Gpu-FI User MANuAL



TAble of content

[**Prepare Application** 3](#_Toc58028418)

[**Configuratıon structure** 3](#_Toc58028419)

[**Sub-Configuration Example** 3](#_Toc58028420)

[**General Configuration** 3](#_Toc58028421)

[**Basic Flags** 3](#_Toc58028422)

[**Fault Injection Directory Configurations** 4](#_Toc58028423)

[**General Fault Injection Configurations** 4](#_Toc58028424)

[**Specification of Fault Injection** 4](#_Toc58028425)

[**Regional Fault Injection Configurations** 5](#_Toc58028426)

[**Loading single/multiple Configurations** 5](#_Toc58028427)

[**Fault InjectIon Test** 6](#_Toc58028428)

[**After Injectıon** 7](#_Toc58028429)

# **Prepare Application**

Uygulamaya başarılı bir şekilde hata enjeksiyonu ve çıkışlarını karşılaştırmak için uygulama çıkışının uygulama bitişinde kaydedilmiş olması ve hatasız örneği ile karşılaştırılabilir olması gerekmektedir. Yani Deterministtik olmalıdır. Aksi taktirde araç iki çıkışı farklı bulduğunda SDC olarak varsayacaktır.

# **Configuratıon structure**

Tasarlanan araç detaylı hata enjeksiyonuna izin vermekte olup birden fazla konfigürasyon dosyası ile test çalıştırma imkânı sağlar. Ayrıca araç genel konfigürasyon dosyası ve alt konfigürasyon dosyaları olarak çalışabilmektedir.

## **Sub-Configuration Example**

Alt konfigürasyon dosyası, genel konfigürasyon dosyasının aksine sadece değiştirilmesi istenen değişkenlerden oluşur. Bu sayede sadece birkaç parametre belirtilerek hata enjeksiyonu testi çalıştırabilir. Ayrıca alt konfigürasyon dosyaları ile birden fazla testi sıralı bir şekilde çalıştırmak mümkündür. Bu konfigürasyon dosyasında belirtilen tüm yollar ***directory*** yolunun üstüne eklenerek hesaplanır örnek olarak ***output*** değişkeninin yolu ***Output\_path = directory + output***

# Directory of Binary File

directory = "./Samples/Polybench/3MM/"

# Binary file

executable = "3mm"

# Uygulamanın kesilmesi istenen satır

breakKernelStart = "3mm.cu:135"

# number of faults will be done

num\_fault = 1000

# Output destination of Execution

output = "Output.txt"

# **General Configuration**

## **Basic Flags**

***verbose*** parametresi raporlama seviyesini belirler. **0** seviyesinde hiçbir şey yazdırılmaz. **1** seviyesinde Enjeksiyon özeti bilgiler yazdırılır. **2** seviyesinde ise uygulama çıkışı dahil her şey yazdırılır enjeksiyon ekranına. ***Verbose*** parametresi bir problem anında durumunda çıkışı kontrol edebilmek için faydalıdır

**clean** parametresi Hata enjeksiyonu bittikten sonra sistemde oluşturulan geçici dosyaları temizler. Bu dosyalar hata enjeksiyonunun parçalı olarak çalışacağı durumlarda gereklidir o sebeple silinmemesi tercih edilir.

Temel bayraklar aracın parçalı olarak çalışmasına olanak sağlar.

***Profile\_flag*:** Profiler kısmının aktifliğini belirler. Profiler uygulamayı hatasız olarak çalıştırıp uygulamadan hatasız çıkış, çalışma süresi ve o satırdaki aktif thread sayısını toparlar. Bu bilgileri araç içinde sonradan kullanılmak üzere **directory/temp/golden\_info.txt** dosyasına kaydeder.

**Fault\_generator\_flag:** parametresi belirlenen sayıda hatayı uniform olarak yapılması için konumlarının belirler. Bu aşama hatanın ***block/thread/register & addr/ bit*** konumlarını **directory/temp/fault\_map.txt** dosyasına yazar.

**Fault\_injector\_flag:** parametresi ise ***Profillerdan*** ve *Fault\_map\_generatorden* aldığı bilgileri kullanarak hataları oluşturur.

# Log level (optional) 0, 1 ,2.

verbose = 2

clean = False               # To clean temp files

Profile\_flag = True         # Use profiler

Fault\_creator\_flag = True   # Use Fault generator

Fault\_injector = True       # Use Fault injector

## **Fault Injection Directory Configurations**

Bu kısımda enjeksiyon yapılacak uygulamanın temel bilgileri belirlenmektedir. Aracın çalışması için uygulama sonucundaki verinin kaydedilip yolunun çıkış olarak belirtilmesi gerekmektedir. Ayrıca hata enjeksiyonu özeti için yaratılacak olan CSV dosyasının ismi de bu kısımda belirtilir.

# Directory of Binary File

directory = "./Samples/Polybench/COVAR/"

# Binary file

executable = "covar"

# Arguments, if not necessary fill with blank string ""

args = ""

# Output results destination of Execution   \*\*\*\*Required for SDC check\*\*\*

output = "Output.txt"

# Fault injection Results output file

output\_csv = "Fault\_injection\_results.csv"

## **General Fault Injection Configurations**

Bu kısımda yapılacak hata enjeksiyonun sayısı, kaynak kodu üzerindeki kesme noktasın yeri ve varsa koşulu belirtilir.

# number of faults

num\_fault = 1000

breakKernelStart = "covariance.cu:159"

#breakConditional = "saxpy.cu:29 if i > 5"

breakConditional = ""

## **Specification of Fault Injection**

Bu kısımda hata enjeksiyonu için programın sınırları belirtilir. Bekleme faktörü programın askıda kalan bir programı öldürmeden önce ne kadar bekleyeceğinin katsayısıdır. Süre hatasız çalışan (golden) programın süresinin bekleme faktörü ile çarpılmasıyla hesaplanır.

# maximum wait time

hang\_factor = 5

# maximum register number

max\_register = 64

# maximum bit number

max\_bit = 32

## **Regional Fault Injection Configurations**

Bu kısımda hata enjeksiyonunun konumu için koşullar belirtilir. Araç farklı hafıza bölgelerinde özel konumlara hata enjeksiyonu gerçekleştirebilmektedir.

# Fault injection memory type/ fault region

# reg, const, @global, @local, @shared, @generic

fault\_reg = "reg"

# In case of specific address space

min\_address = "0x140"

max\_address = "0x250" # address space interval

# In case of fault with specific block and thread

# Example: "1, 0, 0", otherwise ""

# Coordinates are consequently x y z

block\_idx = ""

thread\_idx = ""

# In case of fault with specific register,

# Example: register\_id = "12" or left it blank

register\_id = ""

## **Loading single/multiple Configurations**

Bu kısımda ilk kısımda bahsedilin tekli veya çoklu alt konfigürasyon dosyasının eklenmesi sağlanır. Süreç belirli bir alt klasör içerisinde alt konfigürasyon dosyalarının konumunun eklenmesiyle başlar. Örnek olarak config\_files/3mm klasörünün içindeki konfigürasyon örnekleri verilebilir. Araç üzerinde *(Fault\_injector.py conf\_list*) belirtilen sıra ile konfigürasyon dosyaları çalıştırılır. Aşağıdaki örnekte de görüleceği üzere konfigürasyon dosyalarının isimlerinin önceden belirtilmiş olması gerekmektedir. Burada iki yol izlenebilir. Bunlardan ilki konfigürasyon dosyalarının isimlerini modül tanımlamalarına manuel olarak eklemek. İkinci yol ise alt konfigürasyonlara “config1, config2” şeklinde isim vererek hızlı bir çözüm sağlamak.

**Not:** Eğer aracın çalıştı ana dizinde alt konfigürasyonla aynı isimde başka bir dosya var ise araç bu dosyayı öncelikli kabul eder.

**Not:** Eğer tek konfigürasyon çalıştırılacaksa araç içerisinden sadece istenilen alt konfigürasyon dosyası belirlenebilir. Ayrıca çoklu konfigürasyon iptal edilipte program tekli olarak kullanabilir.

use\_load\_configs = True

# The path of configurations

config\_path = "./config\_files/3mm\_test/"

if use\_load\_configs:

    import sys

    try:

        # With run\_number.txt, the tool will be communicating with config.py

        f = open("run\_number.txt","r")

        number = int(f.read())

        f.close()

    except:

        # if there is no run\_number.txt main configurations will be used

        number = 0

        print("run\_number.txt file cannot found. Main configurations used")

    # New path is appended in the system path

    sys.path.append(config\_path)

    # This block helps to make multiple tests in a single run.

    # The files must be named as config1 or you can specifice the name in the module declaration code. For example: from config\_matmul\_1024\_1 import \*

    if number == 1:

        print("config 1 will be used")

        from config1 import \*

    if number == 2:

        print("config 2 will be used")

        from config2 import \*

    if number == 3:

        print("config 3 will be used")

        from config3 import \*

    if number == 4:

        print("config 4 will be used")

        from config4 import \*

# **Fault InjectIon Test**

Konfigürasyonlar hazırlandıktan sonra, Fault\_injector.py uygulaması içerisindeki conf\_list listesine çalıştırılması istenilen konfigürasyonların numaraları eklenir. Aşağıda verilen örnekte 3 ve 4 verilerek config.py dosyasında config3 ve config4 ün çalıştırılacağı anlamına gelmektedir.

def main():

    # list of configurations

    conf\_list = [3, 4]

    for i in conf\_list:

        # Control config.py via run\_number.txt

        f = open("run\_number.txt","w")

        f.write(str(i))

        f.close()

        # Reload config.py

        reload(conf)

        print("Configration %d is loaded" %(i))

        setup()

        # Profilling application

        if conf.Profile\_flag:

            Profiler()

        # Generation of Faults informations

        if conf.Fault\_creator\_flag:

            Fault\_creator()

        # Fault injection

        if conf.Fault\_injector\_flag:

            Fault\_injector()

    if conf.clean:

        os.system("rm run\_number.txt")

Bu aşamada tamamlandıktan sonra program terminal üzerinden Python 3 ile çalıştırılır ve enjeksiyon işlemi başlar.

>> python3 Fault\_injector.py

# **After Injectıon**

Enjeksiyon sonrasında ise Araç SDC çıkışlarını, sonuç CSV dosyasını *SDC\_Outputs* klasöründe enjeksiyonun başlangıç tarihinde bir dizin oluşturup kaydeder. Detaylı sonuçları daha sonrasında oradan kontrol edebilirsiniz.

# **Known bugs**

* Araç bir uygulama için ilk defa kullanıldığında eksik klasör hatası verebiliyor. İkinci çalıştırılmasında bu hata düzeliyor.
* Askıda kalma durumu kontrolü su anda yapılamıyor.