

Entregável 2.3. Processamento do PyCCD em plataforma de computação avançada (INCD)

Índice

1. Introdução	3
2. Acesso ao cluster do INCD	3
3. Upload dos dados no cluster do INCD	3
4. Instalação dos recursos necessários do PyCCD no cluster do INCD	7
5. Executar scripts no cluster do INCD	8
6. Resultados e conclusões	11
7. Questões em aberto	14
8. Anexos	14

Índice das Figuras

Figura 1 – Ligação ao cluster do INCD.	3
Figura 2 – Configuração do acesso via SFTP pelo WinSCP para aceder ao cluster do INCD.	4
Figura 3 – Configuração do acesso via SFTP pelo WinSCP para aceder ao cluster do INCD.	4
Figura 4 – Configuração avançada do acesso via SFTP pelo WinSCP.	5
Figura 5 – Configuração da chave privada para estabelecer ligação ao cluster do INCD através do WinSCP.	6
Figura 6 – O painel à esquerda apresenta as pastas e diretorias armazenados localmente na máquina do utilizador, enquanto o painel à direita exhibe as pastas e diretorias localizados no cluster do INCD.	7
Figura 7 – Ligação ao cluster do INCD por SSH.	7
Figura 8 – Criação do ambiente virtual (ccdISA) para processamento do PyCCD no cluster do INCD.	8
Figura 9 – Lista dos ambientes virtuais disponíveis no cluster do INCD.	8
Figura 10 – Ativação do ambiente virtual ccdISA no cluster do INCD.	8
Figura 11 – Exemplo do ficheiro de submissão (submit_job.sh) para execução de um script no cluster do INCD, aberto com o WinSCP.	9
Figura 12 – Comando utilizado para submeter o ficheiro de submissão submit_job.sh ao SLURM para agendamento e execução do job no cluster e respetiva mensagem retornada pelo SLURM indicando a submissão bem-sucedida do job, com o identificador único 40039.	9

Figura 13 – Comando utilizado para monitorizar em tempo real o arquivo de log slurm-40039.out, exibindo as mensagens de execução, erros ou resultados intermediários.	10
Figura 14 – Relatório de utilização de recursos do cluster no dia 2024-10-19, incluindo uso de quota de disco (em KB) e comparação entre os recursos de CPU alocados e consumidos (em minutos).	10
Figura 15 – Progresso da execução do PyCCD numa máquina local com 6 núcleos e 3.2 GHz para 50.000 pixels.	11
Figura 16 – Parte do código responsável pela configuração do processamento paralelo com ProcessPoolExecutor.	12

Índice das Tabelas

Tabela 1 – Testes e configurações para a otimização do tempo de processamento para conjuntos de 50.000, 100.000 e 241.968 pixels, realizados com a técnica de paralelização dos pixels dentro dos batches. Os testes foram realizados utilizando apenas um nó dos cinco disponíveis no cluster, explorando duas configurações de tarefas: o máximo permitido de 96 ntasks-per-node e uma configuração intermediária de 24 tasks-per-node.	12
Tabela 2 – Resultados escalados para 120 milhões de pixels, considerando diferentes configurações de ntasks-per-node. O tempo de processamento total e os minutos acumulados de processamento por núcleo são extrapolados a partir de testes com 50.000 pixels.	13

1. Introdução

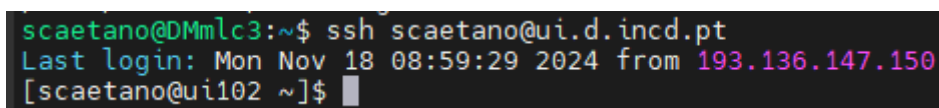
Este relatório descreve as etapas realizadas no âmbito da execução do projeto definido no entregável E.2.3., com foco na avaliação e implementação de alternativas para o processamento computacional. Devido aos elevados requisitos de recursos computacionais necessários para a execução do código PyCCD, foi realizada uma candidatura ao RNCA, na categoria A0 de acesso experimental.

Como resultado, foi concedido acesso ao servidor do INCD, permitindo a utilização dos clusters de computação avançada disponíveis. O objetivo é garantir que o processamento é realizado de forma eficiente e dentro dos requisitos do projeto. Neste contexto, explorou-se a viabilidade de executar o processamento do PyCCD no ambiente fornecido, com o objetivo de, por um lado, alcançar o menor tempo de processamento possível e, por outro, otimizar a utilização dos recursos computacionais disponíveis.

2. Acesso ao cluster do INCD

Após a aceitação do projeto e a criação das contas correspondentes no cluster, foram fornecidos os acessos necessários. O processo de conexão é realizado por meio do programa MobaXterm, disponível em: <https://mobaxterm.mobatek.net/>.

Para estabelecer a ligação ao cluster, basta abrir o MobaXterm e, no terminal, executar o seguinte comando, substituindo 'scaetano' pelo nome de utilizador atribuído a cada membro do projeto: ssh scaetano@ui.d.incd.pt



```
scaetano@DMmlc3:~$ ssh scaetano@ui.d.incd.pt
Last login: Mon Nov 18 08:59:29 2024 from 193.136.147.150
[scaetano@ui102 ~]$
```

Figura 1 – Ligação ao cluster do INCD.

Após inserir as credenciais corretamente, o acesso ao servidor será estabelecido.

3. Upload dos dados no cluster do INCD

Para carregar dados diretamente no cluster do INCD, utilizou-se o programa WinSCP, disponível para download em: <https://winscp.net/eng/download.php>. A configuração do acesso ao cluster utilizando o WinSCP, foi feita seguindo os passos abaixo:

- I. Abrir o WinSCP, de seguida clicar em New Tab. Uma nova janela de login será exibida:

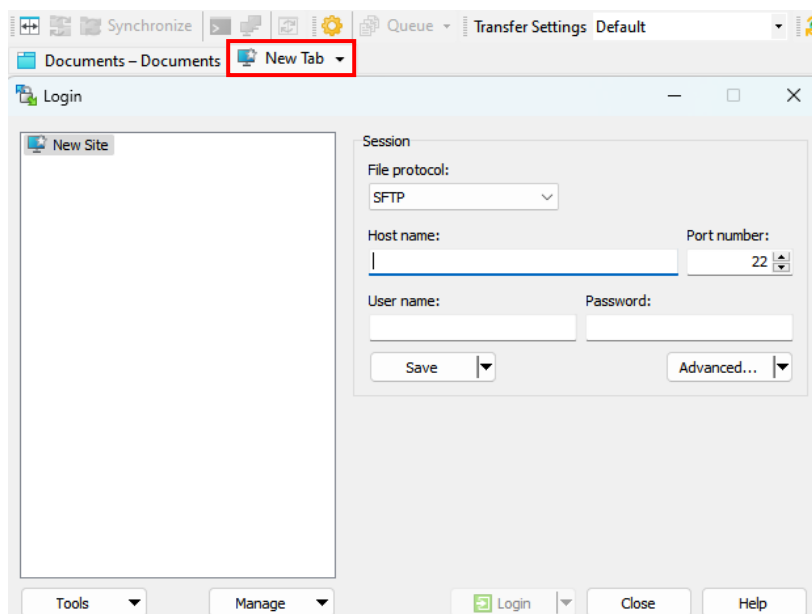


Figura 2 – Configuração do acesso via SFTP pelo WinSCP para aceder ao cluster do INCD.

II. Configuração da sessão: preencher os campos conforme abaixo.

- Hostname: inserir o endereço do cluster (ui.d.incd.pt).
- Username: inserir o nome de utilizador fornecido.

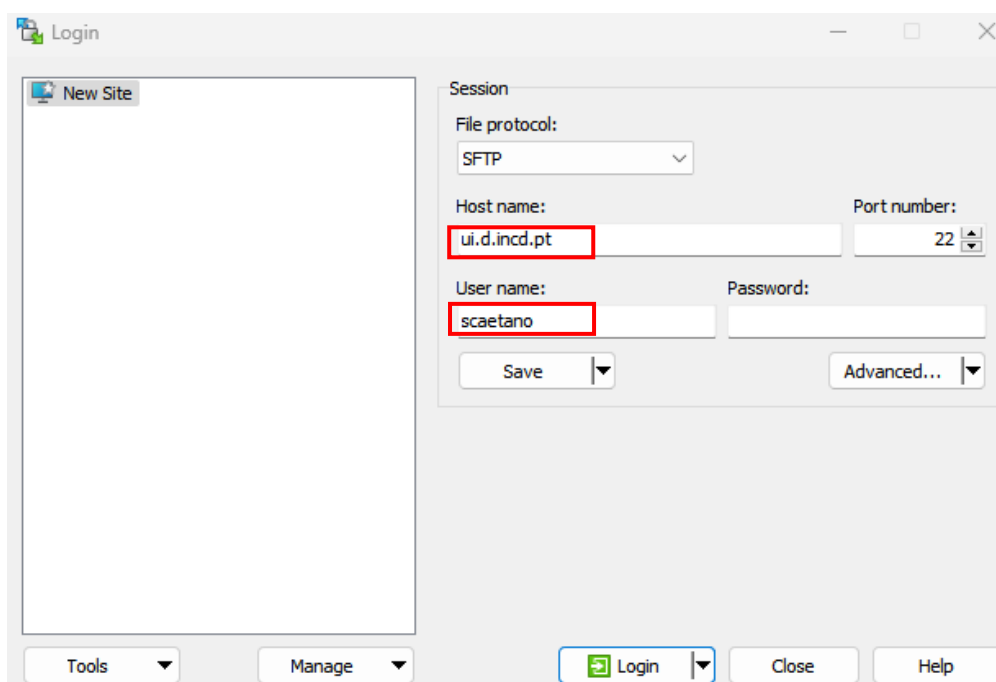


Figura 3 – Configuração do acesso via SFTP pelo WinSCP para aceder ao cluster do INCD.

- III. Configuração Avançada: clicar no botão “Advanced...”. Aceder à aba “SSH > Authentication”. No campo destinado à chave de autenticação, clicar no botão “...” e seleccionar o ficheiro da chave privada (no formato ppk) que foi gerado com a chave pública para a criação da conta no cluster:

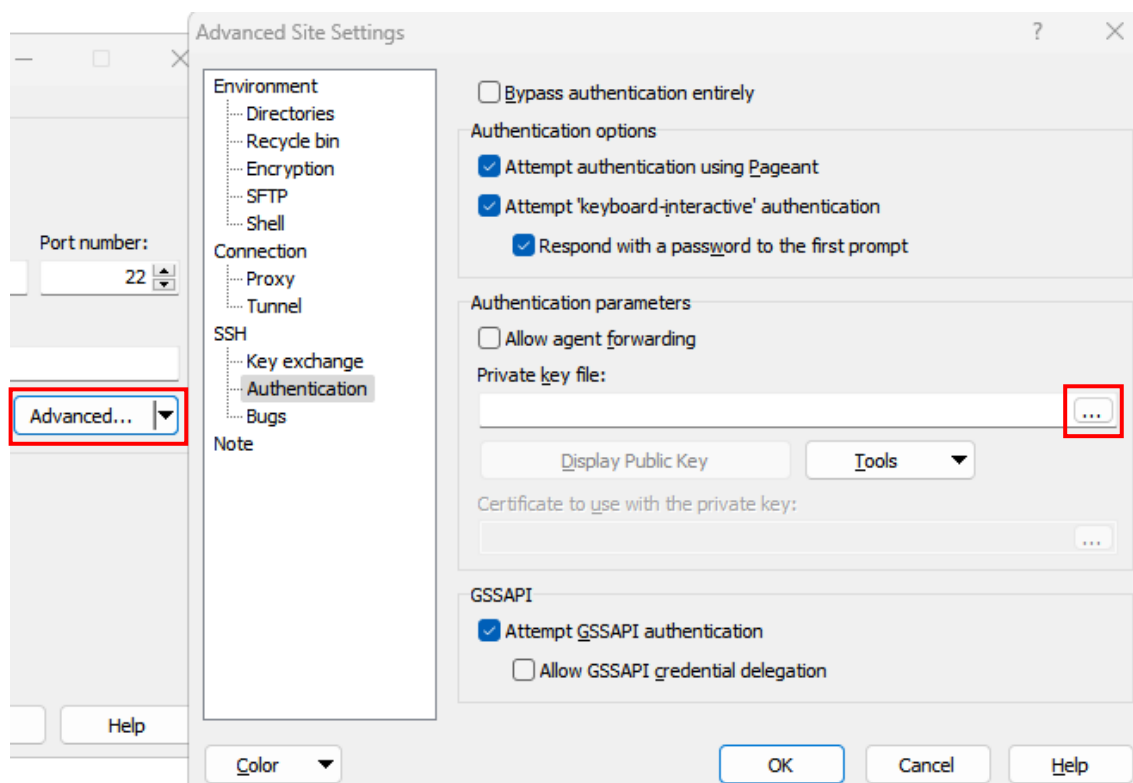


Figura 4 – Configuração avançada do acesso via SFTP pelo WinSCP.

- IV. Após seguir os passos, deverá ficar com o seguinte aspeto:

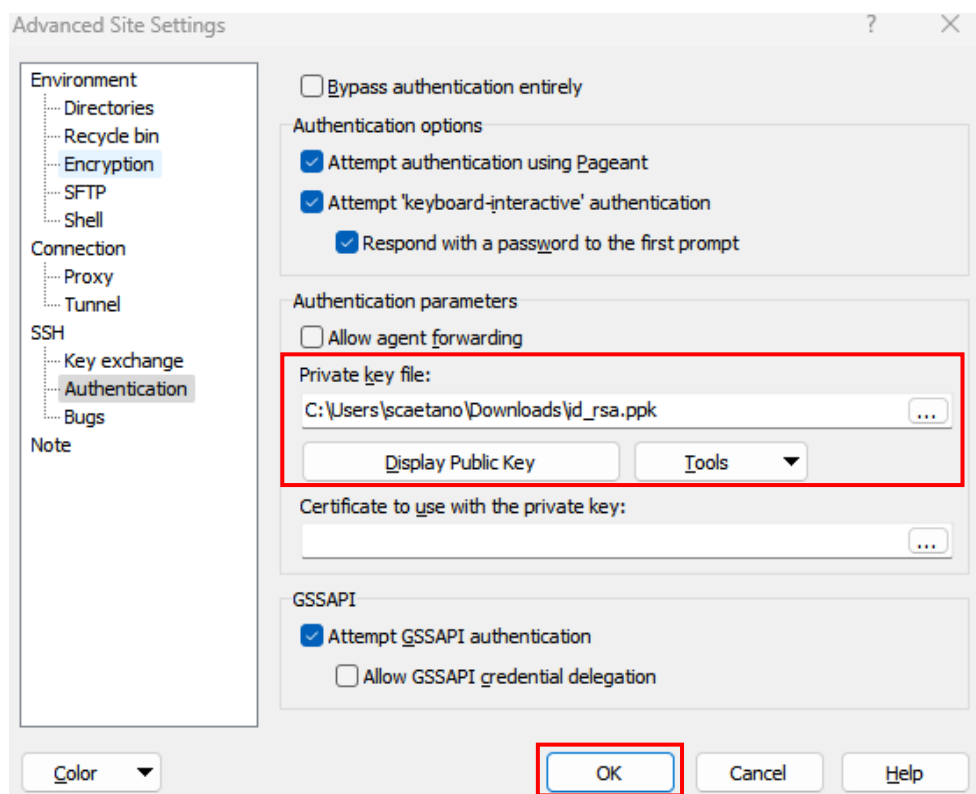


Figura 5 – Configuração da chave privada para estabelecer ligação ao cluster do INCD através do WinSCP.

Com o cluster devidamente configurado no WinSCP, como mostrado na Figura 6, o programa está apto a realizar a transferência de ficheiros entre a máquina local e o cluster do INCD.

- Painel à esquerda: exhibe os ficheiros e diretorias armazenados na máquina local.
- Painel à direita: apresenta os ficheiros e diretorias disponíveis no cluster do INCD.

A partir desta interface, é possível copiar, mover ou editar ficheiros de forma prática e intuitiva entre os dois ambientes.

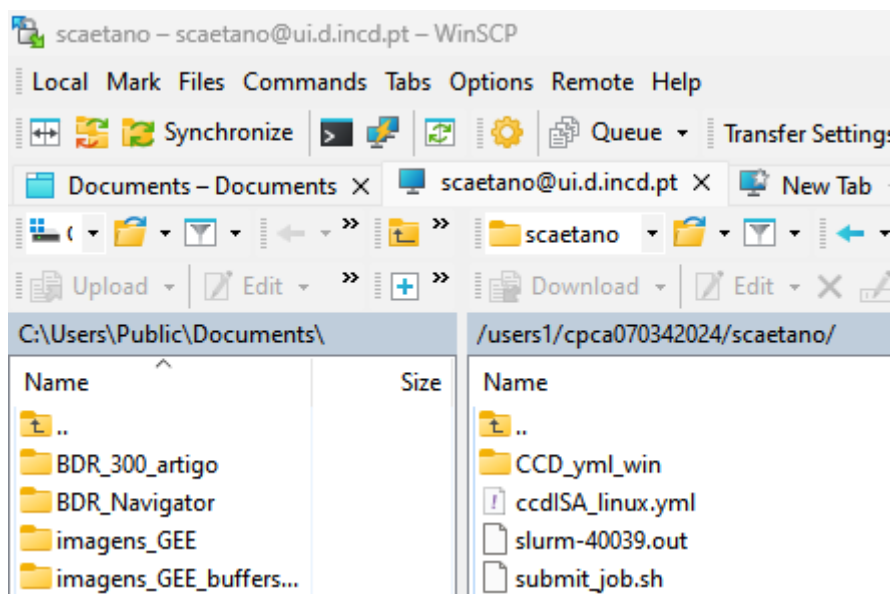


Figura 6 – O painel à esquerda apresenta as pastas e diretorias armazenados localmente na máquina do utilizador, enquanto o painel à direita exibe as pastas e diretorias localizados no cluster do INCD.

4. Instalação dos recursos necessários do PyCCD no cluster do INCD

A configuração do PyCCD no cluster requer a transferência de ficheiros essenciais e a criação de um ambiente virtual apropriado. Primeiramente, é necessário transferir para o cluster a pasta que contém os scripts do PyCCD, os dados de entrada (GeoTiffs das imagens Sentinel-2) e o GeoPackage da região de interesse. Estes ficheiros devem ser organizados dentro do cluster, conforme ilustrado na Figura 6 (painel direito) pela pasta CCD.yml_win.

Além disso, o ficheiro **ccdISA_linux.yml**, que pode ser obtido em: <https://github.com/manuelcampagnolo/S2CHANGE>, deve ser colocado no cluster. Este ficheiro contém as especificações para a criação do ambiente virtual necessário para a execução do PyCCD, conforme também mostrado na Figura 6.

Os passos seguintes só precisam de ser realizados uma única vez para instalar o ambiente virtual no cluster do INCD.

- I. Com os arquivos devidamente transferidos, abrir o mobaXterm e estabelecer ligação com o cluster. No terminal, deve inserir-se o seguinte comando: `ssh scaetano@ui.d.incd.pt`

```
scaetano@DMmlc3:~$ ssh scaetano@ui.d.incd.pt
Last login: Mon Nov 18 08:59:29 2024 from 193.136.147.150
[scaetano@ui102 ~]$
```

Figura 7 – Ligação ao cluster do INCD por SSH.

- II. Após efetuar a ligação ao cluster, é necessário criar o ambiente virtual chamado **ccdISA**, onde os scripts do PyCCD serão executados. Para isso, deve utilizar o seguinte comando: **conda env create -f ccdISA_linux.yml**

```
scaetano@DMmlc3:~$ ssh scaetano@ui.d.incd.pt
Last login: Mon Nov 18 08:59:29 2024 from 193.136.147.150
[scaetano@ui102 ~]$ conda env create -f ccdISA_linux.yml
```

Figura 8 – Criação do ambiente virtual (ccdISA) para processamento do PyCCD no cluster do INCD.

- III. Certifique-se de que o ambiente foi instalado corretamente com o comando: **conda info --envs**. Será exibido uma lista de todos os ambientes disponíveis. Confirme se o ambiente **ccdISA** consta na lista:

```
[scaetano@ui102 ~]$ conda info --envs
# conda environments:
#
ccdISA          /users1/cpca070342024/scaetano/.conda/envs/ccdISA
base            * /usr
```

Figura 9 – Lista dos ambientes virtuais disponíveis no cluster do INCD.

- IV. Uma vez confirmado que o ambiente está instalado, ative-o com o comando: **conda activate ccdISA**. O prompt do terminal será alterado para indicar que o ambiente está ativo, apresentando um formato semelhante a este:

```
[scaetano@ui102 ~]$ conda activate ccdISA
(ccdISA) [scaetano@ui102 ~]$
```

Figura 10 – Ativação do ambiente virtual ccdISA no cluster do INCD.

5. Executar scripts no cluster do INCD

Após a configuração e ativação do ambiente virtual **ccdISA**, a execução de scripts no cluster do INCD é realizada por meio de um ficheiro de submissão chamado **submit_job.sh**, que deve ser criado e armazenado na diretoria correspondente no cluster do INCD (/users1/cpca070342024/user), como ilustrado na Figura 6. Este ficheiro funciona como um guia estruturado, onde especifica os parâmetros essenciais para o agendamento, alocação de recursos e execução do trabalho no SLURM.

A Figura 11 mostra um exemplo de um ficheiro **submit_job.sh** configurado para executar o script principal do PyCCD.


```
#!/bin/bash

#SBATCH --job-name=PyCCD          # Nome do Job
#SBATCH --time=48:00:00          # Tempo de execução (HH:MM:SS)
#SBATCH --nodes=4                # Número de nós
#SBATCH --ntasks-per-node=24     # Número de tarefas por nó
#SBATCH --mem-per-cpu=8000       # Memória por CPU (8 GB por CPU)
#SBATCH --partition=fct          # Partição
#SBATCH --account=cpca070342024  # Conta
#SBATCH --qos=cpca070342024      # Qualidade de serviço (QoS)

# Carregar e executar o script Python com srun
stdbuf -oL python /users1/cpca070342024/scaetano/CCD_yaml_win/S2CHANGE/scripts/pyccd_theia/notebooks/main.py
```

Figura 11 – Exemplo do ficheiro de submissão (submit_job.sh) para execução de um script no cluster do INCD, aberto com o WinSCP.

No ficheiro de submissão **submit_job.sh**, mostrado na Figura 11, encontram-se os principais parâmetros utilizados para configurar o ambiente de execução no SLURM, sendo estes:

- **#SBATCH --job-name:** Define um nome identificador para o projeto. Este nome aparecerá no sistema de monitorização do cluster.
- **#SBATCH --time:** Especifica o tempo máximo de execução. Após esse período, o trabalho será automaticamente encerrado, mesmo que não tenha sido terminado.
- **#SBATCH --nodes:** Determina o número de nós a serem utilizados para o projeto.
- **#SBATCH --ntasks-per-node:** Define o número de tarefas simultâneas que cada nó irá processar.
- **#SBATCH --mem-per-cpu:** Atribui a quantidade de memória para cada CPU. Neste exemplo, são 8 GB por CPU.
- **#SBATCH --partition:** Especifica a partição (grupo de recursos) onde o projeto será executado.
- **#SBATCH --account** e **#SBATCH --qos:** Identificam a conta e os parâmetros que gerenciam os recursos disponíveis para o utilizador.

Para submeter o ficheiro de submissão ao cluster, utiliza-se o comando: **sbatch submit_job.sh**, que envia o job ao sistema de agendamento. Após a submissão, o sistema retorna um identificador único, como no exemplo:

```
(ccdISA) [scaetano@ui102 ~]$ sbatch submit_job.sh
Submitted batch job 40039
```

Figura 12 – Comando utilizado para submeter o ficheiro de submissão submit_job.sh ao SLURM para agendamento e execução do job no cluster e respetiva mensagem retornada pelo SLURM indicando a submissão bem-sucedida do job, com o identificador único 40039.

Este identificador serve para visualizar o status do trabalho e acompanhar a sua execução. Durante a execução do job, o comando: **stdbuf -oL python** do ficheiro **submit_job.sh** é responsável por iniciar o script **main.py** do PyCCD, garantindo que a saída seja exibida em tempo real no terminal. O progresso pode ser monitorizado com o comando: **tail -f slurm-40039.out**, que permite visualizar o conteúdo do arquivo de log gerado pelo SLURM, incluindo mensagens de execução ou possíveis erros:

```
(ccdISA) [scaetano@ui102 ~]$ tail -f slurm-40039.out
* SLURM_NTASKS : 96
* SLURM_CPUS_ON_NODE : 50
* SLURM_TASKS_PER_NODE : 28,29,25,14
* SLURM_JOB_CPUS_PER_NODE : 50,54,47,24
* SLURM_MEM_PER_CPU : 4000
* SUBMIT_HOST : ui102.d.incd.pt
* WORK_DIR : /users1/cpca070342024/scaetano
* JOB_SCRIPT : /users1/cpca070342024/scaetano/submit_job.sh
* -----
Iniciando o script
Diretorio ja existe: C:/Users/Public/Documents/output_BDR300/numpy/T29TNE
Diretorio ja existe: C:/Users/Public/Documents/output_BDR300/plots/T29TNE
Diretorio ja existe: C:/Users/Public/Documents/output_BDR300/tabular/T29TNE
Diretorio ja existe: C:/Users/Public/Documents/output_BDR300/shapefiles/T29TNE
Processar centros dos pontos de cada geometria para corresponder aos centros dos pix
Processar centros dos pixels: 0.34865132967631024 minutos. Número de pixels: 241968
O arquivo 'C:/Users/Public/Documents/output_BDR300/numpy/T29TNE/imagens_Theia-NDVI_X
essando os dados existentes...
A recolher nome e data dos tifs...
241968it [5:20:37, 12.58it/s]
```

Figura 13 – Comando utilizado para monitorizar em tempo real o arquivo de log slurm-40039.out, exibindo as mensagens de execução, erros ou resultados intermediários.

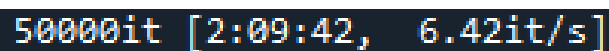
Para uma análise mais detalhada do uso dos recursos do cluster, usa-se o comando: **report 2024-11-10**. Este comando mostra estatísticas sobre a utilização de memória, CPU e outros recursos durante o dia especificado. Além disso, apresenta informações sobre a quota de disco ocupada, permitindo monitorizar o armazenamento utilizado no cluster:

```
[scaetano@ui102 ~]$ report 2024-11-10
Username: scaetano Quota: 9903853 Used: 1073741824
Disk Quota usage in kb
scaetano disk usage 1073741824.0
9903853.0
Allocated vs consumed CPU resources for QOS in minutes
cpca070342024 6000000.0
2154994.88
Consumed energy per qos cpca070342024
Value in Joules: 0
```

Figura 14 – Relatório de utilização de recursos do cluster no dia 2024-10-19, incluindo uso de quota de disco (em KB) e comparação entre os recursos de CPU alocados e consumidos (em minutos).

6. Resultados e conclusões

O objetivo principal deste estudo foi otimizar tanto o tempo de execução quanto a utilização dos recursos disponíveis no processamento de uma amostra de 50.000 pixels da Base de Dados de Referência da DGT. O processamento inicial foi realizado numa máquina local com 6 núcleos e uma velocidade de relógio de 3.2 GHz, onde o tempo necessário foi de aproximadamente 2 horas e 9 minutos, resultando numa taxa de processamento de 6.42 pixels por segundo:



```
50000it [2:09:42, 6.42it/s]
```

Figura 15 – Progresso da execução do PyCCD numa máquina local com 6 núcleos e 3.2 GHz para 50.000 pixels.

Existem três tipos de paralelismo que podem ser aplicados ao código, sendo que dois já foram testados, sendo um deles o atual, e um terceiro será avaliado em trabalho futuro:

- Paralelismo sequencial: inicialmente, foi testado um método sequencial, em que o número total de pixels era distribuído entre os núcleos. No entanto, esta técnica resultou num desempenho significativamente mais lento.
- Paralelização de pixels dentro dos batches: esta é a técnica atualmente implementada no código que divide o número total de pixels em batches menores, definidos pelo parâmetro `batch_size`. Os pixels de cada batch são distribuídos entre os processadores disponíveis no cluster, com o número de núcleos configurado no ficheiro de submissão (**`submit_job.sh`**) por meio do parâmetro `--ntasks-per-node`. O valor inicial de `batch_size` foi definido como 1.000 pixels, que foi o mesmo valor utilizado nos testes da máquina local (Figura 15). A Figura 16 apresenta a parte do código que implementa esta estratégia, utilizando a biblioteca `concurrent.futures` e a função `ProcessPoolExecutor` para gerir o processamento paralelo.
- Paralelização dos batches: embora esta abordagem ainda não tenha sido testada neste estudo, será explorada futuramente. A ideia será distribuir o processamento dos batches de forma paralela, ou seja, processar múltiplos batches simultaneamente, ajustando um valor de `batch_size` adequado. A expectativa é que esta técnica proporcione um aumento na eficiência do processamento.

```

batch_size = 10000 # Número de pixels para processar o lote
num_batches = math.ceil(N / batch_size) # Numero de lotes necessarios
# Executar o processamento em lotes com ProcessPoolExecutor
dfs = []
with ProcessPoolExecutor(max_workers=os.cpu_count()) as executor:
    tqdm_bar = tqdm(total=num_batches)

    for batch_index, start_index in enumerate(range(0, N, batch_size)):
        end_index = min(start_index + batch_size, N)

        # Usar apenas o bloco atual de pontos
        sel_values_block = sel_values[:, :, start_index:end_index]
        xs_slice = xs[start_index:end_index]
        ys_slice = ys[start_index:end_index]

        # Criar argumentos para cada lote de N pixels
        arg_list = [(i, sel_values_block, tif_dates_ord, xs_slice, ys_slice, NODATA_VALUE,
                     MAX_VALUE_NDVI, PASTA_DE_OUTPUTS, CRS_THEIA,
                     CRS_WGS84, img_collection) for i in range(sel_values_block.shape[2])]

        # Mapear os resultados usando executor.map
        for result_df in executor.map(runDetectionForPoint, arg_list):
            dfs.append(result_df)
            tqdm_bar.update(1)

    tqdm_bar.close()

```

Figura 16 – Parte do código responsável pela configuração do processamento paralelo com ProcessPoolExecutor.

O uso do cluster do INCD permitiu uma redução substancial no tempo de processamento do PyCCD em comparação com a máquina local. É importante destacar que os recursos máximos do cluster consistem em até 5 nós e 96 tarefas por nó (ntasks-per-node). Foram realizados diversos testes, variando os parâmetros como o número de pixels, ntasks-per-node e o batch_size, mantendo o número de nós (igual a 1) e a memória por CPU (5333) constantes, com o objetivo de identificar a configuração mais eficiente para o processamento dos dados. A Tabela 1 apresenta os resultados obtidos a partir da paralelização de pixels dentro dos batches, considerando diferentes combinações destas variáveis:

NUMBER OF PIXELS	NTASKS/ NODE (96)	MEMORY/ CPU	TIME	BATCH_SIZE	PROCESSING TIME	CORE MINUTES
50.000	96	5333	24 h	250	00:10:59	1320
50.000	96	5333	24 h	500	00:12:28	1453.8
50.000	96	5333	24 h	1.000	00:17:56	1977.6
50.000	96	5333	24 h	5.000	01:06:40	6664
50.000	24	5333	24 h	250	00:23:00	616
50.000	24	5333	24 h	500	00:21:59	591.6
50.000	24	5333	24 h	1.000	00:26:28	698.4
100.000	96	5333	24 h	250	00:22:16	2632
100.000	96	5333	24 h	1.000	00:37:29	4073.6
241.968	96	5333	24 h	250	00:49:33	5928
241.968	96	5333	24 h	1.000	01:31:38	9895.6

Tabela 1 – Testes e configurações para a otimização do tempo de processamento para conjuntos de 50.000, 100.000 e 241.968 pixels, realizados com a técnica de paralelização dos pixels dentro dos batches. Os testes foram realizados utilizando apenas um nó dos cinco disponíveis no cluster, explorando duas configurações de tarefas: o máximo permitido de 96 ntasks-per-node e uma configuração intermediária de 24 tasks-per-node.

A configuração com 96 tarefas por nó mostrou-se mais eficiente do que com 24 tarefas porque, ao aumentar o número de tarefas, os núcleos conseguem trabalhar continuamente sem precisar esperar que outras tarefas sejam concluídas para começar um novo processamento. Isto garante que os núcleos estejam sempre ocupados, processando dados de forma eficiente e evitando períodos de inatividade, o que maximiza a utilização dos recursos computacionais disponíveis.

O impacto do parâmetro `batch_size` também foi significativo. Valores menores de `batch_size`, como 250, resultaram em tempos de execução substancialmente mais rápidos, pois reduziram a sobrecarga de memória em cada tarefa, facilitando a gestão dos recursos do cluster. Por outro lado, valores maiores de `batch_size`, como 5.000, causaram uma sobrecarga maior nos recursos de memória e aumentaram o tempo de execução. Por exemplo, com 96 tarefas e `batch_size` = 5.000, o tempo de processamento subiu para 1 hora e 6 minutos, enquanto com `batch_size` = 250, o tempo foi mantido em 10 minutos e 59 segundos, evidenciando a importância de um `batch_size` menor para melhorar a eficiência.

O parâmetro `core minutes`, que representa o total de recursos de CPU utilizados durante o processamento, demonstrou que a configuração com 96 tarefas e `batch_size` de 250 consumiu 1320 core minutes, significativamente mais que os 616 core minutes com 24 tarefas. Embora o tempo de processamento tenha sido menor na configuração com 96 tarefas (10 minutos e 59 segundos), o uso maior de núcleos resultou num maior consumo de recursos. Em contrapartida, a configuração com 24 tarefas, apesar de ter levado mais tempo (23 minutos), utilizou menos núcleos, o que resultou num menor consumo de core minutes.

A partir dos testes realizados para 50.000 pixels, é possível extrapolar os tempos de processamento e os core minutes para um cenário com 120 milhões de pixels. Esta projeção considera as configurações de `ntasks-per-node` de 96 e 24, mantendo o `batch_size` igual a 250:

NTASKS/NODE	PROCESSING TIME	CORE MINUTES
96	438 h e 16 mins	3.168.000
24	919 h e 12 mins	1.478.400

Tabela 2 – Resultados escalados para 120 milhões de pixels, considerando diferentes configurações de `ntasks-per-node`. O tempo de processamento total e os minutos acumulados de processamento por núcleo são extrapolados a partir de testes com 50.000 pixels.

Os resultados apresentados na Tabela 2 destacam que a escolha da configuração ideal depende do objetivo: tempo de processamento ou eficiência no uso de recursos computacionais. Se o objetivo for minimizar o tempo total de processamento para 120 milhões de pixels, a configuração com 96 `ntasks-per-node`

é a mais indicada, resultando num tempo estimado de 438 horas e 16 minutos, embora com maior consumo de core minutes (3.168.000).

Por outro lado, caso a prioridade seja a eficiência no uso de recursos computacionais, a configuração com 24 ntasks-per-node é mais vantajosa. Apesar de demorar mais tempo de processamento (919 horas e 12 minutos), o consumo de core minutes é significativamente menor, com 1.478.400 core-minutes estimados.

7. Questões em aberto

Como trabalho futuro será explorada a possibilidade de expandir o processamento para mais nós, ultrapassando a limitação atual de execução num único nó. Esta expansão será importante para identificar novas oportunidades de otimização, visando reduzir ainda mais o tempo de execução e maximizar a eficiência computacional. Paralelamente, será analisada uma mudança na estratégia de paralelização, considerando a execução paralela de batches em vez de paralelizar os pixels dentro de cada batch. Este método poderá otimizar a distribuição de tarefas e reduzir o *overhead*.

8. Anexos

De seguida, encontram-se listados alguns comandos utilizados no SLURM, acompanhados das respetivas descrições:

- **squeue -u <username>** exibe a lista de jobs ativos do utilizador especificado:

```
[scaetano@ui102 ~]$ squeue -u scaetano
```

JOBID	PARTITION	NAME	USER	ST	TIME
-------	-----------	------	------	----	------

- **sbatch <script>** submete um job para execução a partir de um script de submissão:

```
(ccdISA) [scaetano@ui102 ~]$ sbatch submit_job.sh
```

- **scancel <JOBID>** cancela a execução de um job específico, utilizando o seu identificador (JOBID):

```
(ccdISA) [scaetano@ui102 ~]$ scancel 40036
```

- **sinfo** exibe informações sobre as partições e nós disponíveis no cluster:

```
[scaetano@ui102 ~]$ sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES
fct	up	4-00:00:00	1
fct	up	4-00:00:00	5
fct	up	4-00:00:00	1
fct	up	4-00:00:00	29
hpcbigmem	up	4-00:00:00	1
hpcbigmem	up	4-00:00:00	1