INF2610 – Noyau du système d'exploitation

Laboratoire 4 – Hiver 2015

API Windows

|  |  |
| --- | --- |
| Nom, prénom, matricule des membres de l'équipe | Félix La Rocque Carrier, 1621348  Mathieu Gamache, 1626377 |
| Note finale sur 20 | 0 |

# Directives

* Il est suggéré de faire l'activité en équipe de deux. Inscrivez les noms ci-haut.
* Répondez directement dans le questionnaire ODT avec Libre Office.
* La correction se fera directement dans le document électronique.
* Remettez sur Moodle le questionnaire rempli et le code.
* Une seule personne de l'équipe doit effectuer la remise sur Moodle.
* Utilisez les machines du laboratoire pour obtenir vos résultats.
* 10% de la note finale peut être enlevée pour la mauvaise qualité de la langue.

# Mise en situation

Vous travaillez sur *iEffect*, une application de traitement d'images. Cette application a pour but d'appliquer plusieurs effets successifs sur un grand nombre d'images. Ce type de traitement peut se faire plus rapidement **en utilisant un pipeline à plusieurs fils d'exécution.**

La figure 1 représente le traitement de 3 images, sur lesquelles on applique 2 effets. Dans le cas du traitement en pipeline, **un fil d'exécution est démarré pour chaque étape de traitement**, ce qui permet de diminuer le temps total de traitement du lot d'images comparativement à une exécution en série.

Illustration 1: Exemple du traitement en série et en pipeline selon le temps

Les étapes de traitement sont reliées par un objet *ImageQueue*, tel que représenté à la figure 2. *ImageQueue* est une **queue FIFO bloquante**. Lorsque la queue est vide, **le consommateur bloque et attend une image**. Lorsque la queue est pleine, **le producteur bloque en attendant qu'il y ait de l'espace**. On limite la taille de chaque *ImageQueue* pour borner le nombre simultané d'images en mémoire.

Illustration 2: Organisation du pipeline

**Une étape de traitement peut agir comme consommateur, producteur ou les deux**. La première étape du traitement est LoadImage, qui charge l'image en mémoire à partir d'un fichier. Cette étape agit comme productrice dans la chaîne. La dernière étape est SaveImage. Elle enregistre le résultat final et agit en tant que consommatrice. Les étapes EffectStage, situées au centre du pipeline et dont le rôle est d'appliquer un effet sur l'image, agissent comme consommatrices de l'étape précédente et comme productrices pour l'étape suivante.

Lorsque la file est vide, alors le consommateur bloque et attend qu'un élément soit produit. Lorsque la file est pleine, alors le producteur bloque et attend qu'un élément soit libéré.

# Implémentation de iEffect

Votre mission est de compléter iEffect pour réaliser le traitement en pipeline. Deux classes sont à compléter.

* ImageQueue : implémenter la queue FIFO bloquante **à l'aide de sémaphores** (implémentation des méthodes *queue* et *dequeue*.
* PipelineManager : créer les fils d'exécution **pour chaque étape de traitement** PipelineStage en appliquant ce qui a déjà été fait pour Linux au cas de Windows.

Dans le cadre du laboratoire, l'implémentation doit être réalisée **avec l'API Windows**, incluant les sémaphores, les verrous et le démarrage des fils d'exécution.

## Utilisation avec QtCreator

Voici les étapes à suivre.

1. Décompresser les sources dans [C:\Temp](file:///C:\C:\Temp)
2. Changer de répertoire pour la base des sources
3. Ouvrir le fichier inf2610-lab4.pro.
4. Configurer le projet en choisissant les paramètres par défaut.
5. Ctrl-B pour compiler le projet.

Le programme ieffect nécessite des arguments pour son exécution. Voici les étapes à suivre pour définir les arguments à passer.

1. Dans l'onglet « Projects », sélectionner les configurations « Run ».
2. Sélectionner le programme à exécuter sous « Run configuration ».
3. Inscrire les arguments (sans la commande elle-même).

Voici les arguments acceptés par ieffect :

--input PATH Chemin d'accès des images à traiter

--output PATH Chemin d'accès des images finales

--effects eff1,eff2 Liste des effets à appliquer, séparés par des virgules

--queue-size INT Taille maximale des files d'attentes

Les effets sont appliqués dans l'ordre spécifié. La liste des effets est détaillée dans le tableau suivant.

|  |  |
| --- | --- |
| Nom de l'effet | Description |
| *blur* | Applique un filtre gaussien (ajoute du flou) |
| *brightness* | Augmente la luminosité |
| *cool* | Rend l'image bleutée |
| *greyscale* | Rend l'image en noir et blanc |
| *saturation* | Augmente la saturation |
| *sharpen* | Applique un filtre qui accentue les contours |
| *thumbnail* | Crée un vignette de 50 par 50 pixel maximum |
| *warm* | Rend l'image plus rougeâtre |

Pour les fins du laboratoire, un jeu d'images à traiter est fourni (archive *imageset.zip*). Voici un exemple d'arguments à passer dans QtCreator pour traiter les images fournies (ajuster au besoin les chemins d'accès vers les images de départ) :

--input $(ProjectDir)\..\..\imageset\small --output results --effects cool,saturation,sharpen --queue-size 2

Astuce : pour réduire le temps d'exécution pendant le développement, vous pouvez utiliser les images miniatures plutôt que les images pleine grandeur, ou créer un répertoire avec seulement quelques images.

## Correction

|  |  |  |
| --- | --- | --- |
| * 1. Implémentation correcte des classes ImageQueue et PipelineManager. |  | / 12 pt |

# Observation de l'exécution

La performance d'un pipeline est affectée par l'étape la plus lente de la chaîne. Dans votre implémentation et pour un traitement en particulier, quelle est l'étape qui limite la performance?

Pour le savoir, **l'objet ImageQueue est instrumenté pour enregistrer le nombre d'images en attente selon le temps**. Les traces sont enregistrées dans un répertoire comportant le moment de l'exécution. Pour charger la trace, utilisez l'outil qview fourni avec les sources du laboratoire.

De plus, la variation dans la taille des données à traiter (dans ce cas la taille des images) peut produire un effet néfaste sur la performance. L'utilisation de files d'attente permet de mitiger cet impact. Vous pouvez observer cet effet en utilisant différentes tailles de files sur les images du dossier « mix ».

Effectuez les expériences suivantes sur les images de tailles mixtes (dossier « mix ») et comportant 4 effets de votre choix, sauf thumbnail.

1. Fixer la taille des files d'attente à 1.
2. Fixer la taille des files d'attente à 10.
3. Ajouter l'étape thumbnail dans la chaîne du traitement.

Insérez ci-bas dans la section « résultats et analyse » les captures d'écran des traces ouvertes dans qview pour chacune des expériences. Inscrivez les effets appliqués. Analysez les résultats par rapport au délai d'exécution, la taille des files d'attentes dans le temps et autres observations que vous pouvez déduire de l'observation. Votre analyse doit comporter un maximum de trois pages, incluant les captures d'écran en lien avec les explications.

## Résultats et analyse

Dans la première expérience, nous avons choisi d’appliquer les filtres : “Brightness”, “Sharpen”, “Saturation” et “Warm”. Css filtres ont été appliqué au dossier d’image mixte contenant de grandes et de petites images et la taille maximale de la file d’attente a été initialiser à 1. Le résultat de la trace est le suivant :

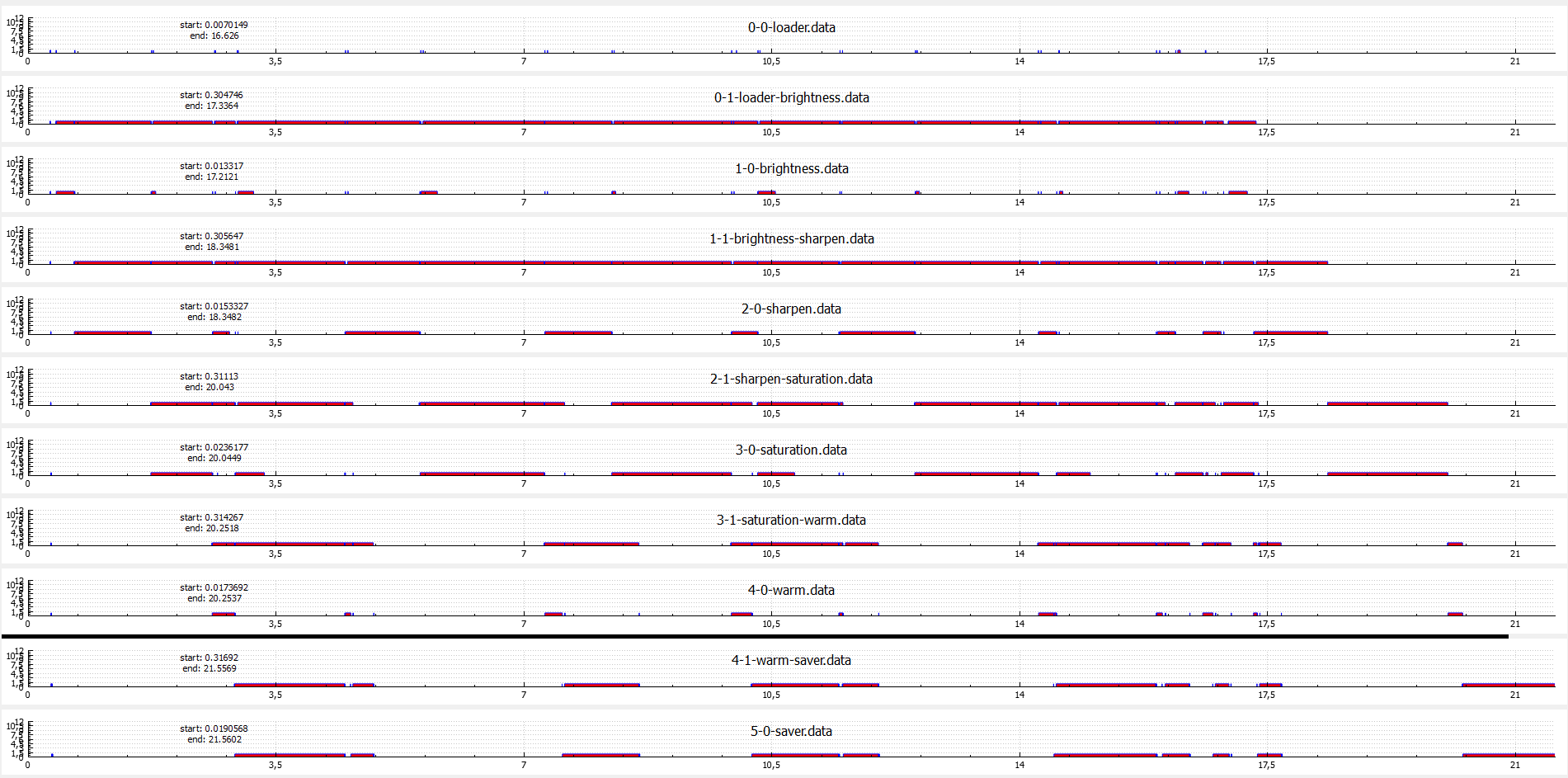


Figure 1: Expérience 1 - 4 filtres sans thumbnail, taille mixte, file max = 1

Nous pouvons clairement voir l’effet de la taille de la file de 1 dans les “loaders”:

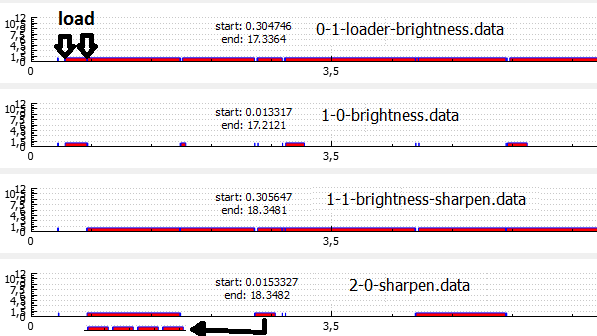


Figure 2: Expérience 1 - Zoom sur les deux premiers filtres et leur file d’attente

Dans la figure 2, on peut observer le chargement de la première image dans la file du premier filtre et l’attente de la terminaison de ce filtre (passer l’image à la prochaine file) avant le chargement de la prochaine image (file limitée à 1). Cette file limitée à 1 avec la taille variable des images nuit grandement à la capacité de parallélisme, puisqu’il y a un bouchon qui se forme à chaque traitement de grande image. Par exemple, ans le filtre “Sharpen” de la figure plus haute, on observe que traiter une grande image prend environ plus de 4 fois le temps d’une petite. Dans la situation où l’étape 2 est en train de traiter une grosse image, l’étape 1 qui pourrait en profiter pour traiter plusieurs petites images ne peut en traiter 1 seul. De l’autre côté, il se pourrait que ça soit l’étape 2 qui épuise rapidement sa pile d’images à traiter et attende l’étape 1. Il en résulte comme le montre la figure 1, qu’il a plusieurs moments où une étape du pipeline est inactive (on perd du temps). On en déduit dont qu’avec une pile de seulement de 1 le temps d’exécution dépend énormément de l’ordre dans lequel les images sont traitées.

Dans la deuxième expérience, nous avons utilisé les mêmes paramètres définissant les types de traitements d’image à appliquer et avons changé la taille de la file d’attente à 10. Les résultats sont les suivants :

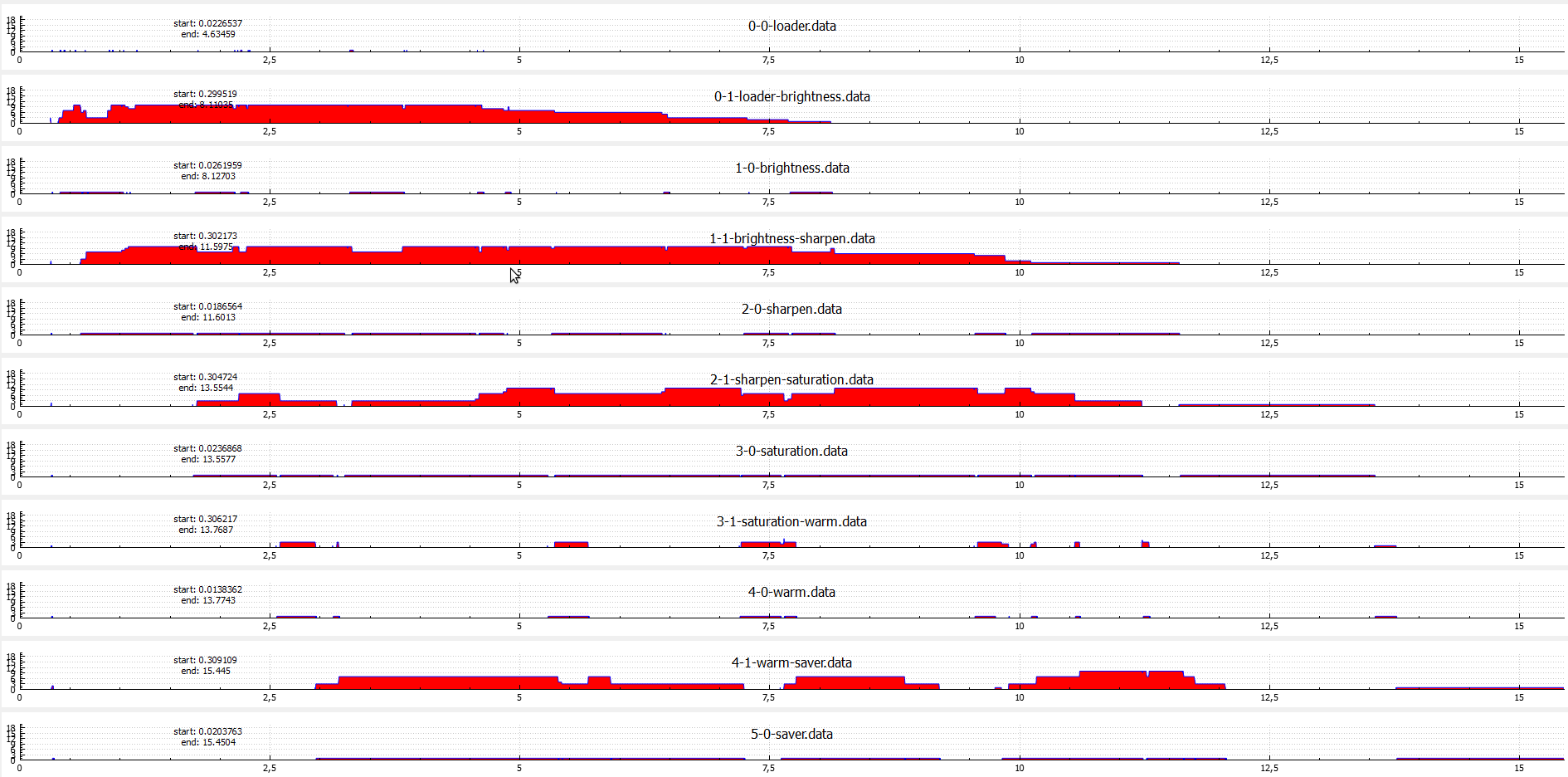


Figure 3: Expérience 2 - 4 filtres sans thumbnail, taille mixte, file max = 10

Contrairement à l’expérience 1, nous pouvons qu’il y a beaucoup moins de perte de temps, car chaque étape du pipeline est presque toujours active. Il en résulte qu’es opérations ont pris 15 secondes versus 21 secondes dans l’expérience 1. Le principe est que la pile permet de diminuer l’impact dans l’ordre que les images soient traitées, même si l’étape 2 traite une grande image, l’étape 1 à la possibilité de traiter plusieurs images, et lorsque l’étape 1 sera rendue à une grande image, l’étape 2 va pouvoir travailler sur sa pile qui s’est accumulée.

Par contre, comme dans l’expérience 1, chaque opération (filtre) est encore limiter son précédent et son suivant.

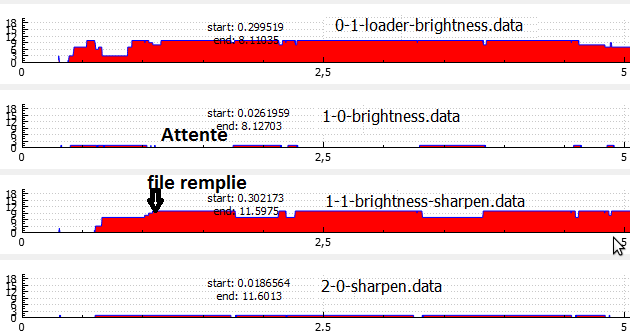


Figure 4: Expérience 2 - Zoom sur les deux premiers filtres et leur file d’attente

On peut constater qu’il y a des goulots d’étranglement, cela se remarque, car leur file d’attente se remplit, bloquant ainsi l’étape précédente. Par exemple l’étape “sharpen” ralentie brightness et l’étape “saturation” ralentie “sharpen”. On remarque aussi que l’étape de sauvegarde est assez coûteuse, car ça file ce rempli.

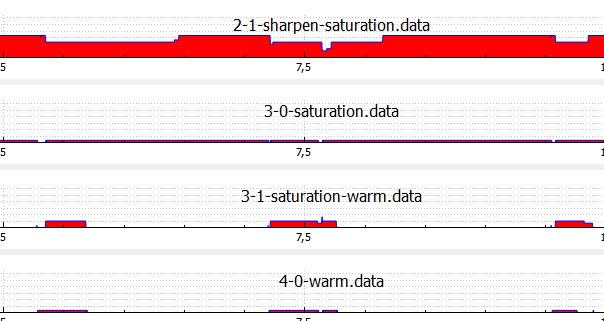


Figure 5 : Expérience 2 - Blocage d’une opération vite par une opération lente

Dans l’expérience 3, nous avons utilisé la même taille de file d’attente ainsi que les mêmes filtres qu’à l’expérience 2 mais avons rajouté l’option de “thumbnail”. Le résultat est le suivant :

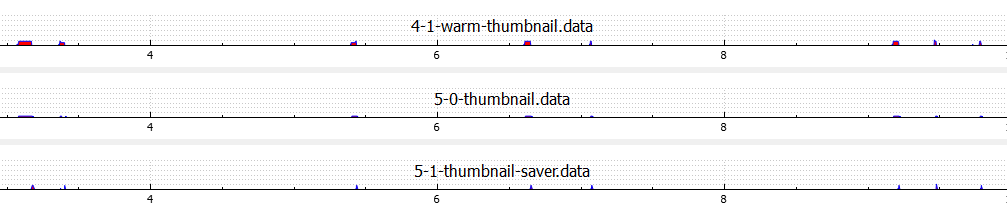


Figure 6: Expérience 3 - 4 filtres avec thumbnail, taille mixte, file max = 10

Au niveau des attentes pour chaque opération avant celle du thumbnail, il n’y a pas de différence entre l’expérience 2 et 3, ce qui est logique, car les premiers filtres traite la même image (elle n’a pas encore été réduite par l’étape thumbnail). Par contre on remarque que la sauvegarde est beaucoup plus rapide, il n’y a plus d’étranglement à cette étape, ce qui veut dire que si on avait placé la transformation thumbnail au début, toutes les étapes suivantes auraient été accélérées, car ils auraient traité une image beaucoup plus petite.

## Correction

|  |  |  |
| --- | --- | --- |
| * 1. Capture d'écran des expériences. |  | / 4pt |
| * 1. Analyse des résultats. |  | / 4pt |

# Correction

|  |  |
| --- | --- |
| Points des questions 1 à 2 | 0 |
| Pénalité de retard | 0 |
| Qualité de la langue (-0,5% par faute) | 0 |
| Points bonus | 0 |
| Note finale sur 20 | 0 |