie del fragmento).

• **Eficiencia**. Debe asegurarse de que los fragmentos sean lo suficientemente grandes para que la cantidad de trabajo para actualizar el fragmento compense la sobrecarga de la gestión de dependencias. Un problema más sutil a considerar es cómo se asignan los fragmentos a las unidades de ejecución.

Un algoritmo paralelo eficaz debe equilibrar la carga entre las unidades de ejecución. Si esto no se hace bien, por ejemplo, varios elementos de procesamiento en la computadora paralela pueden terminar su trabajo antes que los demás, y la escalabilidad general se verá afectada. Esto puede requerir formas inteligentes de dividir el problema.

Por ejemplo, si el problema borra las columnas de una matriz de izquierda a derecha, un mapeo de columnas de la matriz causará problemas ya que las unidades de ejecución con las columnas más a la izquierda terminarán su trabajo antes que las demás. Una descomposición en bloques basada en filas o incluso una descomposición cíclica en bloques (en la que las filas se asignan cíclicamente a los elementos de procesamiento) haría un trabajo mucho mejor para mantener todos los procesadores completamente ocupados.

• **Simplicidad.** Puede parecer obvio, pero muchos programadores han perdido incontables horas intentando depurar descomposiciones de datos demasiado complejas.

Ejemplos:

Imágenes médicas.

Considere el problema de imágenes médicas descritas anteriormente (en la sección “Ejemplos” del patrón DecompositionStrategy). En esta aplicación, se selecciona aleatoriamente un punto dentro de un modelo del cuerpo, se permite que ocurra una desintegración radiactiva en ese punto y se sigue la trayectoria de la partícula emitida. Para crear una simulación estadísticamente significativa, se siguen millas, si no millones, de trayectorias.

Una división basada en tareas es una opción natural para este problema. Sin embargo, las limitaciones de memoria han motivado el desarrollo de descomposiciones basadas en datos para este problema. Cuando se distribuye la memoria del hardware paralelamente subyacente, resulta ventajoso evitar replicar el enorme modelo del cuerpo en cada elemento de procesamiento.

En un análisis basado en datos, el modelo del cuerpo es la gran estructura de datos central.

alrededor de la cual se puede organizar el cálculo. El modelo se divide en segmentos y uno o más segmentos se asocian con cada elemento de procesamiento. Los segmentos del cuerpo solo se leen, no se escriben, durante los cálculos de la trayectoria, por lo que no hay dependencias de datos creadas por la división del modelo del cuerpo.

Una vez que se han descompuesto los datos, debe observar las tareas asociadas con cada segmento de datos. En este caso, cada trayectoria que pasa por el segmento de datos definir una tarea. Las trayectorias se inician y propagan dentro de un segmento exactamente como en el enfoque basado en tareas. La diferencia ocurre cuando se encuentra un límite de segmento. Cuando esto sucede, la trayectoria debe pasar entre segmentos. es esta transferencia la que define las dependencias entre los fragmentos de datos.

Observe que este algoritmo es más complejo que uno basado en un cálculo basado en tareas. Se puede requerir un esfuerzo considerable para implementar la contabilidad necesaria para realizar un seguimiento del conjunto de trayectorias a medida que se mueven a través del modelo del cuerpo.

Multiplicación de matrices.

Considere la multiplicación estándar de dos matrices (C = A · B). En la sección “Ejemplos” del patrón TaskDecomposition analizamos un análisis basado en tareas adecuado para entornos de memoria compartida, pero menos para entornos de memoria distribuida. Para este problema, son posibles varias descomposiciones basadas en datos. Una sencilla sería asignar una fila de la matriz de producto C a cada elemento de procesamiento.

De acuerdo con la definición de multiplicación de matrices, eso significa que cada elemento de procesamiento necesitaría la matriz A completa, pero solo la fila correspondiente de B. Con una revisión de datos de este tipo, la tarea básica de nuestro algoritmo se convierte en el cálculo de una fila de C.

Sin embargo, esto aún requiere la replicación de demasiados datos (la matriz A completa), por lo que podríamos refinar nuestro algoritmo para descomponer las tres matrices en bloques. La tarea básica se convierte entonces en la actualización de un bloque C, con los bloques A y B ciclados entre los nudos a medida que avanza el cálculo. El resultado es la descomposición basada en datos analizados como un ejemplo del patrón GeométricaDecomposición aunque tal división en bloques es más compleja, es sin embargo el enfoque utilizado en la práctica, ya que es el más eficiente.

Usos conocidos

Las descomposiciones de datos son muy comunes en la computación científica paralela. La biblioteca de álgebra lineal paralela ScaLAPACK es un buen ejemplo.