MC970/MO644 – Programação Paralela

Laboratório 11 – Detectando Dependências Loop-Carried

Professor: Guido Araújo
Monitor: Maicol Gomez Zegarra
Autor: João P. L. de Carvalho

1 Dependências Loop-Carried

Neste laboratório iremos detectar dependências em laços `DOACROSS`. As dependências neste tipo de laço são denominadas *Loop-Carried Dependencies*. Como visto em sala, dizemos que um laço é `DOACROSS` quando existe algum *statement* da iteração i que depende de um *statement* da iteração j, sendo $i \neq j$. Ou seja, existe uma relação de ordem entre *statements* de diferentes iteração que, se violada, produz um resultado diferente do esperado.

2 Enunciado

O objetivo do laboratório é identificar as dependências em laços `for` em programas `C/C++`. Para isso, será usado o detector de dependências implementado no Clang [1]. Você não utilizará as ferramentas diretamente, todo o ambiente já está configurado na máquina Parsusy. Para detectar as dependências basta adicionar a clausula `check` aos laços anotados com a diretiva `parallel-for`.

3 Testes e Resultado

Para esse laboratório foram selecionadas 3 aplicações em `C/C++`. As aplicações devem ser compiladas usando os arquivos do CMake distribuídos juntos com o código fonte. As ferramentas estão instaladas na máquina Parsusy. O perfilamento e detecção das dependências deve ser feito usando apenas uma thread. Os passos para compilar uma aplicação são:

[1] https://clang.llvm.org/

1. Definir as variáveis de ambiente para usar a versão customizada do compilador Clang e colocá-lo no PATH. Para isso basta executar o comando `source` passado o *script* `setenv.sh` (distribuído no zip deste laboratório).

```
$ source setenv.sh
```

2. Criar um diretório para configurar e construir o binário. Por exemplo, criar um subdiretório (*build*) no diretório raiz da aplicação.

```
$ ls
CMakeLists.txt main.c
$ mkdir build
$ ls
build CMakeLists.txt main.c
```

3. Entrar no diretório criado no passo anterior e executar o comando `cmake` definindo os compiladores `C` e `C++`.

```
$ cd build
$ CC=clang CXX=clang++ cmake ../
```

4. Para compilar a versão sem detecção de dependências basta executar o comando `make` sem nenhum argumento. A versão com detecção de dependência pode ser compilada passado a nome da aplicação seguido de "-check" (sem espaço).

```
$ make
$ ls
CMakeFiles . . . app
$ make app-check
$ ls
CMakeFiles . . . app app-check
```

5. Para executar, consulte a Tabela 3 com os argumentos de cada aplicação [2] [3]

Tabela 1: Argumentos para executar cada aplicação

Aplicação	Argumentos
bfs	-i <DATA>/bfs/1M/input/graph_input.dat
histo	-i <DATA>/histo/default/input/img.bin -- 20 4
cutcp	-i <DATA>/cutcp/small/input/watbox.sl40.pqr

A aplicação *bfs* já está anotada com as diretiva `parallel-for` do OpenMP e a clausula `check`. Ela pode ajudar na verificação do ambiente e garantir que você entendeu e

[2] http://impact.crhc.illinois.edu/parboil/parboil.aspx
[3] <DATA>: caminho da pasta datasets

seguiu corretamente os passos. As 2 aplicações restantes não estão anotadas. Você deve encontrar o(s) laço(s) `for` "quente(s)" das aplicações e anotá-los com a diretiva e a clausula `check`. Recomenda começar pela aplicação *histo* que é bem simples, e depois passar para a aplicação *cutcp*.

4 Submissões

A submissão deve ser um relatório **sucinto** em **pdf**. O relatório deve apontar as dependências encontradas e discutir, brevemente, sobre as implicações de uma paralelização ingênua na corretude das aplicações. Além disso, o relatório deve conter uma captura de tela do diagnóstico das dependências de cada aplicação (saída da versão com detecção de dependências).