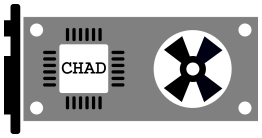


Computação Heterogénea de Alto Desempenho (2019/2020)

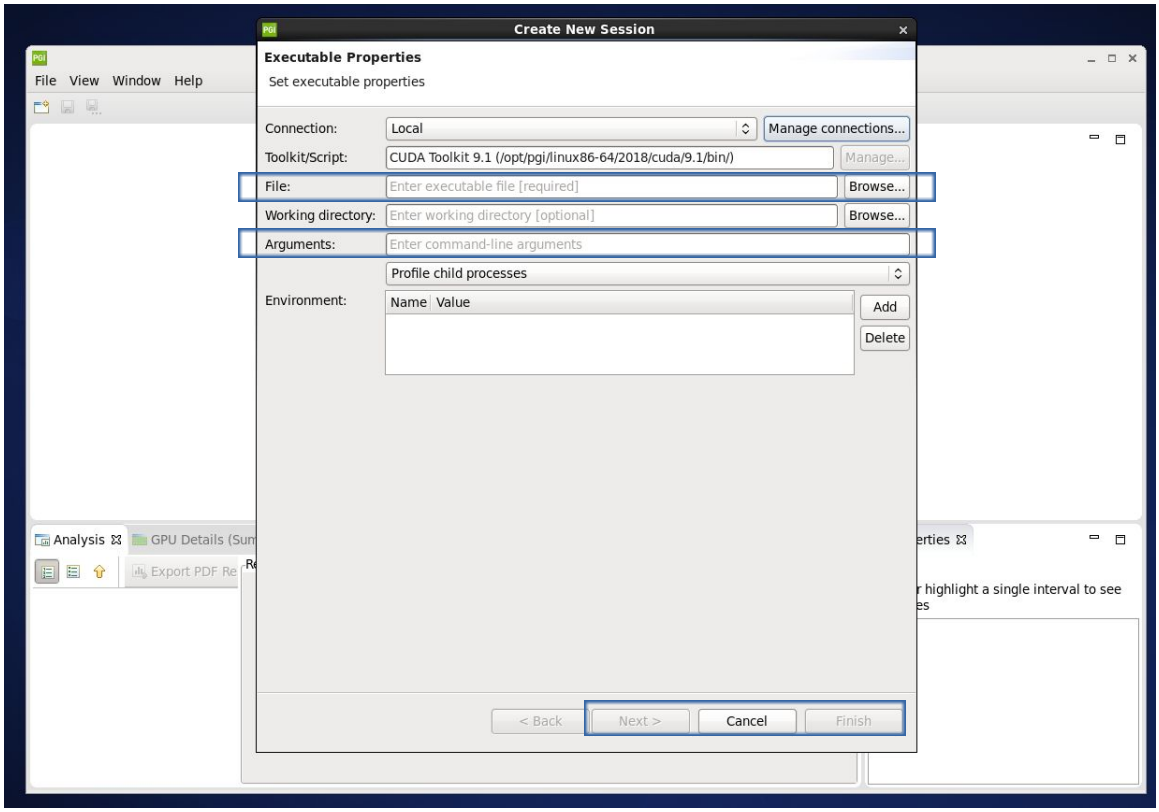
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

LAB 2

– OpenACC –

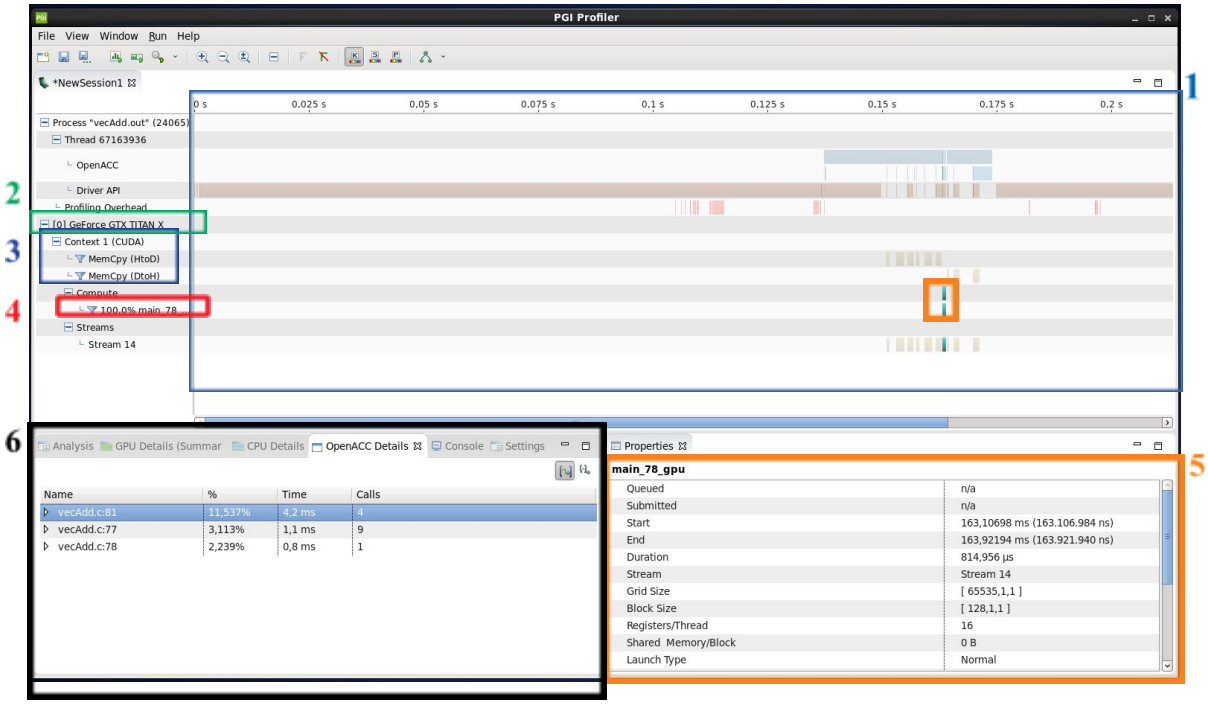
1. **PGI Profiler**. PGI Profiler é uma ferramenta útil para fazer a inspeção de código e perceber o seu nível de desempenho. Para familiarizar-se com esta ferramenta vai executar e analisar o perfil temporal do exemplo de adição de dois vectores ("*vecAdd*") disponibilizado como material de apoio. **Nota:** Antes de proceder à análise deve compilar o ficheiro vecAdd_openacc.c através do comando: **pgcc -lrt -lm -Minfo=all -acc -ta=tesla -o vecAdd_openacc.out vecAdd_openacc.c**

1.1. Execute o PGI Profiler e abra uma nova sessão: **File → New Session**.



1.2. No campo **File** selecione o botão **Browse,** navegue até ao exemplo "*vecAdd*" e selecione o executável **vecAdd_openacc.out**. No campo **Arguments** indique o número de elementos dos vetores (todos os vectores têm o mesmo número de elementos). Avance no botão **Next** e **Finish**.

1.3. Após pressionar o botão **Finish** o programa **vecAdd_openacc.out** vai ser executado e será gerada automaticamente uma linha temporal que descreve todas as instruções efectuadas pelo OpenACC. Na figura abaixo estão descritos alguns campos (indicados de 1 a 6, com cores distintas) importantes para análise. Para mais informações o utilizador deve consultar o manual de utilizador do programa:
https://www.pgroup.com/resources/docs/18.5/x86/profiler-users-guide/index.htm.)



Descrição dos campos 1 a 6:
1. Linha temporal interativa onde o utilizador pode consultar as chamadas do OpenACC API, transferências de memória Host/Device, kernels, etc;
2. Indicação do dispositivo usado na execução do programa;
3. Identificação de todas as transferências de memória entre host e device;
4. Identificação de todas as funções que executam no device;
5. Descrição detalhada da função a executar no dispositivo (para tal, basta clicar em cima da kernel na linha temporal para surgir esta informação). Pode consultar o número de blocos, threads por bloco, tempos de execução em detalhe, etc;
6. Informações mais específicas quanto à chamada da API do OpenACC, informação da consola, etc.

2. **Exercício 1: *vecAdd***. Neste exercício analisam-se as diferenças de performance entre a versão sequencial e a versão OpenACC de uma rotina que soma dois vectores e guarda o resultado num terceiro.

2.1. Inspecione, compile e execute o programa sequencial (vecADD/Sequential_version):

- **gcc -o vecAdd_seq.out vecAdd_seq.c -lrt -lm**
- **./vecAdd_seq.out num_elem**

Execute para $5*10^3$, $5*10^5$, $5*10^7$ elementos e registe os tempos de execução (tenha em atenção que só está a cronometrar o tempo de execução da adição e não da inicialização dos vetores de entrada).

2.2 Inspecione e compile o programa baseado em OpenACC (vecADD/OpenACC_version):

- **pgcc -lrt -lm -Minfo=all -acc ta=tesla -o vecAdd_openacc.out vecAdd_openacc.c**

2.2.1. Antes de executar, analise cada uma das flags de compilação (-lrt e -lm não é necessário dado serem flags do compilador GCC). Pode também pesquisar mais informação usando o comando **pgcc -help**.

2.2.2. Analise a informação que o compilador gerou e descreva-a.

2.2.3. Execute para o mesmo número de elementos acima mencionados e registe os tempos de execução.

- **./vecAdd_seq.out num_elem**

2.3. Explique a diferença de tempos entre a versão sequencial e a versão paralela com OpenACC.

2.4. Inicie o **PGI Profiler** e analise a versão OpenACC (uma sessão nova para cada execução do número de elementos).

2.4.1. Registe os tempos de inicializações da API do OpenACC, da execução do ciclo, assim como das transferências de memória.

2.4.2. Agora que detalhou cada um dos tempos de execução, explique novamente a diferença dos tempos observados entre a versão sequencial e a versão OpenACC.

3. **Exercício 2: matrixMul**. Neste exercício vamos explorar algumas das directivas do OpenACC, tendo como exemplo a multiplicação de duas matrizes quadradas. Tenha sempre a versão sequencial à mão e vá comparando (i.e., validando) ambas as versões de forma directa.

3.1. Edite o ficheiro **matrixMul_openacc.c** introduzindo a directiva `#pragma acc kernels` antes do loop principal (`for(i=0; i<n; i++)`). Compile, execute e verifique os resultados (compare o resultado da versão sequencial com a versão OpenACC imprimindo alguns valores no ecrã).

3.2. Analise as mensagens do compilador para perceber porque estão os resultados errados.

3.3. Adicione a diretiva `#pragma acc data copyin(a[0:n*n],b[0:n*n]),copy(c[0:n*n])` ao ciclo (loop) principal:

```
#pragma acc data copyin(a[0:n*n],b[0:n*n]),copy(c[0:n*n])
{
      #pragma acc kernels
      for(i=0; i<n; i++) {
         for(j=0;j<n;j++) {
            for(k=0;k<n;k++) {
               c[i*n+j]+=a[i*n+k]*b[k*n+j];
            }
         }
      }
}
```

Verifique os resultados e comente.

Explique cada uma das diretivas, especificando o que significa cada um dos seguintes *construtcs*: `copyin`, `copy`, `kernels`.

3.4. Verifique os tempos de execução de cada uma das versões (deve recorrer ao **PGI Profiler**) e comente.

3.5. Substitua o *construct* `kernels` por `parallel`. Se compilar e executar o programa, vai perceber que a *construct* `parallel` resulta em tempos de execução mais longos. Explique porquê (preste especial atenção às *threads per block* e *grid size*).

3.6. Configure a *construct* `parallel` de forma a obter tempos de execução equiparáveis à *construct* `kernels`. Para tal, introduza por esta ordem as seguintes *constructs*: `loop`, `num_gangs(num_gangs)`, `vector_length(vector_length)` e `data`:

3.6.1. *Construct*: `loop`

```
#pragma acc data copyin(a[0:n*n],b[0:n*n]),copy(c[0:n*n])
{
      #pragma acc parallel loop
      for(i=0; i<n; i++) {
         for(j=0;j<n;j++) {
            for(k=0;k<n;k++) {
               c[i*n+j]+=a[i*n+k]*b[k*n+j];
            }
         }
      }
}
```

3.6.2. C*onstructs:* `num_gangs(num_gangs)` e `vector_length(vector_length)`. Tendo em conta a dimensão da matriz, escolha adequadamente os parâmetros `num_gangs` e `vector_length`. Teste para vários valores e comente a performance do código (discuta como é que o paralelismo é extraído pelo compilador e relacione com o número de `num_gangs` e `vector_length` que selecionou).

```
#pragma acc data copyin(a[0:n*n],b[0:n*n]),copy(c[0:n*n])
{
      #pragma acc parallel loop num_gangs(num_gangs)
      vector_length(vector_length)
      for(i=0; i<n; i++) {
         for(j=0;j<n;j++) {
            for(k=0;k<n;k++) {
               c[i*n+j]+=a[i*n+k]*b[k*n+j];
            }
         }
      }
}
```

3.7. *Construct*: `data (create, copyout)`. Utilizando outras cláusulas desta *construct* tente melhorar o tempo total de execução do programa (tenha em atenção que deve cronometrar o tempo total de execução do programa). Comente.

3.8. Consegue melhorar a performance do código relativamente aos testes que efetuou até este ponto?