Introdução ao Desenvolvimento de Aplicações Paralelas com o Paradigma Orientado a Tarefas e o runtime StarPU

Minicurso 2

Lucas Leandro Nesi, Vinícius Garcia Pinto, Marcelo Cogo Miletto,
Lucas Mello Schnorr, Samuel Thibault

15 de abril de 2020

ERAD 2020

Apresentação

Para começar...

Etapas Práticas:

- Acessar site com material:

<https://gitlab.com/lnesi/minicurso-starpu-erad-2020>

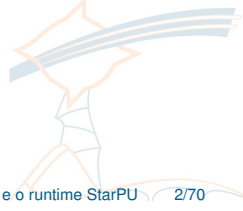
- Instalar StarPU:

```
git clone https://github.com/spack/spack.git
source spack/share/spack/setup-env.sh
git clone https://gitlab.inria.fr/solverstack/spack-repo.git
spack repo add spack-repo/
spack install starpu@1.3.1~fast+fxt~mpi~cuda~openmp
```

Contextualização do Minicurso

Mudanças no cenário de HPC para melhorar a programação paralela

- Desempenho: Heterogeneidade dos recursos (Aceleradores)
- Facilidades de programação: Paradigmas que facilitam a programação



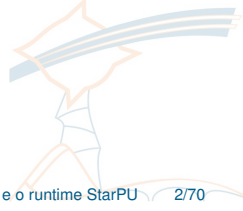
Contextualização do Minicurso

Mudanças no cenário de HPC para melhorar a programação paralela

- Desempenho: Heterogeneidade dos recursos (Aceleradores)
- Facilidades de programação: Paradigmas que facilitam a programação

Paradigmas tradicionais, como MPI, requerem do programador

- Manualmente mapear computação em recursos
- Manualmente explicitar as comunicações entre recursos



Contextualização do Minicurso

Mudanças no cenário de HPC para melhorar a programação paralela

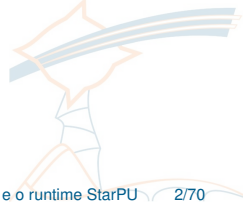
- Desempenho: Heterogeneidade dos recursos (Aceleradores)
- Facilidades de programação: Paradigmas que facilitam a programação

Paradigmas tradicionais, como MPI, requerem do programador

- Manualmente mapear computação em recursos
- Manualmente explicitar as comunicações entre recursos

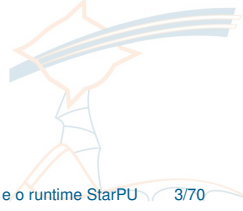
Novos paradigmas para facilitar esta programação

- Programação paralela orientada a tarefas



Objetivos

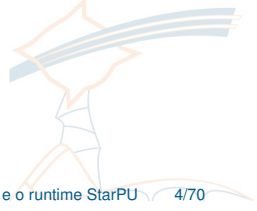
Apresentar aos participantes o paradigma orientado a tarefas e as bases para a sua programação



Estrutura do Minicurso

Dividido em quatro Etapas:

1. Estrutura e componentes de um programa paralelo orientado a tarefas
2. Apresentar o *runtime* StarPU e pequenas particularidades
3. Prática de como programar utilizando esta abordagem
4. Técnicas para analisar estes programas



Propaganda

Quem somos nós?

Lucas Nesi

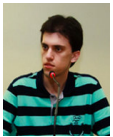
Doutorando PPGC
INF/UFRGS

llnesi@inf.ufrgs.br

Vinícius Pinto

Pós-doutorando
Prof. substituto
INF/UFRGS

vgpinto@inf.ufrgs.br



Marcelo Miletto

Mestrando PPGC
INF/UFRGS

mmiletto@inf.ufrgs.br

Lucas Schnorr

Prof. INF/UFRGS
Orientador PPGC

schnorr@inf.ufrgs.br

Samuel Thibault

Prof. Université de
Bordeaux

samuel.thibault@inria.fr

Programa de Pós-Graduação em Computação (PPGC)

Site do programa: <http://www.inf.ufrgs.br/ppgc/>

Oferece

- Mestrado, entradas anuais em março
  - Edital é lançado no segundo semestre
  - Requisito fundamental é o Poscomp
- Doutorado, entrada com fluxo continuo

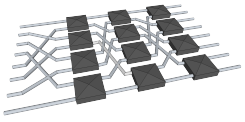
Fatos marcantes

- Conceito máximo (7) pela CAPES
- Internacionalização da formação e da investigação
- Número de Doutores formados: 361
- Número de Mestres formados: 1647

GPPD - Grupo de Processamento Paralelo e Distribuído

Site do grupo de pesquisa:
<http://www.inf.ufrgs.br/gppd/site/>
Eixos principais de investigação

- High Performance Computing
  (Computação de Alto Desempenho)
- Computer Architecture
- Big Data
- Cloud Computing
- FoG & Edge Computing

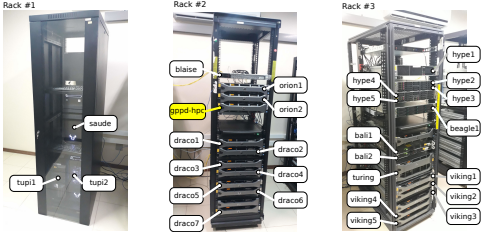


Parque Computacional de Alto Desempenho (PCAD)

Site: <http://gppd-hpc.inf.ufrgs.br/>

Possui aproximadamente 30 nós, 700+ núcleos de CPU e 73000+ de GPU

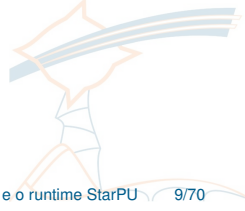
- Computação de alto desempenho heterogênea



Introdução

Visão geral

Problema para ser solucionado com computação de alto desempenho

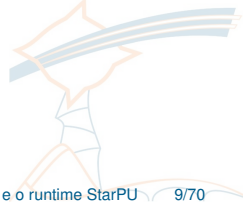


Visão geral

Problema para ser solucionado com computação de alto desempenho

Algoritmo

- Algoritmo para solucionar o problema



Visão geral

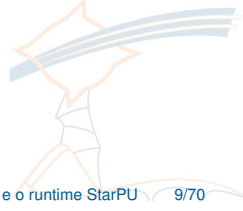
Problema para ser solucionado com computação de alto desempenho

Algoritmo

- Algoritmo para solucionar o problema

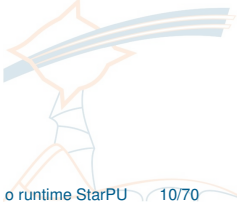
Recursos

- Diversas opções de possíveis recursos



Programando uma aplicação Paralela

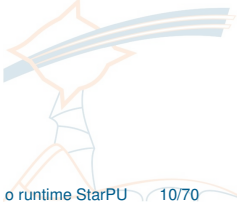
Algumas considerações para começar o desenvolvimento



Programando uma aplicação Paralela

Algumas considerações para começar o desenvolvimento

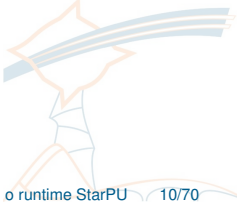
- Projeto da aplicação (PCAM) (Minicurso ERAD 2019)
  - Particionamento
  - Comunicação
  - Aglomeração
  - Mapeamento



Programando uma aplicação Paralela

Algumas considerações para começar o desenvolvimento

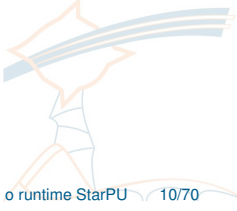
- Projeto da aplicação (PCAM) (Minicurso ERAD 2019)
  - Particionamento
  - Comunicação
  - Aglomeração
  - Mapeamento
- Escolher recursos computacionais
  - CPUs, Aceleradores?, múltiplos nós?



Programando uma aplicação Paralela

Algumas considerações para começar o desenvolvimento

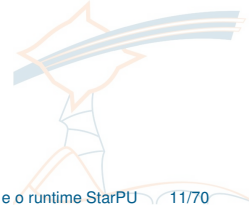
- Projeto da aplicação (PCAM) (Minicurso ERAD 2019)
  - Particionamento
  - Comunicação
  - Aglomeração
  - Mapeamento
- Escolher recursos computacionais
  - CPUs, Aceleradores?, múltiplos nós?
- Escolher paradigma/linguagem/ferramenta
  - MPI/OpenMP/OpenCL/CUDA



Programando uma aplicação Paralela

Algumas considerações durante o desenvolvimento

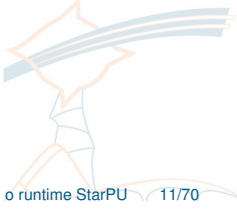
- Se alterar os recursos utilizados devo reescrever o programa?



Programando uma aplicação Paralela

Algumas considerações durante o desenvolvimento

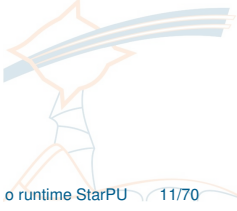
- Se alterar os recursos utilizados devo reescrever o programa?
- Se alterar a carga devo reescrever o programa?



Programando uma aplicação Paralela

Algumas considerações durante o desenvolvimento

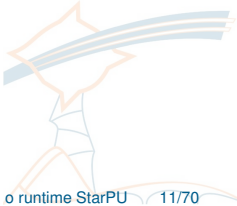
- Se alterar os recursos utilizados devo reescrever o programa?
- Se alterar a carga devo reescrever o programa?
- Como descobrir quanto utilizar de cada recurso?



Programando uma aplicação Paralela

Algumas considerações durante o desenvolvimento

- Se alterar os recursos utilizados devo reescrever o programa?
- Se alterar a carga devo reescrever o programa?
- Como descobrir quanto utilizar de cada recurso?
- A heterogeneidade no meu caso é benéfica?



Programando uma aplicação Paralela

Algumas considerações durante o desenvolvimento

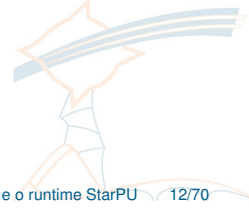
- Se alterar os recursos utilizados devo reescrever o programa?
- Se alterar a carga devo reescrever o programa?
- Como descobrir quanto utilizar de cada recurso?
- A heterogeneidade no meu caso é benéfica?
- Quão fácil é incluir um novo acelerador na minha implementação?

Programação paralela orientada a tarefas

Com o paradigma orientado a Tarefas:

A aplicação é subdivida em Tarefas bem definidas

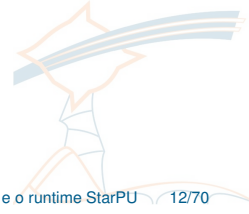
- *Kernels*, fases, iterações, operações são descritas como tarefas



Com o paradigma orientado a Tarefas:

A aplicação é subdivida em Tarefas bem definidas

- *Kernels*, fases, iterações, operações são descritas como tarefas
- O fluxo do programa é determinado pela dependência de dados entre tarefas

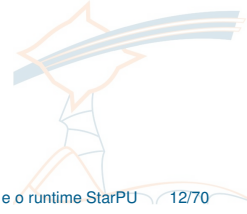


Programação paralela orientada a tarefas

Com o paradigma orientado a Tarefas:

A aplicação é subdivida em Tarefas bem definidas

- *Kernels*, fases, iterações, operações são descritas como tarefas
- O fluxo do programa é determinado pela dependência de dados entre tarefas
- Aplicação executa com o auxílio de um *runtime*

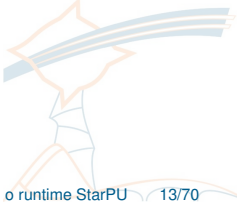


Paradigma de Programação Paralela Orientado a Tarefas

Paradigma de Programação Paralela Orientado a Tarefas

Também chamado de *task-based programming* ou *data flow scheduling*.

- Popularidade crescente
- Desenvolvimento atual de alguns *runtimes*



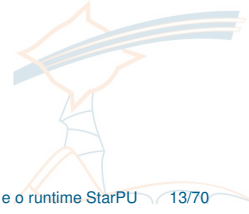
Paradigma de Programação Paralela Orientado a Tarefas

Também chamado de *task-based programming* ou *data flow scheduling*.

- Popularidade crescente
- Desenvolvimento atual de alguns *runtimes*

Estratégia mais declarativa

- Estrutura do programa é submetida a um *runtime*
- Estrutura é organizada em um DAG (Grafo acíclico dirigido)



Paradigma de Programação Paralela Orientado a Tarefas

Também chamado de *task-based programming* ou *data flow scheduling*.

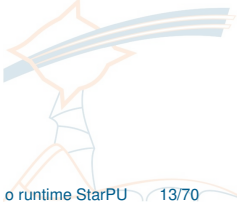
- Popularidade crescente
- Desenvolvimento atual de alguns *runtimes*

Estratégia mais declarativa

- Estrutura do programa é submetida a um *runtime*
- Estrutura é organizada em um DAG (Grafo acíclico dirigido)

Runtime toma decisões (e possíveis otimizações):

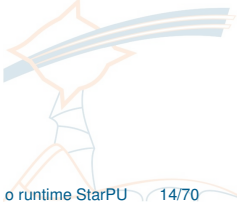
- Escalonamento
- Transferência de memória



Vantagens e Desvantagens

Benefícios:

- Redução de complexidade na programação
- Possíveis otimizações são tomadas pelo *runtime*
- Comportamento estocástico



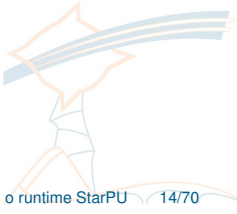
Vantagens e Desvantagens

Benefícios:

- Redução de complexidade na programação
- Possíveis otimizações são tomadas pelo *runtime*
- Comportamento estocástico

Desvantagens:

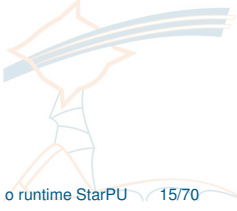
- Mais uma camada de complexidade (*Runtime*)
- Perda do controle de algumas operações
- Comportamento estocástico



Estrutura de uma aplicação

Representada por um DAG, elementos:

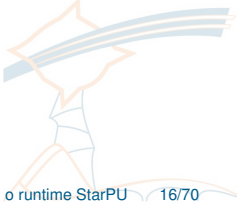
1. Tarefas
2. Blocos de Dados
3. Dependências



Tarefas

Unidade de computação, usualmente código sequencial ou *kernel* em um acelerador

- Atua sobre blocos de dados bem definidos
- Pode possuir múltiplas implementações



Tarefas

Unidade de computação, usualmente código sequencial ou *kernel* em um acelerador

- Atua sobre blocos de dados bem definidos
- Pode possuir múltiplas implementações

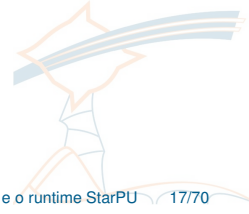
Exemplos:

- Multiplicação de blocos (*Tiles*) de uma matriz
- Aplicação de uma etapa (*Equação*) de um modelo de simulação

Blocos de dados

Subdivisão do domínio a ser trabalhado

- Utilizado como entrada e resultado das tarefas
- Blocos podem ser Lidos/Escritos



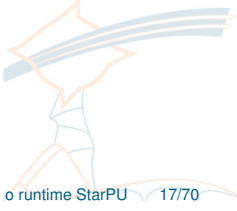
Blocos de dados

Subdivisão do domínio a ser trabalhado

- Utilizado como entrada e resultado das tarefas
- Blocos podem ser Lidos/Escritos

Exemplos:

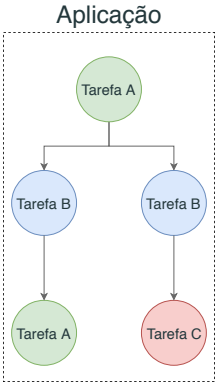
- Sub-blocos (*Tiles*) de uma matriz
- Subconjunto de elementos a serem simulados



Dependências entre tarefas

Dependências entre tarefas resultantes da utilização dos mesmos dados.

- Tarefa do tipo A gera dados que serão utilizados por duas Tarefas do tipo B



Como explicitar um algoritmo para a orientação a tarefas?

Discretizar o algoritmo/problema em tarefas/dados

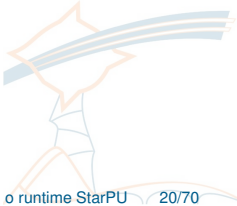
- Assim como no modelo PCAM (Particionamento, Comunicação, Aglomeração, Mapeamento)
- Discretizar um algoritmo para o paradigma orientado a tarefas nem sempre é um processo trivial
  - Nem todos os algoritmos são bons candidatos para tal conversão
  - Entretanto, isso não significa que os problemas não admitam algoritmos que utilizem este paradigma de forma eficiente.

Exemplo de construção: Encontrar maior elemento

Problema:
Dado um vetor de elementos, encontrar o maior elemento

Como construir:

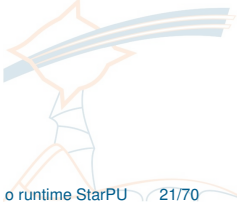
- Possíveis tarefas?
- Possíveis dados?
- Possíveis dependências?



Exemplo de construção: Encontrar maior elemento

Tarefa
Encontrar o maior valor em um subvetor

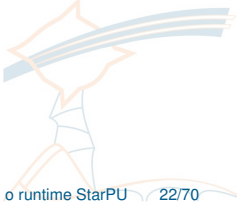
- Recebe um vector de tamanho N
- Resultado: maior elemento
- Chamaremos de MD3 (Maior de 3)



Exemplo de construção: Encontrar maior elemento

Dados
Subvetores do vetor original, de tamanho N

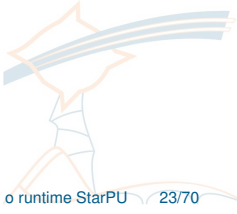
- Para nosso caso, vamos definir como 3.
- Possibilidade de unir os resultados das tarefas em um novo subvetor.



Exemplo de construção: Encontrar maior elemento

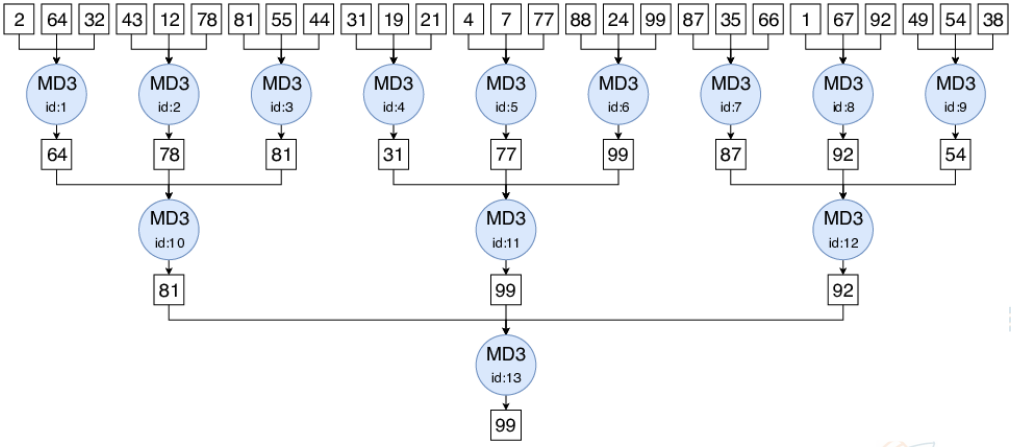
Dependências
Dado um vetor de 27 elementos, podemos executar nossa tarefa MD3 nove vezes.

- O resultado será 9 elementos distintos.
- Dos nove elementos podemos formar novos 3 subvetores
- Podemos reaplicar nossa tarefa MD3 nos resultados



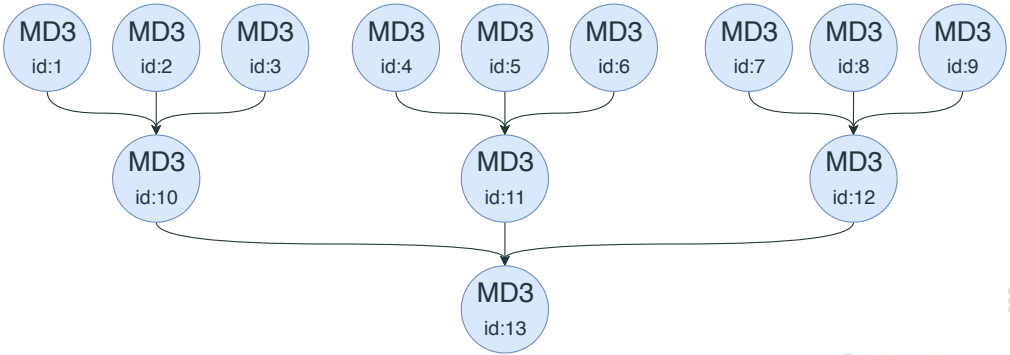
Exemplo de construção: Encontrar maior elemento

Grafo de execução



Exemplo de construção: Encontrar maior elemento

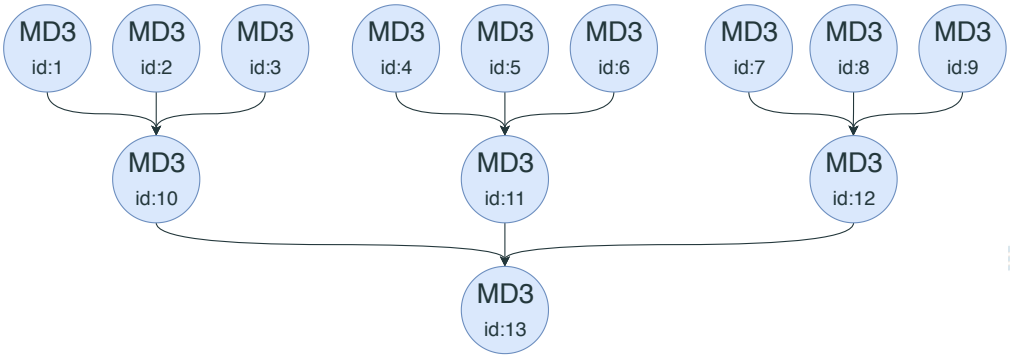
DAG



Exemplo de construção: Encontrar maior elemento

Oportunidades de Paralelismo

- Tarefas que não utilizam mesmos dados e estão com dependências prontas

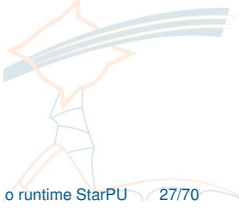


Runtimes

Runtimes: Auxiliam na execução de aplicações

Muito utilizados na área de HPC:

- OpenMP
- MPI



Runtimes

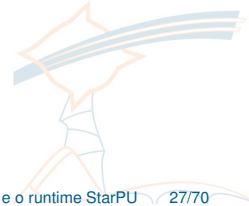
Runtimes: Auxiliam na execução de aplicações

Muito utilizados na área de HPC:

- OpenMP
- MPI

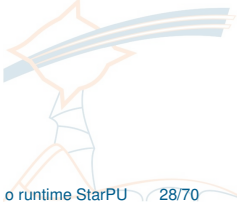
Exemplos de *runtimes* para tarefas:

- PaRSEC
- OmpSs
- OpenMP >4.0
- XKaapi
- StarPU



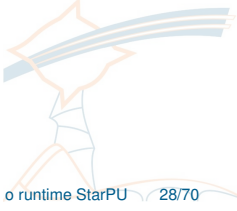
Qual a vantagem?

- Realizar operações automaticamente



Qual a vantagem?

- Realizar operações automaticamente
  - Realizar diversas otimizações baseado no DAG.
  - Mapeamento das tarefas para os recursos
  - Identificar as oportunidades de paralelismo
  - Realizar comunicações automaticamente

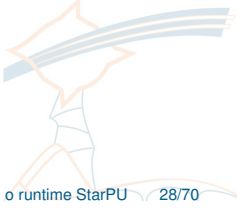


Qual a vantagem?

- Realizar operações automaticamente
  - Realizar diversas otimizações baseado no DAG.
  - Mapeamento das tarefas para os recursos
  - Identificar as oportunidades de paralelismo
  - Realizar comunicações automaticamente

Qual a desvantagem?

- Overhead
- Perda de controle

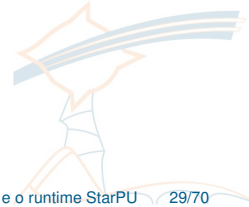


StarPU

StarPU

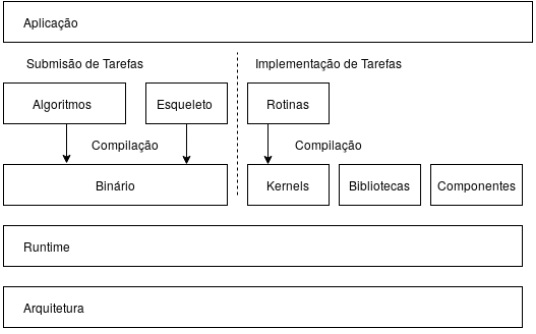
Runtime orientado a tarefas de propósito geral

- Programação de aplicações paralelas para computadores heterogêneos *multicore*.
- Desenvolvido na *Université de Bordeaux* / INRIA.
- Interface (Linguagem `C/C++/Fortran`) para que aplicações submetam suas tarefas em recursos.



Localização do StarPU

Pilha de software de uma aplicação paralela orientada a tarefas e a localização da aplicação e do *runtime* sobre a arquitetura da máquina.

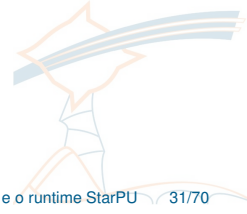


Fonte: Adaptado de Thibault, 2018

Modelo STF

O StarPU usa o modelo STF (*Sequential Task Flow*)

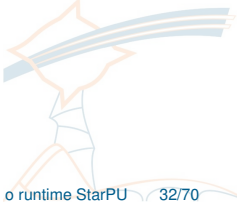
- A aplicação submete sequencialmente as tarefas durante a execução e o *runtime* dinamicamente as escalona para os recursos.
- O DAG não precisa ser inteiramente conhecido
- Tarefas podem ir sendo submetidas gradativamente/dinamicamente



Trabalhadores

Entidade de processamento

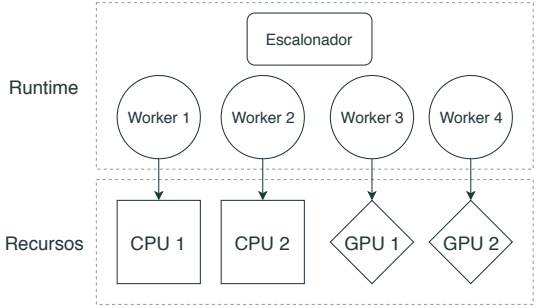
- Entidade que realiza a execução de uma tarefa



Trabalhadores

Entidade de processamento

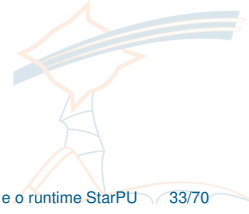
- Entidade que realiza a execução de uma tarefa
- Cada recurso computacional tem um ou mais trabalhadores



Implementações

Recursos heterogêneos requerem implementações diferentes

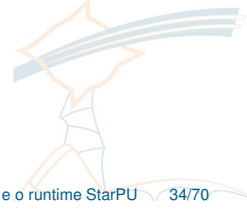
- Tarefas são estruturadas no codelet que podem possuir uma implementação por tipo de recurso
- A decisão de qual implementação utilizar é feita pelo *runtime* durante a execução



Estruturas da programação orientada a Tarefas no StarPU

As estruturas são descritas da seguinte forma:

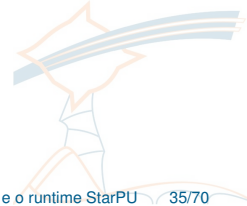
- Tarefas são descritas em codelets
- Blocos de memória são descritos em data handles
- Dependências são implicitamente calculadas pela reutilização de dados e pela ordem de submissão



Escalonamento

Múltiplos escalonadores disponíveis:

- `lws` (local work-stealing)
- `eager` (fila centralizada de tarefas)
- `dm` (deque model) baseado no algoritmo heft
- `dmda` (deque model data aware)

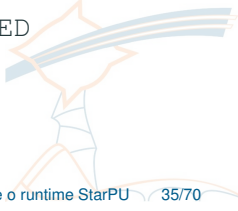


Escalonamento

Múltiplos escalonadores disponíveis:

- `lws` (local work-stealing)
- `eager` (fila centralizada de tarefas)
- `dm` (deque model) baseado no algoritmo heft
- `dmda` (deque model data aware)

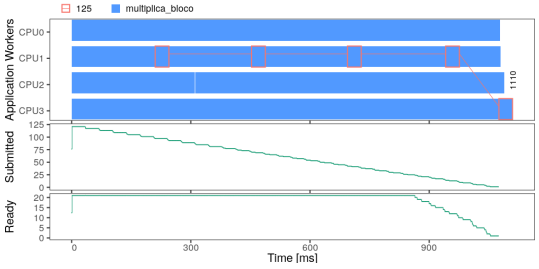
Escalonador é configurável via variável de ambiente `STARPU_SCHED`



Análise de Desempenho e Rastros

A análise de desempenho das aplicações em StarPU pode ser feita utilizando rastros:

- Formato FxT utilizado para gravar diversos eventos
- Pode ser convertido para outros formatos como Pajé



Como construir seu primeiro programa

Instalação utilizando spack

Instalando spack

```
git clone https://github.com/spack/spack.git
source spack/share/spack/setup-env.sh
```

Adicionando Repositório solverstack

```
git clone https://gitlab.inria.fr/solverstack/spack-repo.git
spack repo add spack-repo/
```

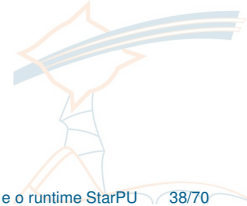
Instalação do StarPU e Ambiente

Instalar StarPU sem CUDA

```
spack install starpu@1.3.1~fast+fxt~mpi~cuda~openmp
```

Instalar StarPU com CUDA

```
spack install starpu@1.3.1~fast+fxt~mpi+cuda~openmp ^cuda@10.0.130
```



Instalação do StarPU e Ambiente

Configurar Ambiente

```
export STARPU_HOME=$(pwd)/starpu
export PKG_CONFIG_PATH=$STARPU_HOME/pkgconfig:\
       $STARPU_HOME/lib/pkgconfig:$PKG_CONFIG_PATH
export LD_LIBRARY_PATH=$STARPU_HOME/lib:\
       $STARPU_HOME/lib64:$LD_LIBRARY_PATH
```

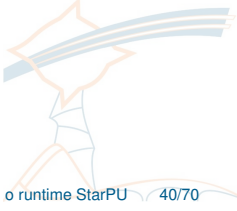
Criar diretório de instalação do StarPU

```
spack view soft $STARPU_HOME starpu@1.3.1
```

Primeiro Programa: Hello World

Programa na Linguagem C que utiliza a interface do StarPU

Material: <https://gitlab.com/lnesi/minicurso-starpu-erad-2020>



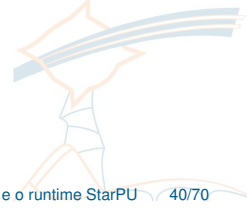
Primeiro Programa: Hello World

Programa na Linguagem C que utiliza a interface do StarPU

Material: <https://gitlab.com/lnesi/minicurso-starpu-erad-2020>

Incluir biblioteca `starpu.h`.

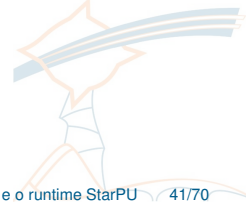
```
1 #include <starpu.h>
```



Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

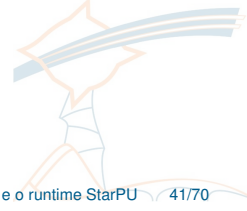


Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

- Inicializa o StarPU

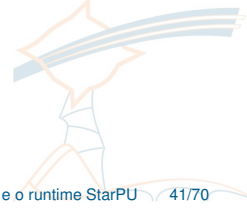


Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

- Mostra a topologia dos recursos

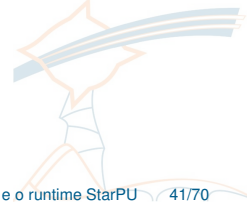


Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

- Insere uma tarefa descrita pelo codelet `codelet_world`

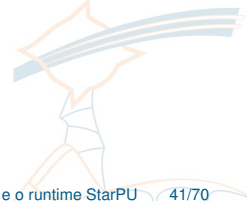


Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

- Espera por todas as tarefas terminarem

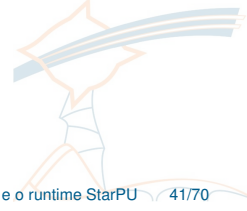


Primeiro Programa: Hello World

Criando o *main* de uma aplicação em StarPU:

```
1 int main(){
2     starpu_init(NULL);
3     starpu_topology_print(stdout);
4     starpu_task_insert(&codelet_world, 0);
5     starpu_task_wait_for_all();
6     starpu_shutdown();
7 }
```

- Desliga o StarPU



Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Utilizar a estrutura starpu-codelet e criar um codelet

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Definir a função da implementação CPU

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Definir a quantidade de blocos de dados que a tarefa usa

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Definir o nome da tarefa

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Definir a função em CPU, assinatura: `void fun(void *b[], void *a)`

Primeiro Programa: Definição de Tarefa

Definindo a tarefa que imprime "Hello World!"

```
1  void func_cpu(void *buffers[], void *args)
2  {
3      printf("Hello World!\n");
4  }
5
6  struct starpu_codelet codelet_world =
7  {
8      .cpu_funcs = { func_cpu },
9      .nbuffers = 0,
10     .name = "hello_world",
11 };
```

- Imprimir Hello World

Compilação da aplicação

Com as variáveis de ambiente bem definidas podemos utilizar:

StarPU com CUDA

```
1 gcc $(pkg-config --cflags starpu-1.3 cuda-10.0) ./hello_world.c\
2     $(pkg-config --libs starpu-1.3 cuda-10.0) -o hw
```

StarPU sem CUDA

```
1 gcc $(pkg-config --cflags starpu-1.3) ./hello_world.c\
2     $(pkg-config --libs starpu-1.3) -o hw
```

Hello World: Resultado

Executar nossa aplicação:

```
./hw
```

Resultado da nossa aplicação:

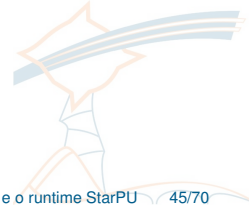
```
  pack 0  core 0  PU 0  CPU 0
      PU 1
    core 1  PU 2  CPU 1
      PU 3
    core 2  PU 4  CPU 2
      PU 5
    core 3  PU 6  CUDA 0.0 (GeForce GTX 1080 7.1 GiB 01:00.0)
      PU 7
Hello World!
```

Exemplo: Soma de Matrizes

Soma de Matrizes

Realizar a soma das matrizes A e B na matriz C.

- Divisão das matrizes em blocos menores



Descritor de funções (*codelet*)

A Tarefa no nosto caso realizará a função $C_{ij} = A_{ij} + B_{ij}$, recebendo cada bloco
Configurando o Codelet:

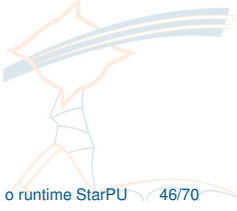
```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

Descritor de funções (*codelet*)

Configurando o Codelet:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

- nbuffers será 3, blocos C, A, B.



Descritor de funções (*codelet*)

Configurando o Codelet:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

Devemos definir os tipos de acesso:

- STARPU_R para somente leitura, STARPU_W para somente escrita.

Descritor de funções (*codelet*)

Configurando o Codelet:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

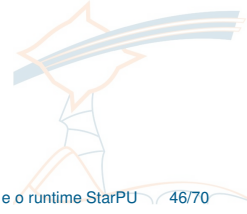
- Neste caso, se queremos os dois primeiros buffers como leitura e o terceiro como escrita, utilizamos STARPU_R, STARPU_R, STARPU_W

Descritor de funções (*codelet*)

Configurando o Codelet:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

- Nome da tarefa.

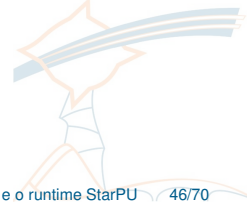


Descritor de funções (*codelet*)

Configurando o Codelet:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_W },
6     .name = "soma_bloco"
7 };
```

- Implementação em CPU.



Acessando Blocos memória na implementação CPU

Programando a implementação em CPU:

```
1 void func_cpu(void *buffers[], void *args)
2 {
3     float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4     float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5     float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6     for(int i=0; i < BLOCK_TOTAL_SIZE; i++){
7         C[i] = A[i] + B[i];
8     }
9 }
```

Acessando Blocos memória na implementação CPU

Programando a implementação em CPU:

```
1 void func_cpu(void *buffers[], void *args)
2 {
3     float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4     float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5     float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6     for(int i=0; i < BLOCK_TOTAL_SIZE; i++){
7         C[i] = A[i] + B[i];
8     }
9 }
```

- Assinatura da função: void func(void *buffers[], void *args)

Acessando Blocos memória na implementação CPU

Programando a implementação em CPU:

```
1 void func_cpu(void *buffers[], void *args)
2 {
3     float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4     float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5     float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6     for(int i=0; i < BLOCK_TOTAL_SIZE; i++){
7         C[i] = A[i] + B[i];
8     }
9 }
```

- Utilizar STARPU_VECTOR_GET_PTR no buffer correspondente
- Nosso caso: A, B, C (ordem importante)

Acessando Blocos memória na implementação CPU

Programando a implementação em CPU:

```
1 void func_cpu(void *buffers[], void *args)
2 {
3     float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4     float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5     float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6     for(int i=0; i < BLOCK_TOTAL_SIZE; i++){
7         C[i] = A[i] + B[i];
8     }
9 }
```

- Realizar a operação sequencialmente

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

- Armazenamos os ponteiros para nosso blocos da matriz

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

- Temos um vetor de mesmo tamanho da estrutura `starpu_data_handle_t`

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

- Para cada bloco realizaremos dois passos

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

- 1. Alocação normal do bloco

Criação de Descritor de dados (*data handle*)

Para as tarefas utilizarem os blocos de memória, devemos registrá-los no StarPU na forma de `starpu_data_handle_t`.

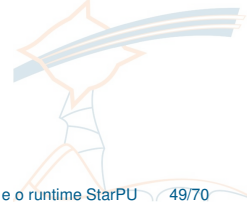
```
1 float* matrix_a[NUMBER_BLOCKS];
2 starpu_data_handle_t matrix_a_handle[NUMBER_BLOCKS];
3 for(int i=0; i<NUMBER_BLOCKS; i++){
4     matrix_a[i]=(float*)malloc(WIDTH * WIDTH * sizeof(float));
5     starpu_matrix_data_register(&matrix_a_handle[i], STARPU_MAIN_RAM,
6         (uintptr_t)matrix_a[i], WIDTH, WIDTH, WIDTH, sizeof(float));
7 }
```

- 2. Registro do bloco em um data_handle no starpu

Inserindo Tarefas

Devemos submeter uma tarefa para cada coordenada de bloco realizar a operação $bloco_c[i] = bloco_a[i] + bloco_b[i]$:

```
1 for(int i=0; i<NUMBER_BLOCKS; i++){
2    starpu_task_insert(&codelet_soma, STARPU_R, matrix_a_handle[i],
3     STARPU_R, matrix_b_handle[i], STARPU_W, matrix_c_handle[i], 0);
4 }
```

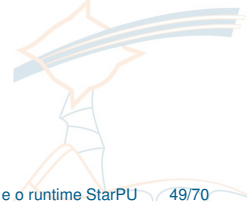


Inserindo Tarefas

Devemos submeter uma tarefa para cada coordenada de bloco realizar a operação $bloco_c[i] = bloco_a[i] + bloco_b[i]$:

```
1 for(int i=0; i<NUMBER_BLOCKS; i++){
2    starpu_task_insert(&codelet_soma, STARPU_R, matrix_a_handle[i],
3     STARPU_R, matrix_b_handle[i], STARPU_W, matrix_c_handle[i], 0);
4 }
```

- Para cada coordenada de bloco

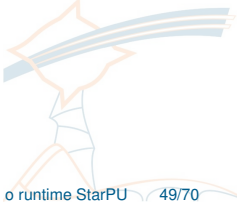


Inserindo Tarefas

Devemos submeter uma tarefa para cada coordenada de bloco realizar a operação $bloco_c[i] = bloco_a[i] + bloco_b[i]$:

```
1 for(int i=0; i<NUMBER_BLOCKS; i++){
2    starpu_task_insert(&codelet_soma, STARPU_R, matrix_a_handle[i],
3     STARPU_R, matrix_b_handle[i], STARPU_W, matrix_c_handle[i], 0);
4 }
```

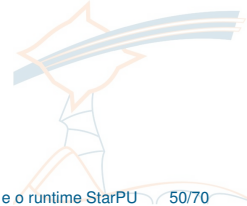
- Inserir uma tarefa do tipo "codelet_soma",
- Passando os blocos (handles) A, B, C



De-inicializando blocos de memória

Antes da finalização do starpu, com `starpu_shutdown` é necessário desalocar os `data_handles` utilizando a função `starpu_data_unregister()`.

```
1 for(int i=0; i < NUMBER_BLOCKS; i++){
2     starpu_data_unregister(matrix_a_handle[i]);
3     starpu_data_unregister(matrix_b_handle[i]);
4     starpu_data_unregister(matrix_c_handle[i]);
5 }
```

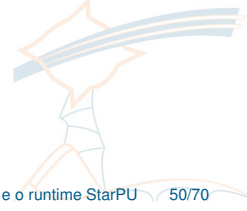


De-inicializando blocos de memória

Antes da finalização do starpu, com `starpu_shutdown` é necessário desalocar os `data_handles` utilizando a função `starpu_data_unregister()`.

```
1 for(int i=0; i < NUMBER_BLOCKS; i++){
2     starpu_data_unregister(matrix_a_handle[i]);
3     starpu_data_unregister(matrix_b_handle[i]);
4     starpu_data_unregister(matrix_c_handle[i]);
5 }
```

- Para cada bloco

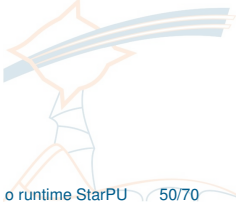


De-inicializando blocos de memória

Antes da finalização do starpu, com `starpu_shutdown` é necessário desalocar os `data_handles` utilizando a função `starpu_data_unregister()`.

```
1 for(int i=0; i < NUMBER_BLOCKS; i++){
2     starpu_data_unregister(matrix_a_handle[i]);
3     starpu_data_unregister(matrix_b_handle[i]);
4     starpu_data_unregister(matrix_c_handle[i]);
5 }
```

- Desalocar o bloco das matrizes A, B, C

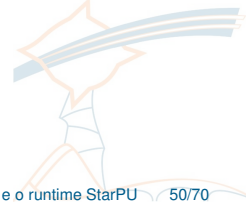


De-inicializando blocos de memória

Antes da finalização do starpu, com `starpu_shutdown` é necessário desalocar os `data_handles` utilizando a função `starpu_data_unregister()`.

```
1 for(int i=0; i < NUMBER_BLOCKS; i++){
2     starpu_data_unregister(matrix_a_handle[i]);
3     starpu_data_unregister(matrix_b_handle[i]);
4     starpu_data_unregister(matrix_c_handle[i]);
5 }
```

- Igualmente devemos realizar o free() de alocações anteriores!



Heterogeneidade e implementações em GPU

Adicionando uma implementação em GPU CUDA em uma Tarefa:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .cuda_funcs = { func_gpu },
5     .nbuffers = 3,
6     .modes = { STARPU_R, STARPU_R, STARPU_W },
7     .name = "soma_bloco",
8     .where = STARPU_CUDA
9 };
```

Heterogeneidade e implementações em GPU

Adicionando uma implementação em GPU CUDA em uma Tarefa:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .cuda_funcs = { func_gpu },
5     .nbuffers = 3,
6     .modes = { STARPU_R, STARPU_R, STARPU_W },
7     .name = "soma_bloco",
8     .where = STARPU_CUDA
9 };
```

- Mesma estrutura

Heterogeneidade e implementações em GPU

Adicionando uma implementação em GPU CUDA em uma Tarefa:

```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .cuda_funcs = { func_gpu },
5     .nbuffers = 3,
6     .modes = { STARPU_R, STARPU_R, STARPU_W },
7     .name = "soma_bloco",
8     .where = STARPU_CUDA
9 };
```

- Adicação da função para executar kernels em GPU

Heterogeneidade e implementações em GPU

Adicionando uma implementação em GPU CUDA em uma Tarefa:

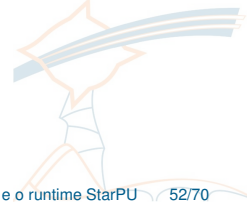
```
1 struct starpu_codelet codelet_soma =
2 {
3     .cpu_funcs = { func_cpu },
4     .cuda_funcs = { func_gpu },
5     .nbuffers = 3,
6     .modes = { STARPU_R, STARPU_R, STARPU_W },
7     .name = "soma_bloco",
8     .where = STARPU_CUDA
9 };
```

- Marcador para obrigar a execução em GPU

Implementação GPU

Podemos utilizar bibliotecas da API do CUDA, como cuBLAS.

```
1 #include <cuda_runtime.h>
2 #include <cublas_v2.h>
3 #define CHECK_CUBLAS(x) if(x!=CUBLAS_STATUS_SUCCESS){printf("Cublass
      error: %d\n", x);};
4 cublasHandle_t cublas_mainhandle;
```

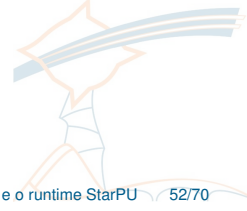


Implementação GPU

Podemos utilizar bibliotecas da API do CUDA, como cuBLAS.

```
1 #include <cuda_runtime.h>
2 #include <cublas_v2.h>
3 #define CHECK_CUBLAS(x) if(x!=CUBLAS_STATUS_SUCCESS){printf("Cublass
      error: %d\n", x);};
4 cublasHandle_t cublas_mainhandle;
```

- Mesma situação que uma aplicação comum CUDA



Implementação GPU - Função executada

Uma função em CPU que invoca as chamadas CUDA necessárias.

```
1  void func_gpu(void *buffers[], void *args){
2      float alpha_beta = 1;
3      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6      cublasSetStream(cublas_mainhandle, starpu_cuda_get_local_stream());
7      CHECK_CUBLAS(cublasSgeam(cublas_mainhandle, CUBLAS_OP_N, CUBLAS_OP_N,
8                  WIDTH, WIDTH, &alpha_beta, A, WIDTH, &alpha_beta,
9                  B, WIDTH, C, WIDTH));
10     cudaStreamSynchronize(starpu_cuda_get_local_stream());
11 }
```

Implementação GPU - Função executada

Uma função em CPU que invoca as chamadas CUDA necessárias.

```
1  void func_gpu(void *buffers[], void *args){
2      float alpha_beta = 1;
3      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6      cublasSetStream(cublas_mainhandle, starpu_cuda_get_local_stream());
7      CHECK_CUBLAS(cublasSgeam(cublas_mainhandle, CUBLAS_OP_N, CUBLAS_OP_N,
8                  WIDTH, WIDTH, &alpha_beta, A, WIDTH, &alpha_beta,
9                  B, WIDTH, C, WIDTH));
10     cudaStreamSynchronize(starpu_cuda_get_local_stream());
11 }
```

- Mesma assinatura da função em CPU

Implementação GPU - Função executada

Uma função em CPU que invoca as chamadas CUDA necessárias.

```
1  void func_gpu(void *buffers[], void *args){
2      float alpha_beta = 1;
3      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6      cublasSetStream(cublas_mainhandle, starpu_cuda_get_local_stream());
7      CHECK_CUBLAS(cublasSgeam(cublas_mainhandle, CUBLAS_OP_N, CUBLAS_OP_N,
8                  WIDTH, WIDTH, &alpha_beta, A, WIDTH, &alpha_beta,
9                  B, WIDTH, C, WIDTH));
10     cudaStreamSynchronize(starpu_cuda_get_local_stream());
11 }
```

- Acessamos os dados da mesma maneira que em CPU
- Os Ponteiros obtidos são de memória já em GPU!

Implementação GPU - Função executada

Uma função em CPU que invoca as chamadas CUDA necessárias.

```
1  void func_gpu(void *buffers[], void *args){
2      float alpha_beta = 1;
3      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6      cublasSetStream(cublas_mainhandle, starpu_cuda_get_local_stream());
7      CHECK_CUBLAS(cublasSgeam(cublas_mainhandle, CUBLAS_OP_N, CUBLAS_OP_N,
8                  WIDTH, WIDTH, &alpha_beta, A, WIDTH, &alpha_beta,
9                  B, WIDTH, C, WIDTH));
10     cudaStreamSynchronize(starpu_cuda_get_local_stream());
11 }
```

- Invocamos o kernel ou a função em CUDA

Implementação GPU - Função executada

Uma função em CPU que invoca as chamadas CUDA necessárias.

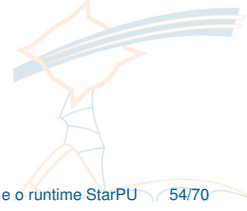
```
1  void func_gpu(void *buffers[], void *args){
2      float alpha_beta = 1;
3      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
4      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
5      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
6      cublasSetStream(cublas_mainhandle, starpu_cuda_get_local_stream());
7      CHECK_CUBLAS(cublasSgeam(cublas_mainhandle, CUBLAS_OP_N, CUBLAS_OP_N,
8                  WIDTH, WIDTH, &alpha_beta, A, WIDTH, &alpha_beta,
9                  B, WIDTH, C, WIDTH));
10     cudaStreamSynchronize(starpu_cuda_get_local_stream());
11 }
```

- Cuidado com chamadas assíncronas!

Compilando Com GPU

Para compilar agora com funções cuda devemos compilar normalmente utilizando as bibliotecas em CUDA.

```
1 gcc $(pkg-config --cflags starpu-1.3 cuda-10.0) \
2     ./soma_matrix_cuda.c \
3     $(pkg-config --libs starpu-1.3 cudart-10.0 cublas-10.0)
```

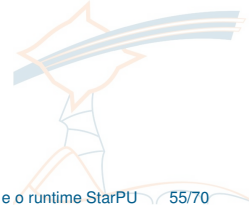


Exemplo: Multiplicação de Matrizes

Multiplicação de matrizes - Algoritmo

Para realizar a multiplicação de matrizes podemos utilizar a versão do algoritmo baseada em blocos. Sendo k a largura do bloco, o bloco $C[i][j]$ é por:

$$C[i][j] = SUM(A[k][j] \times B[i][k]) \quad (1)$$



Multiplicação de matrizes - Tarefas

Tarefa pode computar $C_{ij} = C_{ij} + A_{kj} \times B_{ik}$.

- Tarefas que atualizem o mesmo C não podem executar ao mesmo tempo.

Estrutura do codelet:

```
1 struct starpu_codelet codelet_multi =
2 {
3     .cpu_funcs = { block_mm_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_RW },
6   .where = STARPU_CPU,
7     .name = "multiplica_por_bloco"
8 };
```

- Mesma estrutura

Multiplicação de matrizes - Tarefas

Tarefa pode computar $C_{ij} = C_{ij} + A_{kj} \times B_{ik}$.

- Tarefas que atualizem o mesmo C não podem executar ao mesmo tempo.

Estrutura do codelet:

```
1 struct starpu_codelet codelet_multi =
2 {
3     .cpu_funcs = { block_mm_cpu },
4     .nbuffers = 3,
5     .modes = { STARPU_R, STARPU_R, STARPU_RW },
6   .where = STARPU_CPU,
7     .name = "multiplica_por_bloco"
8 };
```

- Mudamos o nome e a chamda de função

Multiplicação de matrizes - Tarefas

Função a ser invocada:

```
1  void block_mm_cpu (void *buffers[], void *args){
2      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
3      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
4      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
5
6      for(int i=0; i < WIDTH; i++){
7          for(int j=0; j < WIDTH; j++){
8              for(int k=0; k < WIDTH; k++){
9                  C[j * WIDTH + i] += A[j * WIDTH + k] * B[k * WIDTH + i];
10             }
11         }
12     }
13 }
```

Multiplicação de matrizes - Tarefas

Função a ser invocada:

```
1  void block_mm_cpu (void *buffers[], void *args){
2      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
3      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
4      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
5
6      for(int i=0; i < WIDTH; i++){
7          for(int j=0; j < WIDTH; j++){
8              for(int k=0; k < WIDTH; k++){
9                  C[j * WIDTH + i] += A[j * WIDTH + k] * B[k * WIDTH + i];
10             }
11         }
12     }
13 }
```

- Mesma assinatura

Multiplicação de matrizes - Tarefas

Função a ser invocada:

```
1  void block_mm_cpu (void *buffers[], void *args){
2      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
3      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
4      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
5
6      for(int i=0; i < WIDTH; i++){
7          for(int j=0; j < WIDTH; j++){
8              for(int k=0; k < WIDTH; k++){
9                  C[j * WIDTH + i] += A[j * WIDTH + k] * B[k * WIDTH + i];
10             }
11         }
12     }
13 }
```

- Mesmo acesso de dados

Multiplicação de matrizes - Tarefas

Função a ser invocada:

```
1  void block_mm_cpu (void *buffers[], void *args){
2      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
3      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
4      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
5
6      for(int i=0; i < WIDTH; i++){
7          for(int j=0; j < WIDTH; j++){
8              for(int k=0; k < WIDTH; k++){
9                  C[j * WIDTH + i] += A[j * WIDTH + k] * B[k * WIDTH + i];
10             }
11         }
12     }
13 }
```

- Realizar a operação de MM no bloco

Multiplicação de matrizes - Tarefas

Função a ser invocada:

```
1  void block_mm_cpu (void *buffers[], void *args){
2      float *A = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
3      float *B = (float *)STARPU_VECTOR_GET_PTR(buffers[1]);
4      float *C = (float *)STARPU_VECTOR_GET_PTR(buffers[2]);
5
6      for(int i=0; i < WIDTH; i++){
7          for(int j=0; j < WIDTH; j++){
8              for(int k=0; k < WIDTH; k++){
9                  C[j * WIDTH + i] += A[j * WIDTH + k] * B[k * WIDTH + i];
10             }
11         }
12     }
13 }
```

- Poderia ser uma chamada de alguma biblioteca BLAS

Multiplicação de matrizes - Submissão de Tarefas

Para a submissão de tarefas:

```
for(int i=0; i<NUMBER_BLOCKS_WIDTH; i++){
    for(int j=0; j<NUMBER_BLOCKS_WIDTH; j++){
        for(int k=0; k<NUMBER_BLOCKS_WIDTH; k++){
            starpu_task_insert(&codelet_multi,
                STARPU_R, matrix_a_handle[j * NUMBER_BLOCKS_WIDTH + k],
                STARPU_R, matrix_b_handle[k * NUMBER_BLOCKS_WIDTH + i],
                STARPU_RW, matrix_c_handle[j * NUMBER_BLOCKS_WIDTH + i],
                0);
        }
    }
}
```

Multiplicação de matrizes - Submissão de Tarefas

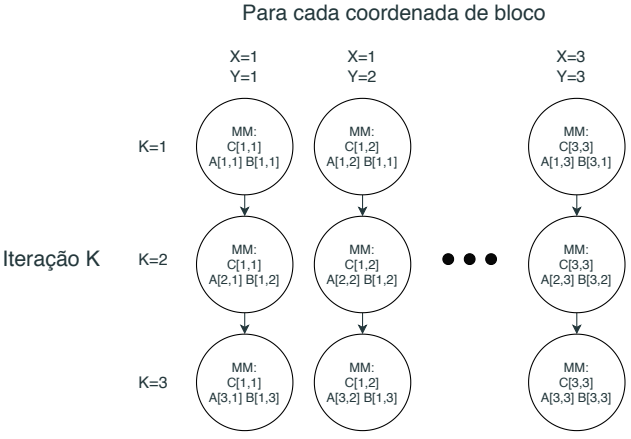
Para a submissão de tarefas:

```
1  for(int i=0; i<NUMBER_BLOCKS_WIDTH; i++){
2      for(int j=0; j<NUMBER_BLOCKS_WIDTH; j++){
3          for(int k=0; k<NUMBER_BLOCKS_WIDTH; k++){
4              starpu_task_insert(&codelet_multi,
5                  STARPU_R, matrix_a_handle[j * NUMBER_BLOCKS_WIDTH + k],
6                  STARPU_R, matrix_b_handle[k * NUMBER_BLOCKS_WIDTH + i],
7                  STARPU_RW, matrix_c_handle[j * NUMBER_BLOCKS_WIDTH + i],
8                  0);
9          }
10     }
11 }
```

- Chamando as tarefas para cada MM

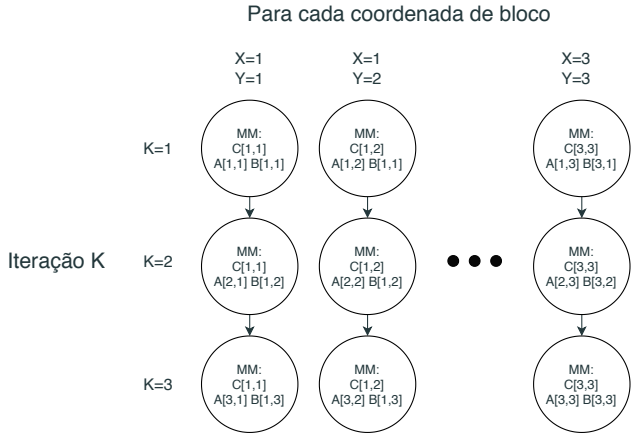
Multiplicação de matrizes - DAG da aplicação

O DAG desta implementação:



Multiplicação de matrizes - DAG da aplicação

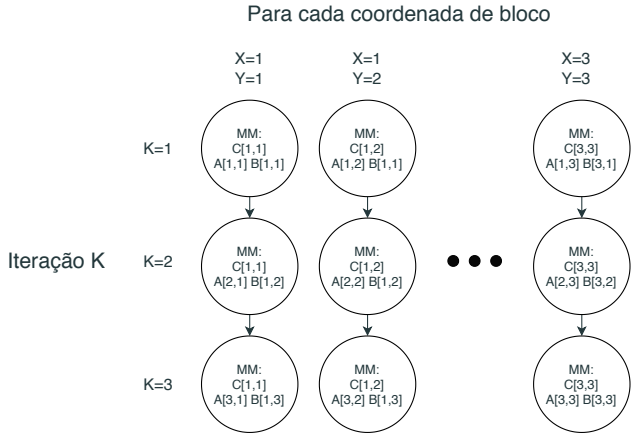
O DAG desta implementação:



Dependencias nesta ordem de K porque a submissão foi feita assim

Multiplicação de matrizes - DAG da aplicação

O DAG desta implementação:



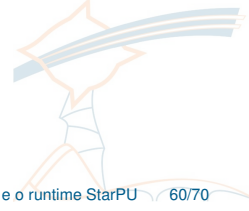
Poderia ser diferente sem consequencia ao resultado final

Analisando Aplicações StarPU

Métricas

Métricas fundamentais para análise de desempenho de aplicações paralelas

- *Makespan*
- *Speed-up*
- Eficiência



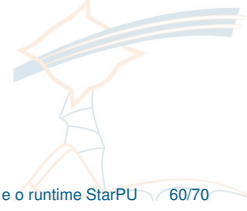
Métricas

Métricas fundamentais para análise de desempenho de aplicações paralelas

- *Makespan*
- *Speed-up*
- Eficiência

Nem sempre são suficientes!

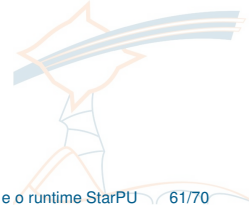
- Métricas podem mascaram comportamentos
- Oportunidades para aplicações orientadas a Tarefas



Visualizações e StarVZ

StarVZ é um workflow para a visualização da execução/desempenho de aplicações baseadas em Tarefas

- Diversos painéis (Visualizações de dados específicos)
- Fortemente suporta StarPU
- Escrito em `R/bash/C++`
- Dividido em 2 fases
  1. Tratamento dos dados
  2. Visualização



Instalação do StarVZ

Para instalar podemos executar o seguinte comando:

```
git clone https://github.com/schnorr/starvz.git
apt install -y r-base libxml2-dev libssl-dev \
    libcurl4-openssl-dev libgit2-dev libboost-dev
./starvz/R/install.R
# pajeng
apt install -y git cmake build-essential \
    libboost-dev asciidoc flex bison
git clone git://github.com/schnorr/pajeng.git
mkdir -p pajeng/b ; cd pajeng/b
cmake ..
make
```

Adquirir rastros da execução StarPU

StarPU deve ter sido compilado com FxT

Ativar variável de ambiente:

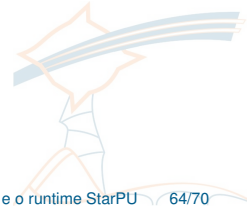
- `STARPU_GENERATE_TRACE=1`
- `STARPU_FXT_PREFIX`: diretório para gravar rastros

```
gcc $(pkg-config --cflags starpu-1.3) ./exemplos/mult_matrix.c \
    $(pkg-config --libs starpu-1.3) -o mult_matrix
STARPU_FXT_PREFIX=$PWD/ STARPU_GENERATE_TRACE=1 ./mult_matrix
```

Execução do StarVZ

Fase 1
Converter arquivos FxT (`prof_*`) para `.feather` (Formato de dados colunar).

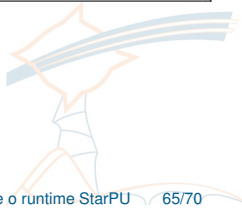
```
export PATH=starvz/:$PATH
export PATH=pajeng/b:$PATH
export PATH=$STARPU_HOME/bin:$PATH
./starvz/src/phase1-workflow.sh ./ ""
```



Fase 2 - Ler Dados e Configuração
Executada em um ambiente R.

Modificar arquivo de configurações `.yaml` para escolher os painéis.

```
library(starvz)
dtrace <- the_fast_reader_function("./")
pajer <- config::get(file = "starvz/full_config.yaml")
```

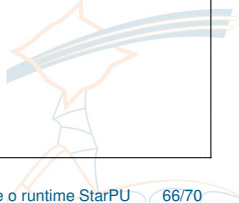


Fase 2 - Alterar algumas configurações

Executada em um ambiente R.

```
pajer$starpu$active = FALSE
# habilita curvas de tarefas submetidas e ativas
pajer$submitted$active = TRUE
pajer$submitted$height  = 1
pajer$ready$active = TRUE
pajer$ready$height  = 1
pajer$st$labels = "ALL"

# seleciona tarefas para desenhar as dependencias
pajer$st$tasks$active = TRUE
pajer$st$tasks$levels = 5
pajer$st$tasks$list = c("125")
pajer$st$height  = 1.8
```

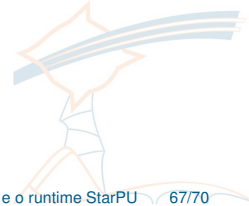


Execução do StarVZ

Fase 2 - Criar visualização

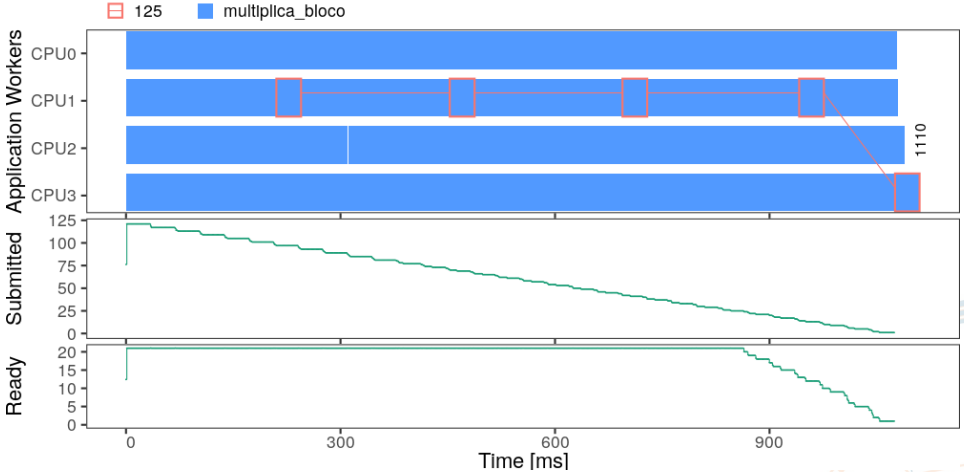
Invocar chamada `starvz_plot()`

```
starvz_plot(dtrace)
```



Exemplo de Visualização StarVZ

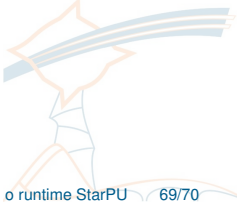
Visualizações: Trabalhadores | Tarefas Submetidas | Tarefas Prontas



Conclusão

Conclusão

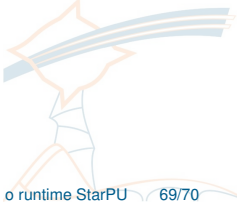
A programação orientada a Tarefas pode auxiliar na programação paralela de suas aplicações



Conclusão

A programação orientada a Tarefas pode auxiliar na programação paralela de suas aplicações

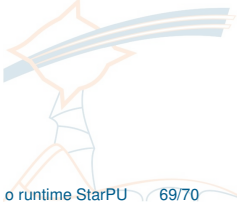
- *Runtime* StarPU pode ser utilizado para experimentar esse paradigma



Conclusão

A programação orientada a Tarefas pode auxiliar na programação paralela de suas aplicações

- *Runtime* StarPU pode ser utilizado para experimentar esse paradigma
- Ferramentas especializadas para auxiliar na analise de desempenho



Obrigado por participar e pela atenção! Perguntas?

Material: <https://gitlab.com/lnesi/minicurso-starpu-erad-2020>

Material Extra: <http://starpu.gforge.inria.fr/tutorials/>

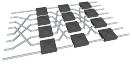
Contato Apresentador: Lucas Leandro Nesi - lucas.nesi@inf.ufrgs.br

Contatos:

Vinícius Garcia Pinto, Marcelo Cogo Miletto, Lucas Mello Schnorr, Samuel Thibault
{vinicius.pinto, marcelo.miletto, schnorr}@inf.ufrgs.br, samuel.thibault@inria.fr

Agradecimentos

Este trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001, do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), e dos projetos: FAPERGS ReDaS (19/711-6), MultiGPU (16/354-8) e GreenCloud (16/488-9), CNPq 447311/2014-0, CAPES/Brafitec 182/15 e CAPES/Cofecub 899/18, e Petrobras (2018/00263-5).



Free Cultural Works

Este documento está licenciado sob a Licença *Atribuição-CompartilhaIgual 4.0 Internacional (CC BY-SA 4.0)* da *Creative Commons* (CC). Em resumo, você deve creditar a obra da forma especificada pelo autor ou licenciante (mas não de maneira que sugira que estes concedem qualquer aval a você ou ao seu uso da obra). Você pode usar esta obra para fins comerciais. Se você alterar, transformar ou criar com base nesta obra, você poderá distribuir a obra resultante apenas sob a mesma licença, ou sob uma licença similar à presente. Para ver uma cópia desta licença, visite <https://creativecommons.org/licenses/by-sa/4.0/>.