Université de Caen Basse-Normandie
U.F.R. des Sciences
Département d'informatique

Bâtiment Sciences 3 - Campus Côte de Nacre
F-14032 Caen Cédex, FRANCE

Examen terminal

Niveau	**L3**
Parcours	**Informatique**
Unité d'enseignement	**INF5E - Informatique Industrielle**
Élément constitutif	**INF5E2 - Parallélisme**
Responsables	**Emmanuel Cagniot** (**emmanuel.cagniot@ensicaen.fr**) **Alexandre Niveau** (**alexandre.niveau@unicaen.fr**)
Session	$1^{\text{ère}}$ **session** 2020 – 2021
Durée	**1h30**
Documents	**Tout document autorisé**
Consignes	**Chaque candidat doit, au début de l'épreuve, porter son nom dans le coin de la copie qu'il cachera par collage après avoir pointé. Il devra, en outre, porter son numéro de place sur chacune des copies, intercalaires, ou pièces annexées.**

1 Exercice (3 pts)

Les quatre exercices de cet énoncé sont indépendants. Répondez sur un simple fichier texte (uniquement du .txt) via votre éditeur de texte préféré (geany, notepad, etc.) en indiquant à chaque fois l'exercice abordé et le numéro de la question traitée.

Lorsque le résultat d'un calcul est demandé, vous pouvez l'exprimer sous la forme d'une simple fraction en faisant attention à la compatibilité des unités de mesure (inutile de recourir à la calculatrice). Le barème est donné à titre indicatif.

La figure 1 présente un résultat obtenu par la commande `perf` étudiée dans le TP n°1 (ici sur la commande UNIX `ls`). Certaines informations (les calculs effectués par `perf` et placés à la fin de chaque ligne) ont volontairement été supprimées.

1.1 Question (1 pt)

Calculez le taux d'utilisation du CPU.

1.2 Question (1 pt)

Calculez la fréquence en Hz à laquelle a travaillé le CPU.

```
Performance counter stats for 'ls' (30 runs):

          0,392846      task-clock (msec)
                 0      context-switches
                 0      cpu-migrations
                95      page-faults
         1 601 850      cycles
         1 480 635      instructions
           289 349      branches
            11 409      branch-misses

       0,000505496 seconds time elapsed
```

FIGURE 1 – Exploitation du service `stat` pour la commande UNIX `ls`.

1.3 Question (1 pt)

Calculez le nombre moyen d'instructions exécuté au cours d'un cycle CPU.

2 Exercice (4 pts)

La fonction canonique (sans optimisation) `blas_sdsdot` de la figure 2 permet de calculer puis retourner la somme $\alpha + x^T y$ pour deux vecteurs x et y, leurs composantes étant de type `float` (réels en simple précision).

```
float
blas_sdsdot(float alpha, float x[], float y[], int n) {
  float acc = 0.0;
  for (int i = 0; i < n; i ++) {
    acc = acc + (x[i] * y[i]);
  }
  return alpha + acc;
}
```

FIGURE 2 – Fonction canonique `blas_sdsdot`.

Nous disposons d'un processeur à la fois superscalaire et équipé d'une unité d'exécution SIMD. Par conséquent, comme dans le TP n^{o}1, nous souhaitons optimiser notre fonction via la technique du déroulage de boucle et celle de la vectorisation. Nous supposons ici que la taille n des vecteurs x et y est un multiple de 4.

2.1 Question (2 pts)

Écrivez la fonction `blas_sdsdot_r4` qui représente la forme optimisée par déroulage de boucle sur une profondeur de 4.

Comme dans le TP n^{o}1, nous supposons l'existence du type union `xmm_t` défini comme dans la figure 3.

2.2 Question (2 pts)

En vous aidant du type `xmm_t` présenté ci-dessus, écrivez la fonction `blas_sdsdot_sse_r4` qui représente la forme optimisée par vectorisation dans le jeu d'instruction SSE.

```
/*
 * Union permettant d'accéder aux quatre nombre flottants simple précision
 * compactés dans un registre 128 bits.
 */
typedef union {
  __m128 m128_vec;     // Le registre.
  float  m128_f32[4]; // Ce même registre vu comme un tableau de taille 4.
} xmm_t;
```

FIGURE 3 – Type union `xmm_t`.

3 Exercice (6 pts)

Considérons une séquence $\mathcal{I}_n = \{op_1, op_2, \cdots, op_n\}$ constituée de n occurrences d'une même opération élémentaire *op*. Cette opération peut être exécutée sur un opérateur pipeline à trois niveaux (un étage par niveau) et dont la table de réservation est donnée par :

	1	2	3	4
stage$_3$				×
stage$_2$		×	×	
stage$_1$	×			

Cette table indique que les durées de traversée des étages **stage**$_1$, **stage**$_2$ et **stage**$_3$ valent respectivement 1τ (1 top d'horloge), 2τ et 1τ.

Comme dans le TD n^{o}1 nous allons caractériser cet opérateur.

3.1 Question ($0,5$ pt)

Donnez l'expression de la durée t_1 qui représente la durée d'exécution de la séquence $\mathcal{I}_n$ en mode séquentiel.

3.2 Question (1 pt)

Donnez l'expression de la durée t_2 qui représente la durée d'exécution de la séquence $\mathcal{I}_n$ en mode pipeline.

3.3 Question ($1,5$ pts)

Donnez l'expression du facteur d'accélération (*speedup*) s_3^n (3 niveaux d'étages, n opérations) et celle de sa limite s_3^∞.

3.4 Question (3 pts)

Montrez que le compilateur exploite systématiquement le mode pipeline de notre opérateur (quelle que soit la longueur n de la séquence $\mathcal{I}_n$) et jamais le mode séquentiel.

4 Exercice (7 pts)

La figure 4 présente la définition d'une fonction `count_if` permettant de comptabiliser puis de retourner le nombre d'éléments d'un tableau d'entiers satisfaisant une condition particulière.

Plus précisément, les paramètres `array` et `n` représentent respectivement le tableau et sa taille. Le paramètre `pred` représente quant à lui un pointeur vers toute fonction dont la signature est de la forme :

```
bool an_unary_predicate(int e);
```

une telle fonction étant appelée « prédicat unaire ».

L'utilisation d'un pointeur de fonction fait que `count_if` peut être utilisée avec n'importe quel prédicat pourvu que celui-çi possède la signature attendue.

```
1  int
2  count_if(const int array[], int n, pred_ptr_t pred) {
3
4    // Le compteur.
5    int acc = 0;
6
7    // Boucle de parcours des éléments du tableau.
8    for (int i = 0; i < n; i ++) {
9
10     // Si l'élément courant satisfait le prédicat alors incrémenter le  compteur.
11     if (pred(array[i])) {
12       acc ++;
13     }
14
15   }
16
17   // C'est terminé.
18   return acc;
19
20 }
```

FIGURE 4 – Fonction `count_if`.

Nous souhaitons écrire une version OPENMP de la fonction `count_if`.

4.1 Question (1,5 pts)

Montrez que la boucle `for` est de type `forall`.

4.2 Question (1,5 pts)

Montrez que la nombre d'itérations à répartir sur l'ensemble des threads disponibles peut être précalculé par l'implémentation OPENMP du compilateur utilisé.

Les deux questions précédentes permettent de démontrer que notre boucle `for` est parallélisable en OPENMP. Il nous faut à présent déterminer comment répartir ses itérations entre les threads disponibles afin que tous puissent terminer « à peu près » en même temps (principe de l'équilibrage de la charge de travail). Pour cela, il faut s'intéresser à la nature du corps de cette boucle.

Il existe quatre cas de figure possibles :
— le corps de boucle est homogène c'est à dire que toutes les itérations ont exactement la même durée d'exécution ;
— il est quasi-homogène c'est à dire qu'il existe bien des petites variations mais rien de bien méchant ;
— il est hétérogène c'est à dire qu'il existe de fortes variations d'une itération à une autre ;

— on ne peut pas conclure car les informations fournies par le seul examen du code source sont insuffisantes.

Dans le cas homogène ou quasi-homogène, la clause `schedule(static)` est utilisée (les N itérations de la boucle sont réparties sur les P threads disponibles à raison d'un bloc de N/P itérations consécutives par thread).

Dans le cas hétérogène c'est la clause `schedule(dynamic)` qui est utilisée : les threads vont chercher une itération à exécuter (n'importe laquelle), l'exécutent puis vont en cherche une autre.

Dans le dernier cas c'est la clause `schedule(auto)` qui est utilisée c'est à dire que vous vous en remettez au compilateur en espérant qu'il fasse le bon choix.

4.3 Question (2 pts)

À quel cas de figure correspond notre boucle `for` (justifiez très précisément votre réponse).

4.4 Question (2 pts)

Écrivez la fonction `omp_count_if` (juste ses instructions) qui représente notre version OpenMP de la fonction `count_if`.