MC970/MO644 – Programação Paralela

Laboratório 12 – Paralelismo de Laços DOACROSS

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

1 Paralelismo de Laços DOACROSS

Neste laboratório iremos paralelizar laços com dependências loop-carried (`DOACROSS`). Como visto em sala, esse tipo de laço usualmente não é paralelizado automaticamente pelo compilador dado a dificuldade, ou até mesmo impossibilidade, em detectar dependências estaticamente, algo que no caso geral é sabidamente NP-Completo [1]. Objetivando atacar esse problema e fornecer uma alternativa ao programador a clausula `use` foi implementada no Clang [2]. Como também visto em sala, a `use` permite selecionar qual algoritmo de paralelização usar para executar um laço `DOACROSS`.

2 Enunciado

O objetivo do laboratório é paralelizar os laços `DOACROSS` em programas `C/C++`. Para isso, empregaremos a ferramenta de detecção de dependências utilizado também no Laboratório "Detectando Dependências Loop-Carried". O detector fornecerá duas informações que guiarão a escolha do algoritmo mais adequado: (i) as dependências propriamente ditas e (ii) a probabilidade, medida em função da métrica LCP, das dependências. Munido dessas informações e do seu entendimento do comportamento do laço, você deve escolher entre os 3 algoritmos apresentados em sala:

DOAX: Uma implementação de uma variante do algoritmo de Cytron: Ron Cytron. "Doacross: Beyond vectorization for multiprocessors", ICPP 1986.

[1] Dror E. Maydan, John L. Hennessy, and Monica S. Lam. "Effectiveness of data dependence analysis". IJPP 1995

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

BDX: Uma implementação do algoritmo Batched `DOACROSS` de Cézão et al.: D. C. S. Lucas and G. Araujo, "The Batched DOACROSS loop parallelization algorithm", HPCS 2015.

TLS: Uma implementação do algoritmo Thread-Level Speculation baseado em Memória Transacional em Hardware de Salamanca et al.: Salamanca, J. and Amaral, José Nelson and Araujo, Guido. "Using Hardware Transactional Memory Support to Implement Thread-Level Speculation", TPDS 2017.

3 Testes e Resultado

Para esse laboratório foram selecionadas 2 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 e executar a versão com detecção de dependências são:

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` passando o script `setenv.sh` (distribuído na pasta deste laboratório).

```
$ source setenv.sh
```

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

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

3. Entrar no diretório criado no passo anterior e executar o comando `cmake`.

```
$ cd build-check
$ cmake -DCHECK=ON ../
```

4. Executar o comando `make`:

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

5. Para executar, consulte a Tabela 3 com a linha de comando de cada aplicação. As aplicações devem ser executadas no diretório build/build-check. Recomenda-se redirecionar a saída para um arquivo pois o número de dependências pode ser grande.

Tabela 1: Linha de comando para executar cada aplicação

Aplicação	Comando
loop-A	./serial-check 1125000
ray-rot	../run.sh serial-check

O laço de interesse já está anotado com a cláusula `check` nas duas aplicações. Esses são os laços que devem ser paralelizados usando a cláusula `use` com o algoritmo mais apropriado, escolhido com base na análise do laço e das dependências. Para análise dos laços utilize a ferramenta de sua preferência.

Uma vez que o laço esteja anotado com a diretiva `parallel-for` e a clausula `use`, para compilar os passos são similares ao da compilação com detecção de dependências. **IMPORTANTE: NÃO** usar o mesmo diretório para gerar a versão paralelizada e a versão com detecção de dependências. Para compilar e executar a versão paralelizada com a `use` basta:

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 na 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`. Note que a opção `-DCHECK` **NÃO** deve ser usada nesse caso!

```
$ cd build
$ cmake ../
```

4. Executar o comando `make` passando o nome do algoritmo selecionado com a clausula `use`. Note que você pode gerar a versão sequencial da aplicação passando `serial` como argumento. Essa versão pode ser usada para perfilamento.

```
$ make < bdx | tls | doax | serial >
$ ls
CMakeFiles . . . bdx
```

5. Para executar, consulte a Tabela 3. Note que agora, ao invés de `serial-check`, você deve usar o nome do binário que deseja executar.

4 Submissões

A submissão deve ser um relatório **sucinto** em **pdf**. O relatório deve apontar qual o melhor algoritmo para cada aplicação, discutir brevemente o processo de decisão para sua escolha, bem como o *speedup* obtido com cada um deles. Além disso, o relatório deve conter uma captura de tela do diagnóstico das dependências mais relevantes para a tomada de decisão na escolha do algoritmo de paralelização.

IMPORTANTE: a saída da versão paralela deve ser igual a da versão serial. Essa verificação é feita automaticamente no caso da aplicação `ray-rot`. O script `run.sh` reporta essa informação ao final da execução, mas para isso você deve ter executado a versão serial antes. Para a aplicação loop-A, desconsiderando informação de tempo, a saída padrão da versão paralela deve ser igual a serial (você deve verificar se os valores de contagem são os mesmo).