

Documentação do MultiExplorer

Laura Barauna Ludgero

Samuel S. Rodrigues

1 Manual de instalação

Para utilizar o MultiExplorer, você precisará do código-fonte, Docker, e XLauncher (versão Windows). Para o fluxo de CPU, é necessária uma versão pré-compilada do simulador Sniper. Java será necessário no fluxo de máquinas virtuais. O Cuda e o GPGPU-Sim será necessário no fluxo de GPUs.

1. Repositório com o código-fonte do MultiExplorer:
<https://github.com/lscad-facom-ufms/multiexplorer>
2. Versão pré-compilada do Sniper:
<https://drive.google.com/file/d/1aXNxy6OZ7NjP1XUgnh0GuTFAUePtwZkW/view>
3. XLauncher:
<https://sourceforge.net/projects/vcxsrv/>
4. Docker:
<https://docs.docker.com/desktop/install/windows-install/>
5. Java:
https://download.oracle.com/java/22/latest/jdk-22_linux-x64_bin.deb
6. Cuda:
<https://developer.nvidia.com/cuda-11.0-download-archive>
7. GPGPU-Sim:
https://github.com/gpgpu-sim/gpgpu-sim_distribution.git

1.1 Código-Fonte

Para obter acesso ao repositório do MultiExplorer basta acessar o link: <https://github.com/lscad-facom-ufms/multiexplorer>. Também é possível solicitar acesso para se tornar colaborador do projeto.

1.2 Para o Fluxo de CPU

1.2.1 Sniper

Após baixar a versão pré-compilada do Sniper, extraia o conteúdo do diretório "snipersim-8.0" para a pasta raiz do código-fonte.

1.2.2 XLaucher

Essa ferramenta tem como objetivo permitir que o Docker acesse a API gráfica do sistema operacional no ambiente Windows. Após baixar e instalar o programa, execute-o. Uma tela será exibida para selecionar o modo de exibição da interface gráfica, com as opções: múltiplas janelas, tela cheia, uma janela ampla ou uma janela sem barra de título. A escolha inicial é livre; para este processo, será selecionada a opção de múltiplas janelas, conforme mostrado na figura 1, e em seguida, clique em avançar.

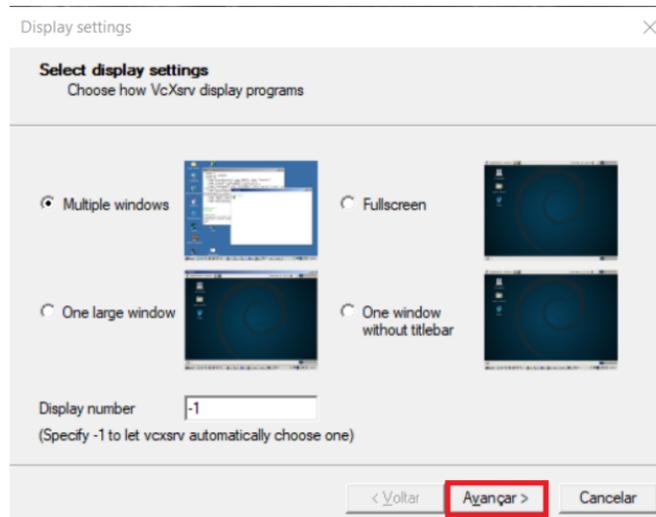


Figura 1

Na próxima janela, selecione a opção “Start no client” e clique em “Avançar”, conforme ilustrado na figura 2.

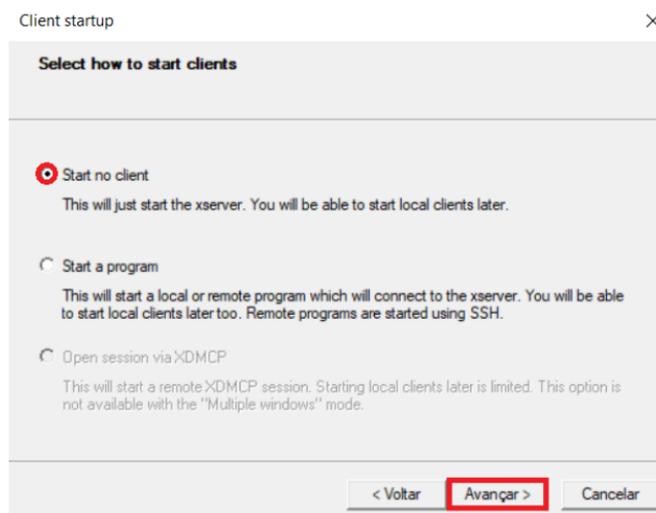


Figura 2

Na etapa seguinte, escolha a opção “Disable Access control” e clique em “Avançar”, como mostrado na figura 3. Esse passo é crucial, pois, caso o controle de acesso esteja ativo, o Docker não terá permissão para acessar a API gráfica do sistema.

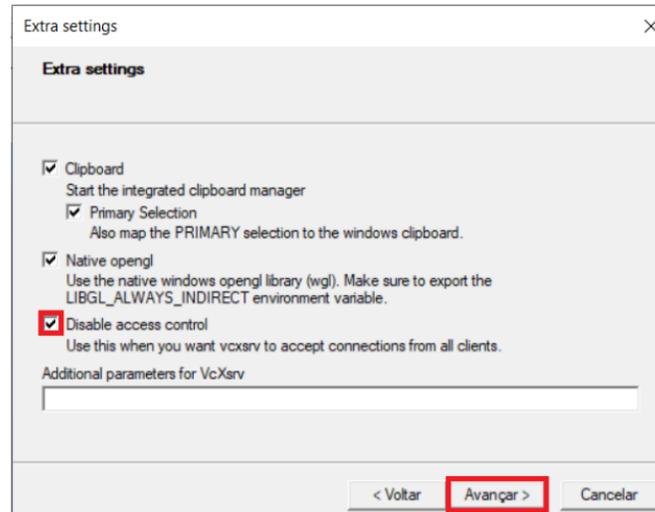


Figura 3

Por último, você pode optar por salvar as configurações feitas anteriormente para reutilizá-las no futuro ou simplesmente clicar em “Concluir”, conforme ilustrado na figura 4.

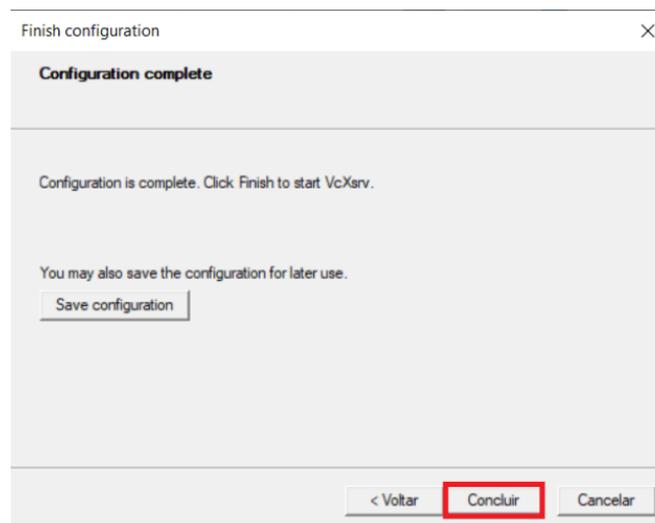


Figura 4

Após concluir essas configurações, a ferramenta estará pronta para ser utilizada na próxima etapa.



1.2.3 Docker

Esta ferramenta tem como objetivo fornecer um ambiente de desenvolvimento e execução uniforme para uma aplicação. Sem ela, seria necessário que o usuário utilizasse obrigatoriamente um sistema Linux e configurasse manualmente os pré-requisitos do **MultiExplorer**.

Com o **Docker** já em funcionamento, será preciso determinar o IP local da máquina (host). Para isso, utilize o comando:

```
ipconfig
```

Anote o endereço IPv4 fornecido no resultado. Depois, no diretório onde o projeto foi clonado, verifique se existe o arquivo ".env". Caso ele não esteja presente, crie-o com o seguinte conteúdo:

```
DISPLAY="IPV4":0.0
```

Substitua "IPV4" pelo endereço IP anotado na etapa anterior. Salve o arquivo na pasta principal do projeto com o nome ".env".

Posteriormente, utilizando o prompt de comando, navegue até o diretório do projeto com o comando:

```
cd + caminho do projeto
```

Depois, execute o comando:

```
docker compose up -d
```

Isso iniciará o processo de construção da imagem do sistema no Docker. Ao finalizar, o ambiente virtual para o uso do MultiExplorer estará configurado. Em seguida, utilize o comando:

```
docker exec -it multiexplorer-dev-1 bash
```

Esse comando permitirá que você acesse o contêiner criado anteriormente. Depois, execute o seguinte comando:

```
make install
```

Esse comando executará o script necessário para construir o software. Durante o processo, pode ser solicitado o preenchimento manual de parâmetros de configuração. Após a conclusão da instalação, execute o comando:

```
make config
```

Com essa etapa finalizada, a plataforma estará pronta para uso. Para abrir a interface gráfica do MultiExplorer, certifique-se de que o **XLauncher** está ativo. Caso esteja, basta executar o seguinte comando:

```
python ME.py
```

Assim, a interface gráfica da ferramenta será exibida e estará pronta para utilização.



1.3 Para fluxo de VM

1.3.1 Java

Após baixar o pacote Debian do **Java** JDK, coloque-o na pasta do código-fonte e execute:

```
dpkg -i "nome-do-pacote".deb
```

Substitua "nome-do-pacote" pelo nome do pacote Debian.

1.4 Para o fluxo de GPUs

Esta seção descreve como simular um modelo de GPU com GPGPU-Sim.

1.4.1 Baixar CUDA versão 11.0 ou anterior.

<https://developer.nvidia.com/cuda-11.0-download-archive>

1.4.2 Clonar o repositório GPGPU-Sim

```
git clone https://github.com/gpgpu-sim/gpgpu-sim_distribution.git
```

O repositório do GPGPU-Sim também pode ser encontrado na pasta MultiExplorer:
/multiexplorer/gpgpu-sim_distribution

Se a pasta estiver vazia, execute o seguinte comando:

```
git submodule update --init --recursive
```

1.4.3 Instalar dependências

```
sudo apt-get install build-essential xutils-dev bison
```

```
↔ zlib1g-dev flex libglu1-mesa-dev
```

```
sudo apt-get install doxygen graphviz
```

```
sudo apt-get install python-pmw python-ply python-numpy
```

```
↔ libpng12-dev python-matplotlib
```

```
sudo apt-get install libxi-dev libxmu-dev libglut3-dev
```

1.4.4 Compilar o GPGPU-Sim

No diretório do GPGPU-Sim (`gpgpu-sim_distribution`), execute:

```
export CUDA_INSTALL_PATH=/usr/local/cuda-11.0/
```

```
↔ (O local do CUDA deve ser atualizado caso seja diferente)
```

```
source setup_environment
```

```
make
```



1.4.5 Executando

Crie uma pasta de execução. Nessa pasta, coloque os arquivos `gpgpusim.config`, XML e `config_fermi.islip.config` (se presentes). Adicionalmente, coloque o executável da aplicação na mesma pasta. Exporte o caminho `CUDA_INSTALL_PATH` nessa pasta:

```
export CUDA_INSTALL_PATH=/usr/local/cuda-11.0/
```

Execute o `setup_environment` a partir do diretório `gpgpu-sim_distribution` na sua nova pasta de configuração:

```
source (setup_environment_location)
```

Exemplo:

```
source /home/lscad/Desktop/gpgpu-sim_distribution/setup_environment
```

Depois de configurar o ambiente, execute a aplicação:

```
./(application_name)
```

A simulação será iniciada no terminal. Para salvar a saída em um arquivo TXT, execute:

```
./(application_name) > output.txt
```

1.4.6 Aplicações usadas e resultados pré-simulados

As aplicações usadas podem ser encontradas em:

```
/multiexplorer/Benchmarks.zip
```

Os resultados pré-simulados podem ser encontrados em:

```
/multiexplorer/Multiexplorer/src/MultiexplorerGPGPU/DS_DSE/db/Experimentos
```

2 Utilizando o MultiExplorer

Nesta seção abordaremos a utilização de maneira prática da interface gráfica **MultiExplorer**.

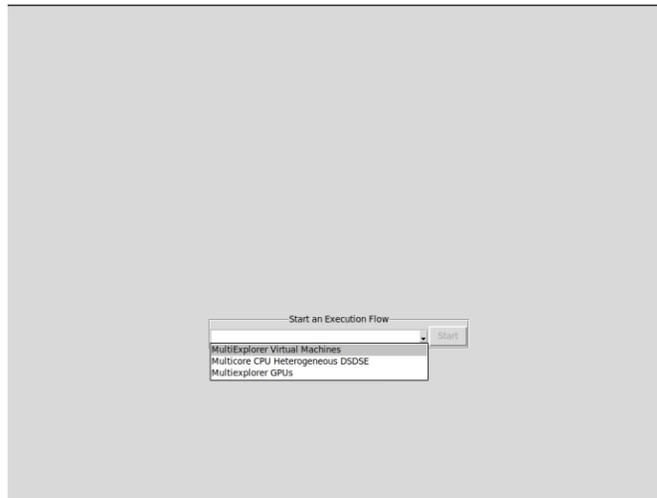


Figura 5: Pagina inicial da interface gráfica do MultiExplorer

Na figura 5, você verá que o primeiro passo é selecionar um fluxo de execução. Haverá três opções: "MultiExplorer Virtual Machines", "Multicore CPU Heterogeneous DSDSE" e "MultiExplorer GPUs".

2.1 Multicore CPU Heterogeneous DSDSE

Para o fluxo de CPU na figura 6, inicialmente, é apresentada uma breve descrição desse processo, abordando seus principais aspectos e funcionalidades. Ele envolve três etapas principais: Simulação, Exploração Física e Exploração do Espaço de Design (DSE), cada uma destinada a avaliar desempenho, consumo de energia e otimização do design para plataformas multicore heterogêneas, com foco em eficiência e redução de "dark silicon".

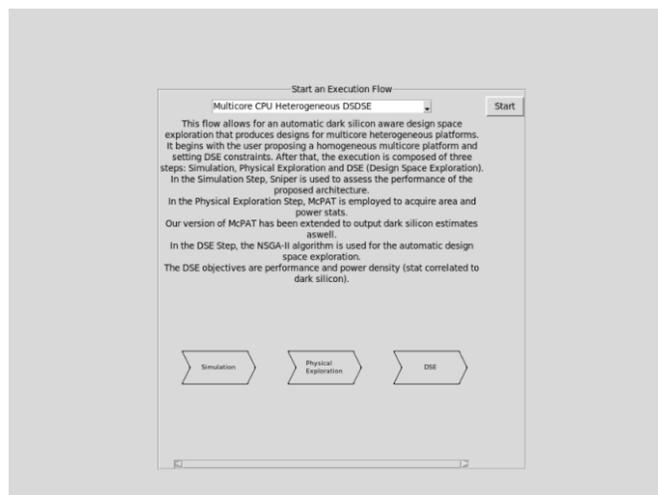


Figura 6: Fluxo de execução para CPUs.

Na Próxima etapa na figura 7, é necessário selecionar a litografia, a quantidade de núcleos e o modelo de núcleo desejado.

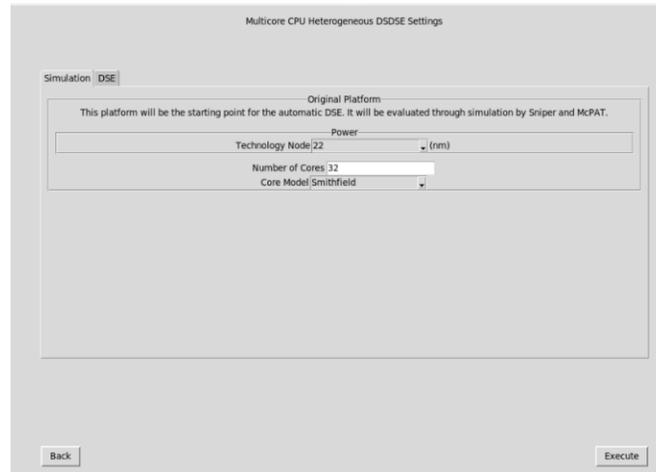


Figura 7: Tela de configuração da plataforma de simulação.

Na etapa seguinte na figura 8, é possível configurar as opções de DSE, que incluem: a quantidade inicial de núcleos desejada para o projeto, o número de núcleos IP a serem utilizados e as restrições do projeto, como densidade máxima de potência e área máxima permitida. Após ajustar essas configurações, clique em "Execute" para iniciar o fluxo do MultiExplorer.

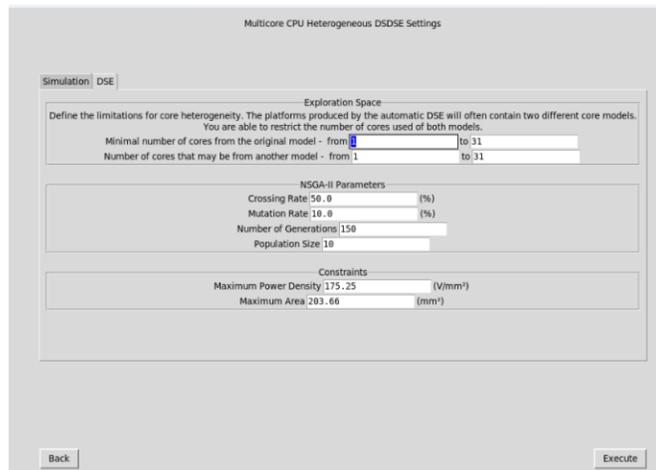


Figura 8: Tela de configuração do DSE

Na etapa seguinte na figura 9, é exibido todo o fluxo do MultiExplorer, incluindo as etapas de simulação, exploração do espaço de projeto e DSE. O módulo atualmente em execução pisca como um recurso visual, indicando qual processo está sendo executado no momento. Após a conclusão do fluxo de execução, o botão "See Results" é habilitado para visualização dos resultados.

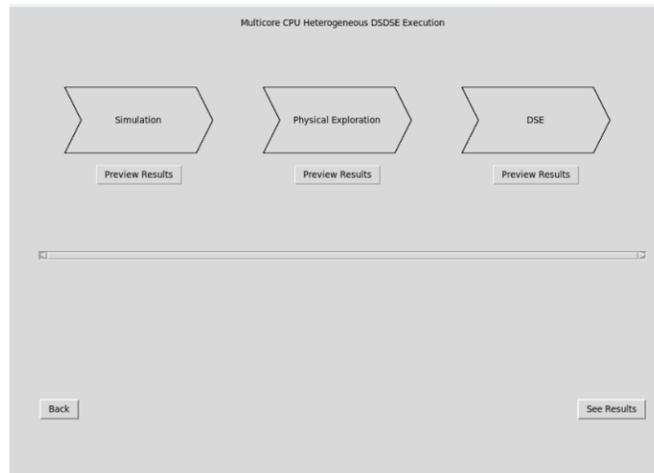


Figura 9: Tela de processamento.

Na etapa final do fluxo de execução na figura 10, é apresentado um gráfico que exhibe as combinações mais eficientes de performance e densidade de potência identificadas pela busca automática no projeto selecionado. Embora a interface gráfica facilite a configuração e visualização dos resultados de forma intuitiva, também é possível acessar os arquivos intermediários e finais diretamente no diretório `/rundir/Multicore_CPU_Heterogeneous_DSDSE`.

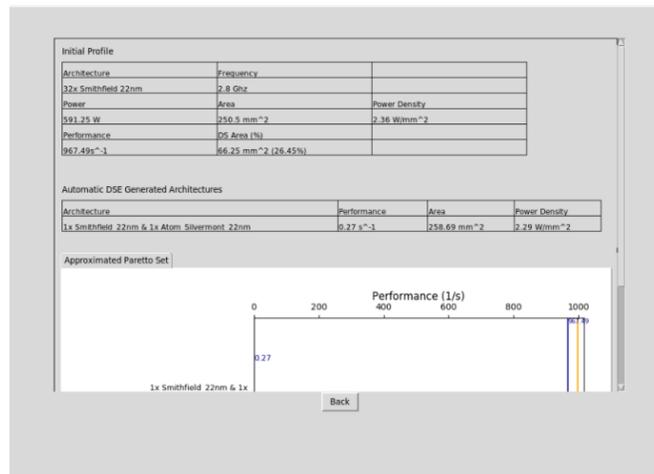


Figura 10: Janela com o resultado do fluxo do MultiExplorer.

2.2 MultiExplorer Virtual Machines

Para o fluxo de máquinas virtuais, primeiro é apresentada uma descrição de cada etapa que será seguida na figura 11. Esse fluxo permite configurar máquinas virtuais com base nos requisitos do usuário e nas restrições da aplicação, utilizando o CloudSim como plataforma de simulação em nuvem. A exploração do espaço de design (DSE) é realizada por meio de um algoritmo baseado em NSGA-II, e, nesta versão, também incorpora uma abordagem de força bruta para explorar todas as alternativas viáveis de design, servindo como validação para a metodologia DSE.

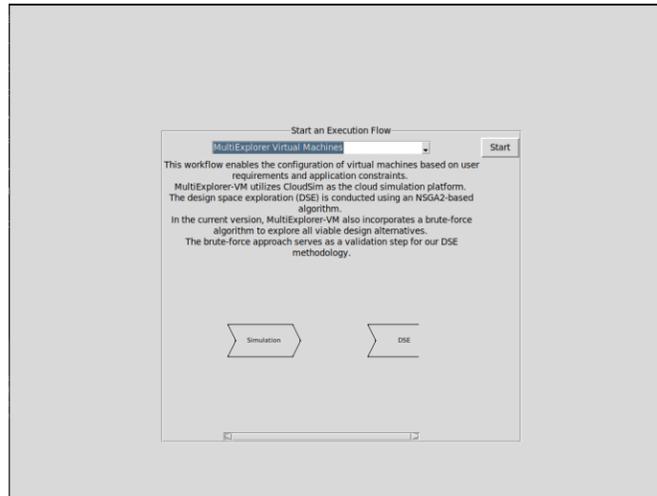


Figura 11: Tela inicial do fluxo de máquinas virtuais

nessa etapa na figura 12, o usuário precisa escolher o aplicativo de benchmark e o modelo de configuração inicial, que será representado por uma instância da AWS (Amazon Web Services). Além disso, é necessário informar a quantidade de núcleos requerida pelo cloudlet, uma classe utilizada para modelar aplicativos e serviços baseados em nuvem comumente empregados em data centers. Para aplicar as configurações recomendadas, o algoritmo DSE exigirá as restrições apropriadas.



Figura 12: Entrada de usuário para etapa de simulação.

Nesta etapa na figura 13, o usuário define as configurações do espaço de exploração (DSE) e ajusta os parâmetros do algoritmo NSGA-II. É possível habilitar a abordagem de força bruta para complementar a análise, determinar o número de máquinas virtuais suplementares e originais dentro de intervalos específicos, e ajustar parâmetros como taxa de cruzamento, taxa de mutação, número de gerações e tamanho da população. Essas configurações permitem personalizar a busca

por alternativas de design para máquinas virtuais, otimizando o processo conforme os requisitos do projeto.



Figura 13: Entrada de usuário para etapa de DSE.

Na etapa seguinte na figura 14, é exibido todo o fluxo do MultiExplorer, incluindo as etapas de simulação e DSE. O módulo atualmente em execução pisca como um recurso visual, indicando qual processo está sendo executado no momento. Após a conclusão do fluxo de execução, o botão "See Results" é habilitado para visualização dos resultados.



Figura 14: Tela de processamento.

Na etapa final do fluxo de execução na figura 15, exibe as configurações recomendadas na forma de tabelas e gráficos. Se o algoritmo de força bruta foi selecionado, suas configurações também serão exibidas.

Architecture	Predicted cost (USDh)	Predicted time (hours)
1x m5n.large & 1x r5.large	2.45e-01	1.04e-02
1x m5n.large & 1x m5n.2xlarge	5.95e-01	6.88e-03
1x m5n.large & 1x c5.xlarge	4.59e-01	5.92e-03
1x m5n.large & 2x m5n.large	3.57e-01	4.92e-03
1x m5n.large & 2x c5.large	2.89e-01	4.24e-03
1x m5n.large & 1x c5.4xlarge	7.99e-01	3.68e-03
3x m5n.large & 1x c5.xlarge	5.73e-01	3.55e-03
1x m5n.large & 2x c5n.xlarge	5.51e-01	3.55e-03
1x m5n.large & 2x m5n.xlarge	5.95e-01	3.44e-03
1x m5n.large & 3x c5.large	3.74e-01	2.60e-03

NSGA-II Approximated Pareto Set

Predicted time (hours)

Back

Figura 15: Resultados dispostos em tabelas e gráficos.

2.3 MultiExplorer GPUs

Para o fluxo de GPUs na figura 16. O processo começa com a seleção da configuração inicial e do aplicativo a ser utilizado. Em seguida, são especificadas restrições de densidade de potência e área. O desempenho da configuração inicial é avaliado por meio do simulador GPGPU-Sim, enquanto parâmetros físicos, como consumo energético, são estimados com a ferramenta GPGPUWatch, integrada ao simulador. Por fim, a exploração do espaço de design é realizada utilizando um algoritmo genético, como o NSGA-II, ou uma abordagem de força bruta, dependendo da estratégia escolhida para a análise.

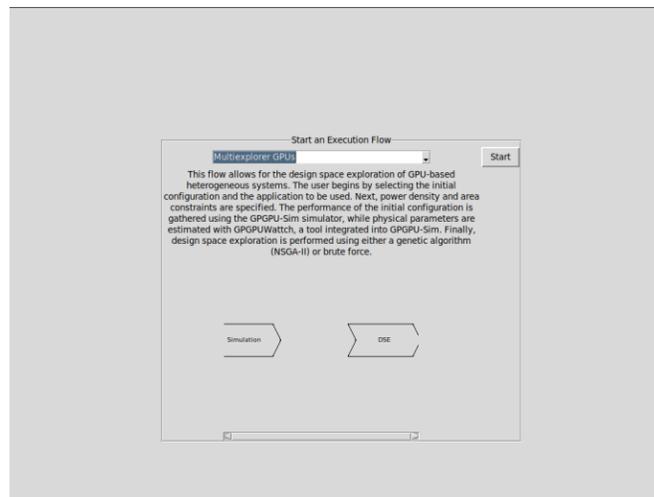


Figura 16: Tela inicial para o fluxo de GPUs.

Para a etapa de simulação na figura 17, o usuário deve escolher o aplicativo que será usado na simulação da configuração inicial (Modelo). Informações e parâmetros adicionais relacionados aos modelos de GPU estão disponíveis no diretório `"/multiexplorer/inputexamples/GPUs"`.

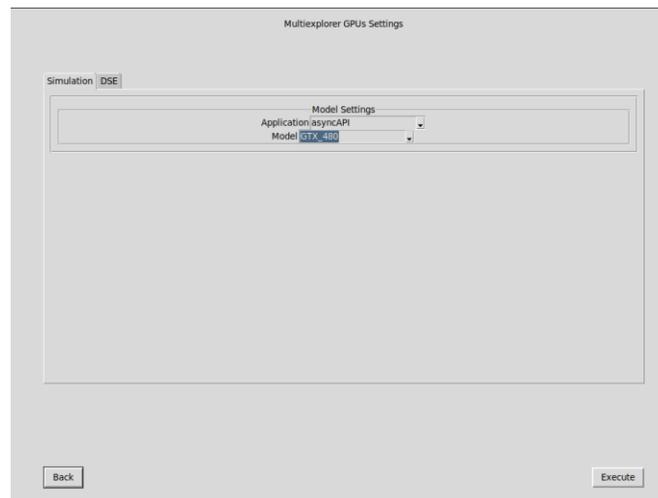


Figura 17: Entrada de usuário para etapa de simulação

Na próxima etapa as configurações do DSE na figura 18, o intervalo para a quantidade de Streaming Multiprocessors (SMs) deve ser especificado tanto para a configuração inicial quanto para o modelo de combinação heterogênea. Além disso, é possível ajustar os parâmetros padrão do algoritmo NSGA-II, como a taxa de cruzamento, a taxa de mutação, o número de gerações e o tamanho da população. O usuário também pode optar por executar um algoritmo de força bruta, que explora todas as combinações possíveis dentro dos intervalos definidos, complementando o processo realizado pelo NSGA-II.

Adicionalmente, o usuário deve definir restrições importantes para o design, como a densidade máxima de potência (em V/mm^3) e a área máxima do chip (em mm^2). Após configurar todos os parâmetros e restrições, o botão "Execute" pode ser acionado para iniciar a exploração do espaço de design. Essas etapas permitem personalizar o processo de otimização para atender às especificações do projeto.



Figura 18: Entrada de usuário para etapa de DSE

Na etapa seguinte na figura 19, é exibido todo o fluxo do MultiExplorer na GPUs, incluindo as etapas de simulação e DSE. O módulo atualmente em execução pisca como um recurso visual, indicando qual processo está sendo executado no momento. Após a conclusão do fluxo de execução, o botão "See Results" é habilitado para visualização dos resultados.



Figura 19: Tela de processamento

A etapa final apresentará as configurações recomendadas, resultantes da combinação entre a configuração inicial e os modelos selecionados do banco de dados. Essas configurações serão exibidas de forma clara e organizada, utilizando tabelas para detalhar os parâmetros e gráficos para ilustrar as relações e os resultados obtidos durante o processo de exploração do espaço de projeto, que resulta em combinações de GPUs heterogêneas.

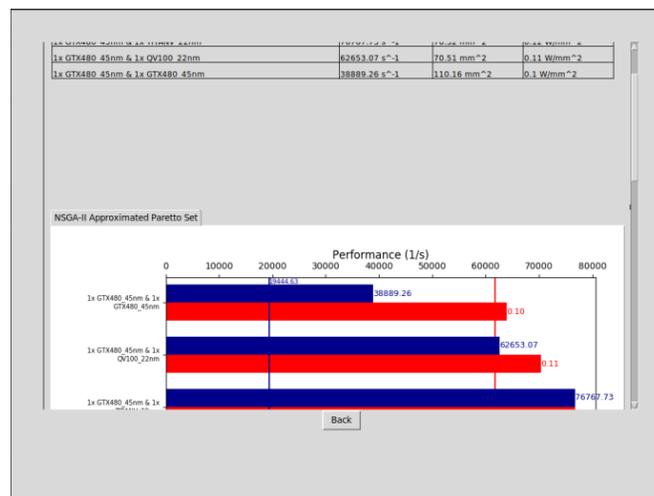


Figura 20: Configurações recomendadas de combinações heterogêneas de GPUs em tabelas e gráficos.

3 Como expandir o MultiExplorer para novos fluxos de execução

Esta seção fornece uma visão geral de alto nível de como integrar novos fluxos de execução na ferramenta MultiExplorer. A complexidade e as etapas específicas podem variar dependendo no fluxo e nos recursos que o usuário pretende incorporar.

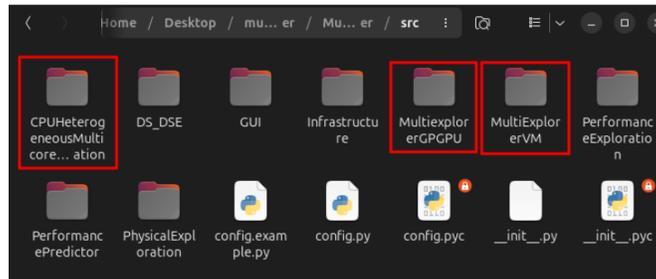


Figura 21: Pasta de código-fonte do MultiExplorer.

Para integrar um fluxo de execução, uma pasta representando esse fluxo deve ser criada no diretório de origem do *MultiExplorer*: `/multiexplorer/Multiexplorer/src/`. A figura 21 mostra os fluxos de execução atualmente disponíveis.

Dentro dessa nova pasta de fluxo de execução, podem ser criados cinco arquivos Python, junto com quaisquer arquivos de dependência necessários. Esses arquivos incluem: `AllowedValues.py`, `Steps.py`, `Adapters.py`, `Presenters.py`, e `MultiExplorer{flow_name}.py`.

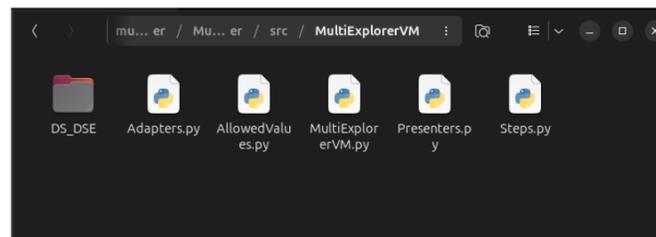


Figura 22: Pasta de fluxo de execução

A figura 22 ilustra um exemplo de estrutura de pasta de fluxo de execução.

3.1 Steps

Cada fluxo de execução é dividido em etapas, com cada etapa realizando um tipo diferente de processamento. Para o fluxo de CPU, existem três etapas: simulação, exploração física e DSE. Para os fluxos de GPU e VM, existem duas etapas: simulação e DSE. A diferença é que a exploração física não é necessária para o fluxo de VM e está integrada à ferramenta de simulação para GPUs.

Cada etapa aparecerá como uma aba na interface do *MultiExplorer* – se a etapa requerer entradas do usuário – como mostrado na figura 23.

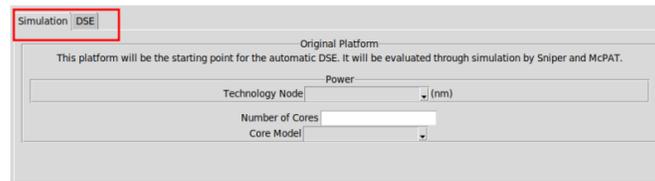


Figura 23: Guias de fluxo de execução.

O código Python (Código 1) fornece um exemplo de um arquivo **Steps.py** com uma classe de etapa.

O arquivo **Steps.py** contém funções para lidar com os resultados de seus adaptadores (`get_results`), as entradas do usuário (`get_user_inputs`), o apresentador da etapa (`get_presenter`) e o nome da etapa a ser exibido na interface (`get_label`).

```

1 import os
2 import sys
3 from ..Infrastructure.Events import Event
4 from ..Infrastructure.ExecutionFlow import Step
5 from Presenters import CloudSimPresenter, NSGAPresenter
6 from Adapters import CloudsimAdapter, NsgaIIPredDSEAdapter
7
8 ...
9
10 class NSGAIIDSEStep(Step):
11
12     def get_results(self):
13         return self.adapter.get_results()
14
15     def get_presenter(self):
16         return NSGAPresenter()
17
18     def __new__(cls):
19         if not hasattr(cls, 'instance'):
20             cls.instance = super(
21                 NSGAIIDSEStep,
22                 cls
23             ).__new__(cls)
24
25         return cls.instance
26
27     def __init__(self):
28         super(NSGAIIDSEStep, self).__init__()
29
30         self.adapter = NsgaIIPredDSEAdapter()
31
32     @staticmethod
33     def get_label():

```

```
34         return 'DSE'
35
36     @staticmethod
37     def has_user_input():
38         return True
39
40     def get_user_inputs(self):
41         return self.adapter.get_user_inputs()
42
43     def __execute__(self):
44         self.execution_exception = None
45
46         try:
47             self.adapter.execute()
48         except BaseException as exception:
49             self.execution_exception = exception
50
51     def __finish__(self):
52         if self.execution_exception is None:
53             self.fire(Event.STEP_EXECUTION_ENDED)
54         else:
55             self.fire(Event.STEP_EXECUTION_FAILED, self)
```

Listing 1: Steps Python code

3.2 Adapters

O adaptador é o arquivo que gerencia as operações de uma etapa, como obter entradas do usuário, realizar operações em arquivos e executar aplicações. As funções neste arquivo dependem da etapa específica.

Neste exemplo de **Adapters.py**, o adaptador lida com as entradas do usuário para a etapa específica. Cada entrada do usuário possui configurações, como tipo, valor padrão e valores mínimos e máximos. O adaptador inclui funções que preparam arquivos JSON de acordo com essas entradas do usuário, como `prepare`, `change_json_parameters`, entre outras. Com essas modificações e preparações, os respectivos algoritmos de DSE serão executados na função `execute`, como `dse` e/ou `brute_force`. O adaptador também deve incluir a função `get_results` para que os dados possam ser acessados por outros arquivos (e.g., *Presenters*).

O Código 2 ilustra o adaptador correspondente à etapa exibida no Código 1.

```
1
2 class NsgaIIPredDSEAdapter(Adapter):
3
4     def __init__(self):
5         Adapter.__init__(self)
6
```



```
7     self.set_inputs([
8         InputGroup({
9             'label': "Exploration Space",
10            'key': 'exploration_space',
11            'inputs': [
12                Input({
13                    'label': 'Run brute force aswell',
14                    'key': 'run_brute_force',
15                    "is_user_input": True,
16                    "required": False,
17                    'type': InputType.Checkbutton,
18                    'value': True,
19                }),
20                Input({
21                    'label': 'Original number VM for design',
22                    'key': 'original_vm_for_design',
23                    "is_user_input": True,
24                    "required": True,
25                    'type': InputType.IntegerRange,
26                    'min_val': 1,
27                    'max_val': 31,
28                })
29            ]
30        }),
31        ...
32
33    def execute(self):
34        self.prepare()
35
36        if self.inputs['exploration_space']['run_brute_force']:
37            self.dse_brute_force()
38
39        self.dse()
40        self.register_nsga_results()
41        self.register_brute_force_results()
42        self.ord_values()
43
44    def get_json_name(self):
45        ...
46
47    def get_input_json(self):
48        return self.project_folder + '/' + self.get_json_name()
49
50    def change_json_parameters(self):
51        ...
52
53    def get_json_data(self):
```



```
54     with open(self.input_json) as json_file:
55         self.json_data = json.load(json_file)
56
57     def prepare(self):
58         self.project_folder = self.get_output_path()
59         self.input_json = self.get_input_json()
60         self.change_json_parameters()
61         self.get_json_data()
62
63     def dse(self):
64         self.nsga = Nsga2Main (
65             self.project_folder,
66             self.input_json,
67             float(self.inputs['nsga_parameters']['
68                 mutation_strength']/100,
69             float(self.inputs['nsga_parameters']['
70                 mutation_rate']/100,
71             int(self.inputs['nsga_parameters']['
72                 num_of_individuals']),
73             int(self.inputs['nsga_parameters']['
74                 num_of_generations']),
75             self.inputs['exploration_space']['run_prediction'
76                 ]
77         )
78         print("DSE NSGA2: OK")
79
80     def dse_brute_force(self):
81         self.brute_force = DsDseBruteForce(self.project_folder, self.
82             input_json, self.inputs['exploration_space']['run_prediction'
83             ])
84         print("DSE Brute Force: OK")
85
86     def register_nsga_results(self):
87         ...
88
89     def register_brute_force_results(self):
90         ...
91
92     def get_results(self):
93         return self.presentable_results
94
95     def ord_values(self):
96         ..
97
98     def get_filtered_results(self, solutions):
99         ...
```

Listing 2: Adapters Python code

3.3 AllowedValues

O arquivo `AllowedValues.py` contém listas de dados. Se uma entrada do usuário precisar ser selecionada a partir de um conjunto predefinido de modelos, uma classe representando essa lista deve ser criada no arquivo `AllowedValues.py` e usada no campo `'type'` dentro dos adaptadores. Por exemplo, no fluxo de execução para GPU (Código 3), há uma lista de nove aplicações para o *benchmark*. Essa abordagem permite uma expansão fácil do banco de dados de aplicações. Além disso, oferece outros benefícios, como simplificar a recuperação de um caminho de arquivo JSON para cada modelo.

```
1
2 from enum import Enum
3 from ..config import PATH_INPUTS
4
5 ...
6
7 class Applications(Enum):
8     asyncAPI = 1
9     backprop = 2
10    bfs = 3
11    clock = 4
12    dwt2d = 5
13    hotspot = 6
14    needle = 7
15    nn = 8
16    vectorAdd = 9
17
18    @staticmethod
19    def belongs(value):
20        return value in set(item.value for item in Applications)
21
22    @staticmethod
23    def get_label(value):
24
25        if value == Applications.asyncAPI:
26            return "asyncAPI"
27        if value == Applications.backprop:
28            return "backprop"
29        if value == Applications.bfs:
30            return "bfs"
31        if value == Applications.clock:
32            return "clock"
33        if value == Applications.dwt2d:
```

```
34     return "dwt2d"
35 if value == Applications.hotspot:
36     return "hotspot"
37 if value == Applications.needle:
38     return "needle"
39 if value == Applications.nn:
40     return "nn"
41 if value == Applications.vectorAdd:
42     return "vectorAdd"
43
44     raise ValueError("Value does not corresponds to a known predicted
45                        core.")
46
47 @staticmethod
48 def get_dict():
49
50     return {
51         Applications.asyncAPI.value: Applications.get_label(
52             Applications.asyncAPI),
53         Applications.backprop.value: Applications.get_label(
54             Applications.backprop),
55         Applications.bfs.value: Applications.get_label(Applications.
56             bfs),
57         Applications.clock.value: Applications.get_label(Applications
58             .clock),
59         Applications.dwt2d.value: Applications.get_label(Applications
60             .dwt2d),
61         Applications.hotspot.value: Applications.get_label(
62             Applications.hotspot),
63         Applications.needle.value: Applications.get_label(
64             Applications.needle),
65         Applications.nn.value: Applications.get_label(Applications.nn
66             ),
67         Applications.vectorAdd.value: Applications.get_label(
68             Applications.vectorAdd),
69     }
```

Listing 3: AllowedValues Python code

3.4 Presenters

O arquivo **Presenters.py** é responsável pela apresentação dos resultados em forma de tabelas, gráficos, pré-visualizações e mais. Uma classe para cada tabela e cada gráfico – NSGA-II e Força Bruta – deve ser criada. Além disso, é necessária uma classe para as informações de pré-visualização na janela de processamento.

Os dados para a apresentação vêm da função `get_results` do arquivo **Adapters.py**, por



meio do atributo `adapter` no arquivo `Steps.py`. As bibliotecas utilizadas foram `TKinter` e `matplotlib`, mas podem variar de acordo com as necessidades do fluxo de execução.

O Código 4 ilustra uma classe de pré-visualização e uma classe de apresentação de tabela.

```
1
2 import copy
3 import Tkinter
4 import numpy as np
5 from typing import Dict, Tuple
6 from matplotlib.figure import Figure
7 from ..GUI.Widgets import CanvasTable
8 from ..GUI.Presenters import Presenter, PlotbookPresenter
9
10
11 class CloudSimPresenter(Presenter):
12
13     def __init__(self):
14         super(CloudSimPresenter, self).__init__()
15
16         self.table = None
17
18         self.canvas_frame = None
19
20     def present_results(self, frame, results, options=None):
21         return 0
22
23     def present_partials(self, frame, step_results, options=None):
24         raise NotImplementedError
25
26     def get_info(self, step_results, options=None):
27
28         simulation_preview = (
29             "Time: {} hours\n"
30             "Price: {} USD/h\n"
31         ).format('%0.2e' % step_results['original_time'], '%0.2e' % round(
32             step_results['original_price'], 2)
33         )
34
35         return simulation_preview
36
37 class NSGATablePresenter(Presenter):
38
39     def __init__(self):
40         super(NSGATablePresenter, self).__init__()
41
42         self.canvas = None
```



```
43     self.og_table = None
44
45     self.sol_table = None
46
47     def present_partials(self, frame, step_results, options=None):
48         raise NotImplementedError
49
50     def present_results(self, frame, results, options=None):
51
52         if 'nsga_solutions' not in results['dsdse']:
53             return 0
54
55         cell_height = 25
56
57         table_options = {
58             'pos': (2, 3*(cell_height+2)),
59             'cells_width': [250, 250, 250],
60             'font_height': 12,
61             'cell_height': cell_height,
62             'nbr_of_columns': 3,
63             'nbr_of_rows': 6,
64             'center': False,
65         }
66
67         self.canvas = Tkinter.Canvas(frame)
68
69         solutions = results['dsdse']['nsga_solutions']
70
71         sorted_solution = results['dsdse']['sorted_nsga']
72
73         nbr_of_solutions = len(sorted_solution)
74
75         height = table_options['cell_height'] * (6 + nbr_of_solutions +
76             6)
77
78         self.canvas.config(width=options['width'], height=height)
79
80         self.canvas.pack(
81             fill=Tkinter.BOTH,
82             expand=True
83         )
84
85         self.canvas.create_text(2, 2*(cell_height+2), text="NSGA-II
86             Generated Architectures",
87             anchor=Tkinter.NW)
```



```
88
89     table_options['cells_width'] = [445, 145, 140, 140]
90
91     table_options['nbr_of_rows'] = nbr_of_solutions + 1
92
93     table_options['nbr_of_columns'] = 3
94
95     solutions_data = [
96         ['Architecture', 'Predicted cost (USD/h)', 'Predicted time (
97             hours)'],
98     ]
99
100     for s in sorted_solution:
101         solutions_data.append([
102             s,
103             '%.2e' % round(solutions[s]['cost_pred'], 10),
104             "%.2e" % round(float(solutions[s]['time_pred']), 10),
105         ])
106
107     table_options['data'] = solutions_data
108
109     self.sol_table = CanvasTable(self.canvas, table_options)
110
111     return height
112
113     def get_info(self, step_results, options=None):
114         raise NotImplemented
```

Listing 4: Presenters Python code

3.5 MultiExplorer{flow_name}

O arquivo `MultiExplorer{flow_name}.py` é o arquivo principal de um fluxo de execução. Ele contém a classe do fluxo de execução, que será incluída na lista de fluxos de execução do MultiExplorer e exibida como uma opção. Esta classe inclui a descrição do fluxo de execução (`get_info`), as etapas do fluxo (`self.steps`), o nome a ser exibido na interface do MultiExplorer (`get_label`), gerencia a pasta que contém cada projeto e lida com a execução do fluxo.

O Código 5 ilustra o fluxo de execução de VM.

```
1
2 import os
3 import tkinter as tk
4 from ..config import PATH_RUNDIR
5 from ..Infrastructure.Events import Event
6 from Steps import CloudSimStep, NSGAIIDSEStep
7 from ..Infrastructure.ExecutionFlow import ExecutionFlow
```



```
8 from Presenters import BruteForceTablePresenter, NSGATablePresenter,
   CloudSimPresenter, NSGAPresenter, BruteForcePresenter
9
10 class MultiExplorerVMExecutionFlow(ExecutionFlow):
11
12     @staticmethod
13     def get_info():
14         return (
15             "This workflow enables the configuration of virtual machines
16             based on user requirements and application constraints. \n"
17             + "MultiExplorer-VM utilizes CloudSim as the cloud simulation
18             platform. \n"
19             + "The design space exploration (DSE) is conducted using an
20             NSGA2-based algorithm. \n"
21             + "In the current version, MultiExplorer-VM also incorporates
22             a brute-force algorithm to explore all viable design
23             alternatives. \n"
24             + "The brute-force approach serves as a validation step for
25             our DSE methodology."
26         )
27
28     def __new__(cls):
29         if not hasattr(cls, 'instance'):
30             cls.instance = super(
31                 MultiExplorerVMExecutionFlow,
32                 cls
33             ).__new__(cls)
34
35         return cls.instance
36
37     def __init__(self):
38         super(MultiExplorerVMExecutionFlow, self).__init__()
39
40         self.steps = [
41             CloudSimStep(),
42             NSGAIIDSEStep()
43         ]
44
45     @staticmethod
46     def get_label():
47         return 'MultiExplorer Virtual Machines'
48
49     def get_output_path(self):
50         return (
51             PATH_RUNDIR
52             + "/" + MultiExplorerVMExecutionFlow.get_label().replace(
```



```

    , , , '_')
47 )
48
49 def setup_dirs(self):
50     output_path = self.get_output_path()
51
52     if not os.path.exists(output_path):
53         os.makedirs(output_path)
54
55     nbr_of_dirs = len(next(os.walk(output_path))[1])
56
57     nbr_of_dirs = nbr_of_dirs & 63
58
59     output_path = output_path + "/" + "{:02d}".format(nbr_of_dirs)
60
61     if not os.path.exists(output_path):
62         os.makedirs(output_path)
63
64     for step in self.steps:
65         step.set_output_path(output_path)
66
67 def execute(self):
68     self.setup_dirs()
69
70     ExecutionFlow.execute(self)
71
72 def get_results(self):
73     return {
74         "cloudsim": self.steps[0].get_results(),
75         "dsdse": self.steps[1].get_results()
76     }
77
78 def get_presenters(self):
79     return [
80         NSGATablePresenter(),
81         NSGAPresenter(),
82         BruteForceTablePresenter(),
83         BruteForcePresenter(),
84         CloudSimPresenter()
85     ]
86
87 def handle_step_failure(self, step):
88     tkMessageBox.showerror(
89         "Execution Failure",
90         "The " + step.get_label() + " Step execution wasn't
91         successful. " + str(step.execution_exception)
92     )
```



```
92         self.fire(Event.FLOW_EXECUTION_FAILED)
93
94
95     def finish(self):
96         self.fire(Event.FLOW_EXECUTION_ENDED)
```

Listing 5: Virtual machines execution flow Python code

3.6 Como incluir um fluxo de execução na lista do MultiExplorer

Para incluir um fluxo de execução no MultiExplorer, o fluxo deve ser registrado no arquivo */multiexplorer/MultiExplorer/src/Infrastructure/Registries.py*.

Para registrar um fluxo, a função `register_flow_class` deve ser chamada com a classe do fluxo de execução e seu nome correspondente.

```
1
2 class ExecutionFlowRegistry(object):
3
4     ...
5
6     def __init__(self):
7         self.flow_classes = {} # type: Dict[str, 'class']
8
9         self.register_flow_class(
10             CPUHeterogeneousMulticoreExplorationExecutionFlow,
11             CPUHeterogeneousMulticoreExplorationExecutionFlow.get_label()
12         )
13
14         self.register_flow_class(
15             MultiExplorerVMEExecutionFlow,
16             MultiExplorerVMEExecutionFlow.get_label()
17         )
18
19         self.register_flow_class(
20             MultiexplorerGPGPUExecutionFlow,
21             MultiexplorerGPGPUExecutionFlow.get_label()
22         )
23
24     ...
```

Listing 6: Register an execution flow Python code