Instituto Tecnologico de Iztapalapa

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ENGINEERING IN COMPUTER SYSTEMS

FINAL PROJECT POLLY

PRESENT:

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

NO. CONTROL:

171080043
171080044
161080183
171080151

INTERNAL ADVISOR:

PARRA HERNANDEZ ABIEL TOMAS

MEXICO CITY **JANUARY / 2021**

General summary of your final project.

Polly is LLVM's advanced data localization and loop optimization and optimization framework. It uses abstract mathematical notation based on an integer polyhedral method to analyze and optimize the memory access pattern of the program. We currently perform classic looping transformations, especially mosaic and loop fusion, to improve data placement.

Polly can also take advantage of OpenMP level parallelism to expose opportunities for SIMD. Work on automatic GPU code generation has also been completed.

However, for many users, the most interesting thing is not the existing optimizations in Polly, but the new analyzes and optimizations enabled by the Polly framework.

Preliminary schedule of activities.

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

Description of LLVM Infrastructure Subproject (POLLY)

Polly automatically detects and optimizes generalized matrix multiplication, the calculation C ← α ⊗ C ⊕ β ⊗ A ⊗ B, where A, B and C are three appropriately sized matrices, the operations ⊕ and ⊗ originate from the corresponding matrix semiring, and α and β are constant and beta is not equal to zero. Lets get the highly optimized structured form similar to the expert implementation of GEMM that can be found in GotoBLAS and its successors.

Polly can now automatically optimize all polybench 2.0 kernels without the help of an external optimizer. The compilation time is reasonable and we can show remarkable speedups for various cores.

Risks	Proposed solutions
A sequential hardware	A software is created that optimizes local information
No optimal use of parallel hardware	Polly is used to call OpenMP runtime library with which all hardware can be used appropriately
Truncations in polly processes	It is used at polly to develop a bugpint file that catches this type of errors they have the extension ll
Polly is unable to recognize a bugpoint, clang or opt	A configuration of the tools is required before compiling

Polly - Polyhedral optimization in LLVM

With Polly, we are developing a state-of-the-art polyhedral infrastructure for LLVM, which supports fully automatic transformation of existing programs. Polly detects and extracts relevant code regions without any human interaction. Since Polly accepts LLVM-IR as input, it is independent of the programming language and transparently supports constructs such as C ++ iterators, pointer arithmetic, or loop-based loops.
It is built around an advanced polyhedral library with full support for existentially quantified variables and includes state-of-the-art dependency analysis. Due to a simple file interface, it is possible to apply transformations manually or use an external optimizer. We use this interface to integrate Pluto, a modern data locality optimizer and parallelizer. Thanks to the built-in SIMD and OpenMP code generation, Polly automatically takes advantage of existing and recently exposed parallelism.

Specific implementation problem.

Polly is designed as a set of internal compiler parsing and optimization passes. They can be divided into front, middle and back passes. The front end is translated from LLVM-IR into a polyhedral representation, the middle end transforms and optimizes this representation, and the back end translates it back to LLVM-IR. In addition, there are preparation passes to increase the amount of parsable code, as well as passes to export and re-import the polyhedral representation.

There are three steps to manually optimize a program. First of all, the program is translated into LLVM-IR. Polly is then called to optimize LLVM-IR, and finally the target code is generated. The LLVM-IR representation of a program can be obtained from language-specific LLVM-based compilers. clang is a good choice for C / C ++ / Objective-C, DragonEgg for FORTRAN and ADA, OpenJDP or VMKit for Java VM based languages, unladen-swallow for Python and GHC for Haskell. Polly also provides a direct replacement for gcc which is called pollycc.

Methodology

To make the proper selection we had to first investigate both methodologies (agile and traditional), however we will leave a comparative table found on a school website which briefly describes both methodologies and will allow us to visualize the reason for our choice.

Features	Focus agile	approach traditional
organizational structure	Iterative	linear
scale ofprojects	smallandmedia	big
Requirements	Dynamical	well defined before starting
Implication of the customer	High	Low
Development Model	Delivery evolutionary	life cycle
Participation client	clients involved from the moment you start get the job done.	Clients are involved early in the project, but not once execution has begun.
Escalation management	When problems occur, the entire team works together to solve them.	The problem escalates to the project managers.
Model preferences	The agile model favors adaptation.	The traditional model favors anticipation.
Product or process	Less focus on formal and managerial processes.	More focused on the processes than on the product.
Planning	It is planned from Sprint to Sprint.	Everything is planned in great detail.
Effort estimation	The Scrum Master facilitates the tasks and the team makes the estimation.	The project manager estimates and gets approval from the project owner.
Reviews and Approvals	Reviews are performed after each iteration.	Constant reviews and approvals by the project leaders.

In addition, based on what was said in the spiral magazine about these two methods, we made the decision that the traditional method is the method that most closely matches the way of working since our way of progress is progressive and linear, the scale Despite being a small project, it

has perfectly defined objectives and requirements, we are anticipating problems so that first we have to investigate everything thoroughly so that problems do not arise during development and it is constantly reviewed by the project leader. project which in this case is the teacher whose role is also the client but as a client does not interact beyond their first request.

Requirements	
Functional	**Non-functional**
Code transformation	Dependency in C
Resource management	Admission of threads
Separation into steps	
Use of polyhedral library	

DESIGN AND DEVELOP THE FINAL POLLY PROJECT

Implementing HTTP Call Retries with Exponential Backspace with IHttpClientFactory and Polly Directives

The recommended approach for Exponential Backspace retries is to take advantage of more advanced .NET libraries such as the Code Library open Polly.

Polly is a .NET library that provides transient error handling and resilience capabilities. You can implement these capabilities by applying Polly policies such as retry, switch, compartmentalized isolation, timeout, and reservation. Polly targets .NET Framework 4.x and .NET Standard 1.0, 1.1, and 2.0 (which supports .NET Core).

Here's how to use HTTP retries with Polly built into IHttpClientFactory.

References to ASP.NET Core 3.1

Packages IHttpClientFactory is available since .NET Core 2.1, but it is still recommended that you use the latest ASP.NET Core 3.1 packages from NuGet in your project. Usually you also need to refer to the Microsoft.Extensions.Http.Polly extension pack.

Configure a client with Polly's retry policy, in Startup

You will have to define a client HttpClient configuration with name or type in the standard Startup.ConfigureServices (...) method, but now you will add incremental code in which the directive is specified for HTTP retries with exponential rollback, as follows:

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

services.AddHttpClient <IBasketService, BasketService> ()

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

    .AddPolicyHandler (GetRetryPolicy ());
```

The AddPolicyHandler () method is the one that adds the directives to the HttpClient objects to be used. In this case, a Polly directive is added for HTTP retries with exponential rollback.

For a more modular approach, the HTTP retry policy can be defined in a separate method within the Startup.cs file, as shown in the following code:

```
static IAsyncPolicy <HttpResponseMessage> GetRetryPolicy ()
{
   return HttpPolicyExtensions
      .HandleTransientHttpError ()
                  . OrResult (msg => msg.StatusCode == System.Net.HttpStatusCode.NotFound)
      .WaitAndRetryAsync (6, retryAttempt => TimeSpan.FromSeconds (Math.Pow (2,
                                    retryAttempt)));
```

With Polly, you can define a retry policy with the number of retries, the exponential rollback setting, and the actions required when an HTTP exception occurs, such as logging the error. In this case, the policy is configured to try six times with an exponential retry, starting at two seconds.

Adding a shake strategy to the retry

policy A normal retry policy can affect your system in cases of high scalability and concurrency and high contention. To handle similar retry spikes from different clients in the event of partial outages, a good solution is to add a shake strategy to the retry policy or algorithm. This can improve the overall performance of the system end-to-end by adding randomness to the exponential backtracking. In this way, when problems arise, the spikes are distributed. The principlethis is shown in the following example:

```
behindRandom jitterer = new Random ();

var retryWithJitterPolicy = HttpPolicyExtensions

   .HandleTransientHttpError ()

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

   .WaitAndRetryAsync (6, // exponential back-someplus-plus-some-

                                          offSpantrySpantry
plusAmptySpantrySpantryretrySpantry.TimeecontryoffSpantrySpantry
Spantry.TimeFoundSpantryRetrySpantryoffSpantry-time. Pow (2,
retryAttempt))

                  + TimeSpan.FromMilliseconds (jitterer.Next (0, 100))

   );
```

Allows an application to handle transient errors when trying to connect to a network resource or service, transparently retrying a failed operation. This can improve the stability of the application.

CONTEXT AND PROBLEM.

An application that communicates with elements running in the cloud has to be vulnerable to the transient errors that can occur in this environment. Transient errors include momentary loss of network connectivity in components and services, temporary unavailability of a service, or waiting times that arise when a service is busy.

These errors are usually corrected automatically, and if the action that triggers an error is repeated after an appropriate delay, it is likely to be successful. For example, a database service that processes a large number of concurrent requests might implement a throttling strategy that temporarily rejects successive requests until its workload has decreased. An application trying to access the database might fail to connect, but trying again after a delay might succeed.

SOLUTION

In the cloud, transient errors are common and an application should be designed to handle them gracefully and transparently. This reduces the effects that errors can have on the business tasks that the application performs.

If an application encounters an error when trying to send a request to a remote service, it can handle the error using the following strategies:

Cancel. If the error indicates that the error is not transient or that it is unlikely to be successful if it recurs, the application should cancel the operation and report an exception. For example, an authentication failure caused by providing invalid credentials is unlikely to be successful no matter how many times it is tried.

Retry If the specific error reported is unusual or infrequent, it could be due to unusual circumstances, such as a network packet being corrupted while transmitting. In this case, the application might retry the failed request immediately because the same error is unlikely to be repeated and the request is most likely successful.

Retry after a delay. If the error is due to one of the many common connectivity or availability errors, the network or service may require a short period of time while connectivity issues are corrected or pending work is cleared. The application must wait for the appropriate moment before retrying the request.

For the most common transient errors, the period between retries should be chosen to distribute requests from multiple instances of the application as evenly as possible. This reduces the possibility of a busy service remaining overloaded. If many instances of an application continually overload a service with retry requests, it will take longer for the service to recover.

If the request still fails, the application can wait and make another attempt. If necessary, this process can be repeated, increasing the delays between retries, until a maximum number of requests have been attempted. The delay can be increased exponentially or incrementally, depending on the type of error and the likelihood of it being corrected during this time.

If the request is unsuccessful after a predefined number of attempts, the application should treat the error as an exception and treat it accordingly.

The application must encapsulate all attempts to access a remote service in code that implements a retry policy that matches one of the strategies listed above. Requests sent to different services may be subject to different policies. Some vendors provide libraries that implement retry policies, where the application can specify the maximum number of retries, the time between retries, and other parameters.

An application should record the details of errors and failed operations. This information is useful for operators. If a service is frequently unavailable or busy, it is

usually because the service has exhausted its resources. You can reduce the frequency of these errors by scaling out the service. For example, if a database service is continually overloaded, it could be beneficial to partition the database and spread the load across multiple servers.

References

bryan molina montenegro, harry vite cevallos and jefferson davila cost. (June 2018). Agile methodologies compared to traditional ones in the software development process. spirals, 62, 115-119.

Álvaro Rodelgo. (2019). AGILE MANAGEMENT VS TRADITIONAL PROJECT MANAGEMENT HOW TO CHOOSE ?. 12/20/2020, from feda business school Website: https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-traditional-de-proyectos-como-elegir#:~:text=En % 20the% 20methodology% 20agile% 2C% 20 each, shares% 20the% 20property% 20of% 20project. & Text = In% 20the% 20traditional% 20 approach% 2C% 20 each, of% 20time% 20and% 20 budget% 20estimated.

https://polly.llvm.org/todo.html consultation date 01/15/2021
https://polly.llvm.org/get_started.html consultation date 01/18/2021