INSTITUTO TECNOLOGICO DE IZTAPALAPA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PROYECTO FINAL POLLY

PRESENTAN:

RUEDA IMÁN SOFIA VIRIDIANA
CELISEO GOMEZ ADAN GAMALIEL
MEJIA RAMOS LUIS ENRIQUE
QUINTERO BOLIO ERIK EDUARDO

NO. DE CONTROL:

171080043
171080044
161080183
171080151

ASESOR INTERNO:

PARRA HERNANDEZ ABIEL TOMAS

CIUDAD DE MÉXICO **ENERO/2021**

Resumen general de su proyecto final .

Polly es el marco de optimización y optimización de bucles y localización de datos avanzados de LLVM. Utiliza notación matemática abstracta basada en un método poliédrico entero para analizar y optimizar el patrón de acceso a la memoria del programa. Actualmente realizamos transformaciones de bucle clásicas, especialmente fusión de mosaico y bucle, para mejorar la ubicación de los datos.

Polly también puede aprovechar el paralelismo de nivel OpenMP para exponer oportunidades para SIMD. También se ha completado el trabajo en la generación automática de código de GPU.

Sin embargo, para muchos usuarios, lo más interesante no son las optimizaciones existentes en Polly, sino los nuevos análisis y optimizaciones habilitados por el framework Polly.

Cronograma preliminar de actividades.

INSTITUTO TECNOLOGICO IZTAPALAPA

CRONOGRAMA DE PROYECTO FINAL POLLY

INSTITUTO:	INSTITUTO TECNOLOGICO IZTAPALAPA
ALUMNO:	RUEDA IMAN SOFIA VIRIDIANA NUMERO DE CONTROL 171080043
ALUMNO:	CELISEO GOMEZ ADAN GAMALIEL NUMERO DE CONTROL 171080044
ALUMNO:	MEJIA RAMOS LUIS ENRIQUE NUMERO DE CONTROL 161080183
ALUMNO:	QUINTERO BOLIO ERICK EDUARDO NUMERO DE CONTROL 171080151
CARRERA:	INGENIERIA EN SISTEMAS COMPUTACIONALES
NOMBRE DEL PROYECTO:	POLLY
DOCENTE:	PARRA HERNANDEZ ABIEL TOMAS
FECHA DE INICIO:	SEPTIEMBRE 2020 FECHA DE ENTREGA: ENERO 2021

ACTIVIDAD	NUMERO	SEMANAS															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Resumen	1	■	■														
Cronograma	2			■	■	■											
Análisis de riesgo	3					■	■										
Justificación	4							■	■								
Metodología	5									■	■						
Requerimientos	6											■	■	■			
Diseño y desarrollo	7													■	■	■	■

ENTREGA DE REPORTES	DOCENTE:	PARRA HERNANDEZ ABIEL TOMAS
	ESTUDIANTE:	RUEDA IMAN SOFIA VIRIDIANA NUMERO :171080043
		CELISEO GOMEZ ADAN GAMALIEL NUMERO: 171080044
		MEJIA RAMOS LUIS ENRIQUE NUMERO: 161080183
		QUINTERO BOLIO ERICK EDUARDO NUMERO:171080151

Descripción del subproyecto de la infraestructura LLVM (POLLY)

Polly detecta y optimiza automáticamente la multiplicación de matrices generalizada, el cálculo C ← α ⊗ C ⊕ β ⊗ A ⊗ B, donde A, B y C son tres matrices de tamaño apropiado, las operaciones ⊕ y ⊗ se originan en el semiring de matriz correspondiente, y α y β son constantes y beta no es igual a cero. Permite obtener la forma altamente optimizada estructurada similar a la implementación experta de GEMM que se puede encontrar en GotoBLAS y sus sucesores.

Polly ahora puede optimizar automáticamente todos los núcleos polybench 2.0 sin la ayuda de un optimizador externo. El tiempo de compilación es razonable y podemos mostrar notables aceleraciones para varios núcleos.

Riesgos	Soluciones propuestas
Un hardware secuencial	Se crea un software que optimice la información local
No uso óptimo de hardware paralelo	Se usa polly para llamar OpenMP runtime library con el cual se puede utilizar todo el hardware de forma adecuada
Truncamientos en los procesos de polly	Se utiliza a polly para desarrollar un archivo bugpint que atrape este tipo de errores estos tienen la extensión ll
Polly es incapaz de reconocer un bugpoint, clang o opt	Se requiere una configuración de las herramientas para antes de la compilación

Polly - Polyhedral optimization in LLVM

Con Polly, estamos desarrollando una infraestructura poliédrica de última generación para LLVM, que admite la transformación completamente automática de los programas existentes. Polly detecta y extrae regiones de código relevantes sin ninguna interacción humana. Dado que Polly acepta LLVM-IR como entrada, es independiente del lenguaje de programación y admite de forma transparente construcciones como iteradores de C ++, aritmética de punteros o bucles basados en bucles.
Está construido alrededor de una biblioteca poliédrica avanzada con soporte completo para variables cuantificadas existencialmente e incluye un análisis de dependencia de última generación. Debido a una interfaz de archivo simple, es posible aplicar transformaciones manualmente o utilizar un optimizador externo. Usamos esta interfaz para integrar Plutón , un moderno optimizador y paralelizador de localidad de datos. Gracias a la generación de código SIMD y OpenMP integrada, Polly aprovecha automáticamente el paralelismo existente y recientemente expuesto.

Problema concreto de implementación.

Polly está diseñado como un conjunto de pases de optimización y análisis internos del compilador. Se pueden dividir en pases frontales, intermedios y posteriores. El extremo frontal se traduce de LLVM-IR en una representación poliédrica, el extremo medio transforma y optimiza esta representación y el extremo posterior lo traduce de nuevo a LLVM-IR. Además, existen pases de preparación para aumentar la cantidad de código analizable, así como pases para exportar y reimportar la representación poliédrica.

Para optimizar un programa manualmente se realizan tres pasos. En primer lugar, el programa se traduce a LLVM-IR. Posteriormente, se llama a Polly para optimizar LLVM-IR y, finalmente, se genera el código de destino. La representación LLVM-IR de un programa se puede obtener de compiladores basados en LLVM específicos del lenguaje. clang es una buena opción para C / C ++ / Objective-C, DragonEgg para FORTRAN y ADA, OpenJDP o VMKit para lenguajes basados en Java VM,

unladen-swallow para Python y GHC para Haskell. Polly también proporciona un reemplazo directo para gcc que se llama pollycc.

Selección de metodología

Para realizar la selección adecuada tuvimos primero que investigar ambas metodologías (ágil y tradicional) sin embargo dejaremos una tabla comparativa encontrada en una página web de una escuela la cual describe brevemente ambas metodologías y permitirá visualizar el porqué de nuestra elección.

Características	Enfoque ágil	Enfoque tradicional
Estructura organizativa	Iterativa	Lineal
Escala de proyectos	Pequeños y medios	Grandes
Requisitos	Dinámicos	Bien definidos antes de empezar
Implicación del cliente	Alta	Baja
Modelo de desarrollo	Entrega evolutiva	Ciclo de vida
Participación del cliente	Los clientes participan desde el momento en que se empieza a realizar el trabajo.	Los clientes se involucran al principio del proyecto, pero no una vez que la ejecución ha comenzado.
Gestión de escalado	Cuando ocurren problemas, todo el equipo trabaja junto para resolverlo.	El problema escala a los gerentes del proyecto.
Preferencias del modelo	El modelo ágil favorece la adaptación.	El modelo tradicional favorece la anticipación.
Producto o proceso	Menos enfoque en los procesos formales y directivos.	Más enfocados sobre los procesos que sobre el producto.
Planificación	Se planifica de Sprint en Sprint.	Se planifica todo con gran detalle.
Estimación del esfuerzo	El Scrum Master facilita las tareas y el equipo hace la estimación.	El gestor del proyecto estima y obtiene la aprobación del propietario del proyecto.
Revisiones y aprobaciones	Las revisiones se realizan después de cada iteración.	Constantes revisiones y aprobaciones por parte de los líderes del proyecto.

Además con base a lo dicho en la revista espirales acerca de estos dos métodos es que tomamos la decisión de que el método tradicional es el método que más se apega a la forma de trabajo ya que nuestra manera de avance es progresiva y lineal, la escala a pesar de ser un pequeño proyecto,

tiene los objetivos y requisitos perfectamente definidos, nos estamos anticipando a los problemas de manera que primero tenemos que investigar todo a fondo para que no surjan problemas durante la elaboración y esta en constantes revisiones por parte del líder del proyecto que en este caso es el profesor cuyo papel también es el cliente pero como cliente no interactúa más allá de su primer pedido.

Requerimientos	
Funcionales	**No funcionales**
Transformación de código	Dependencia en C
Administración de los recursos	Admisión de subprocesos
Separación en pasos	
Uso de biblioteca poliédrica	

DISEÑO Y DESARROLLO DEL PROYECTO FINAL POLLY

Implementación de reintentos de llamada HTTP con retroceso exponencial con IHttpClientFactory y las directivas de Polly

El enfoque recomendado para los reintentos con retroceso exponencial consiste en aprovechar las ventajas de las bibliotecas de .NET más avanzadas como la biblioteca de código abierto Polly.

Polly es una biblioteca de .NET que proporciona capacidades de resistencia y control de errores transitorios. Puede implementar esas funcionalidades mediante la aplicación de directivas de Polly como las de reintento, interruptor, aislamiento compartimentado, tiempo de espera y reserva. Polly tiene como destino .NET Framework 4.x y .NET Standard 1.0, 1.1 y 2.0 (que admite .NET Core).

A continuación se muestra cómo usar reintentos HTTP con Polly integrados en IHttpClientFactory.

Referencias a los paquetes de ASP.NET Core 3.1

IHttpClientFactory está disponible desde .NET Core 2.1, pero, a pesar de ello, es recomendable que se usen los últimos paquetes de ASP.NET Core 3.1 de NuGet en su proyecto. Normalmente también es necesario hacer referencia al paquete de extensión Microsoft.Extensions.Http.Polly.

Configurar un cliente con la directiva de reintentos de Polly, en Startup

Se tendrá que definir una configuración HttpClient cliente con nombre o tipo en el método Startup.ConfigureServices(...) estándar, pero ahora agregará código incremental en el que se especifica la directiva para los reintentos HTTP con retroceso exponencial, como se muestra a continuación:

```
//ConfigureServices()  - Startup.cs

services.AddHttpClient<IBasketService, BasketService>()

      .SetHandlerLifetime(TimeSpan.FromMinutes(5))  //Set lifetime to five minutes

    .AddPolicyHandler(GetRetryPolicy());
```

El método AddPolicyHandler() es el que agrega las directivas a los objetos HttpClient que se van a usar. En este caso, se agrega una directiva de Polly para reintentos HTTP con retroceso exponencial.

Para tener un enfoque más modular, la directiva de reintentos HTTP se puede definir en un método independiente dentro del archivo Startup.cs, como se muestra en el código siguiente:

```
static IAsyncPolicy<HttpResponseMessage> GetRetryPolicy()

{

  return HttpPolicyExtensions

    .HandleTransientHttpError()

                    .OrResult(msg => msg.StatusCode == System.Net.HttpStatusCode.NotFound)

                    .WaitAndRetryAsync(6, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2,

                                    retryAttempt)));
```

Con Polly, se puede definir una directiva de reintentos con el número de reintentos, la configuración de retroceso exponencial y las acciones necesarias cuando se produce una excepción de HTTP, como registrar el error. En este caso, la directiva está configurada para intentar seis veces con un reintento exponencial, a partir de dos segundos.

Agregar una estrategia de vibración a la Directiva de reintentos

Una directiva de reintentos normal puede afectar a su sistema en casos de escalabilidad y simultaneidad altas y de gran contención. Para gestionar los picos de reintentos similares procedentes de diferentes clientes en caso de interrupciones parciales, una buena solución es agregar una estrategia de vibración a la directiva o algoritmo de reintento. Esto puede mejorar el rendimiento general del sistema de un extremo a otro añadiendo aleatoriedad al retroceso exponencial. De esta forma, cuando surgen problemas, los picos se reparten. El principio que rige esto se muestra en el ejemplo siguiente:

```
Random jitterer = new Random();

var retryWithJitterPolicy = HttpPolicyExtensions
   .HandleTransientHttpError()
                  .OrResult(msg => msg.StatusCode == System.Net.HttpStatusCode.NotFound)
   .WaitAndRetryAsync(6,    // exponential back-off plus some jitter
                  retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt))
                  + TimeSpan.FromMilliseconds(jitterer.Next(0, 100))
   );
```

Permite que una aplicación trate los errores transitorios cuando intenta conectarse a un servicio o un recurso de red, mediante el reintento de forma transparente de una operación con error. Esto puede mejorar la estabilidad de la aplicación.

CONTEXTO Y PROBLEMA.

Una aplicación que se comunica con elementos que se ejecutan en la nube tiene que ser vulnerable a los errores transitorios que puedan producirse en este entorno. Entre los errores transitorios cabe mencionar la pérdida momentánea de conectividad de red en componentes y servicios, la falta de disponibilidad temporal de un servicio o los tiempos de espera que surgen cuando un servicio está ocupado.

Estos errores normalmente se corrigen automáticamente, y si se repite la acción que desencadena un error tras un retraso adecuado, es probable que tenga éxito. Por ejemplo, un servicio de base de datos que procesa un número elevado de solicitudes simultáneas puede implementar una estrategia de limitación que temporalmente rechace las sucesivas solicitudes hasta que su carga de trabajo haya disminuido. Una aplicación que intenta acceder a la base de datos podría no conectarse, pero si lo intenta de nuevo después de un retraso podría tener éxito.

SOLUCIÓN

En la nube, los errores transitorios son algo habitual y una aplicación debería diseñarse para tratarlos con elegancia y transparencia. De esta forma, se reducen los efectos que pueden tener los errores sobre las tareas empresariales que realiza la aplicación.

Si una aplicación detecta un error al intentar enviar una solicitud a un servicio remoto, puede tratar el error mediante las estrategias siguientes:

Cancelar. Si el error indica que el error no es transitorio o que no es probable que tenga éxito si se repite, la aplicación debe cancelar la operación y notificar una excepción. Por ejemplo, un error de autenticación ocasionado al proporcionarse credenciales no válidas no es probable que tenga éxito sin importar cuántas veces se intente.

Reintentar. Si el error específico notificado es inusual o poco frecuente, se podría deber a circunstancias poco habituales, como un paquete de red que se daña mientras se está transmitiendo. En este caso, la aplicación podría reintentar la solicitud con error inmediatamente porque es improbable que se repita el mismo error y lo más probable es que la solicitud tenga éxito.

Reintentar después de un retraso. Si el error es debido a uno de los muchos errores habituales de conectividad o disponibilidad, puede que la red o el servicio necesiten un corto período de tiempo mientras se corrigen los problemas de conectividad o se borra el trabajo pendiente. La aplicación debe esperar el momento adecuado antes de reintentar la solicitud.

Para los errores transitorios más comunes, se debe elegir el período entre reintentos a fin distribuir las solicitudes de varias instancias de la aplicación lo más uniformemente posible. De esta forma se reduce la posibilidad de que un servicio ocupado siga sobrecargado. Si muchas instancias de una aplicación sobrecargan continuamente un servicio con solicitudes de reintento, el servicio tardará más en recuperarse.

Si la solicitud sigue sin funcionar, la aplicación puede esperar y realizar otro intento. Si es necesario, este proceso puede repetirse aumentando los retrasos entre reintentos, hasta que se haya intentado un número máximo de solicitudes. El retraso se puede aumentar de manera exponencial o incremental, según el tipo de error y la probabilidad de que se corrija durante este tiempo.

Si la solicitud no tiene éxito después de un número predefinido de intentos, la aplicación debe tratar el error como una excepción y tratarlo como corresponda.

La aplicación debe encapsular todos los intentos de acceso a un servicio remoto en un código que implemente una directiva de reintento que coincida con una de las estrategias enumeradas anteriormente. Las solicitudes enviadas a distintos servicios pueden estar sujetas a diferentes directivas. Algunos proveedores proporcionan bibliotecas que implementan directivas de reintento, donde la aplicación puede especificar el número máximo de reintentos, el tiempo entre reintentos y otros parámetros.

Una aplicación debe registrar los detalles de los errores y las operaciones con errores. Esta información es útil para los operadores. Si un servicio está con frecuencia no disponible o ocupado,

Suele ser porque el servicio ha agotado sus recursos. Se puede reducir la frecuencia de estos errores mediante el escalado horizontal del servicio. Por ejemplo, si un servicio de base de datos está continuamente sobrecargado, podría ser beneficioso particionar la base de datos y repartir la carga entre varios servidores.

Referencias

bryan molina montenegro, harry vite cevallos y jefferson davila cuesta. (junio 2018). Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. espirales , 62, 115-119.

Álvaro Rodelgo. (2019). GESTIÓN ÁGIL VS GESTIÓN TRADICIONAL DE PROYECTOS ¿CÓMO ELEGIR?. 20/12/2020, de escuela de negocios feda Sitio web: https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir#:~:text=En%20la%20metodología%20ágil%2C%20cada,comparte%20la%20propiedad%20del%20proyecto.&text=En%20el%20enfoque%20tradicional%2C%20cada,del%20tiempo%20y%20presupuesto%20estimados.

https://polly.llvm.org/todo.html fecha de consulta 15/01/2021
https://polly.llvm.org/get_started.html fecha de consulta 18/01/2021