TD4 – Vectorization & Mémoire

Ce TD a pour but d'apporter certains éléments clés à la compréhensions des mécanismes déployés par les architectures modernes afin de gérer le traffic et les intéractions mémoire.

Introduction:

La mémoire utilisée dans les architectures modernes apparait sous plusieurs formes: disque dur mécanique, disque SSD, clé USB, mémoire RAM, mémoire cache, …
Ces supports de stockage sont basés sur des principes électroniques conçus autour de métriques spécifiques: la vitesse de réponse, la capacité de stockage, le coût des composants électroniques de base, la surface de couverture du circuit, la durée de stockage (persistence), …
Chaque type de mémoire cible une problématique spécifique. Par exemple, la mémoire vive, ou DRAM (Dynamic Random Access Memory), conçue en utilisant des bascules D dynamiques (**Fig.1**) alignées en plusieurs lignes, permet des tailles de stockage raisonnables (plusieurs Giga Octets) pour un coût financier faible à la conception.

La bascule D qui équipe la DRAM présentée sur **Fig.1** est très peu coûteuse en surface et en finances car il ne faut que deux composants (1 transistor et 1 condensateur) pour la réaliser. Malheureusement, ce circuit a un temps de réponse élevé. Chaque lecture vide le condensateur (*Storage capacitor*) stockant la charge électrique représentant le bit logique. Afin de garantir la persistence des données (éviter d'effacer les données à chaque lecture), le condensateur doit être donc rechargé à chaque lecture du bit. C'est cette opération de rafraîchissement qui rajoute de la latence au fonctionnement de la DRAM. Ce rafraîchissement cause aussi des problèmes de fuites électriques qui peuvent se transformer en failles de sécurité (*ref. RowHammer Attack*).

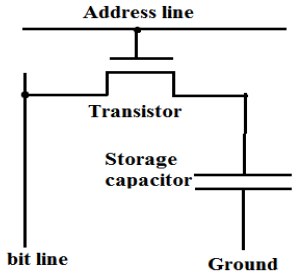


Fig.1 – Cellule DRAM ou bascule **D** dynamique

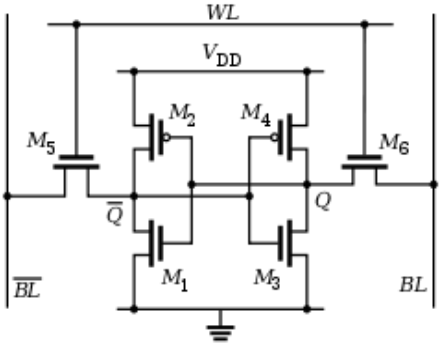
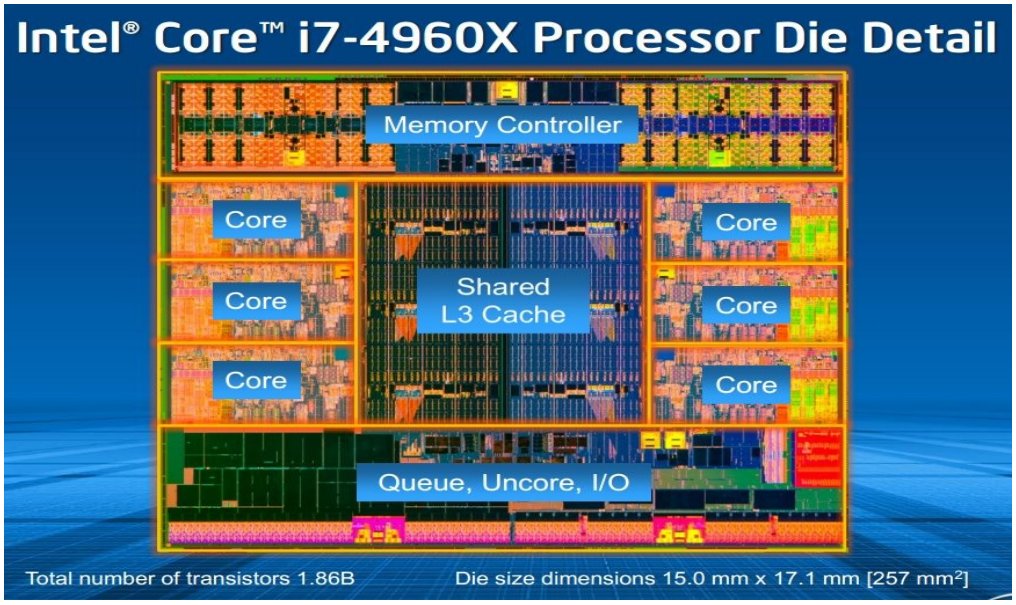


Fig.2 – Cellule SRAM ou bascule **D** statique

Il existe un autre type de mémoires dit statique, SRAM, qui permet de pallier ce problème de rafraîchissement.
Fig.2 représente une cellule SRAM. On peut clairement constaté la complexité du circuit (6 transistors branchés en boucle) comparé à celui présenté sur **Fig.1**. Ce circuit est plus stable électriquement et exhibe un temps de réponse plus rapide que le précédent, mais il nécessite trois fois plus de composants et d'espace pour réaliser la même tâche. C'est pour cette raison que les SRAM ne sont déployées que pour implémenter les mémoires de petites tailles mais à tâches critiques. Par exemple, les caches et registres des processeurs.

Exercice 1:



Un processeur Intel Core i7 Ivy Bridge E à 6 coeurs contient **1,860,000,000** transistors. Chaque coeur contient deux caches privés, un premier cache **L1** de **32KB** et un second cache **L2** de **256KB**. Chaque coeur a aussi accès à un cache **L3** de **8MB** partagé avec les autres coeurs. Les mécanismes de contrôle des caches coûtent eniron **90,500,000** transistors.

1) Sachant qu'un cache est conçu avec des cellules SRAM (1octet = 8 bits = 8 cellules SRAM), calculer le nombre de transitors utilisés pour implémenter les caches sur le processeur cité plus haut. Donner son pourcentage.

2) Sachant que le processeur couvre une surface de 257 mm², quelle est la surface globale couverte par les caches du processeur cité plus haut? Donner son pourcentage.

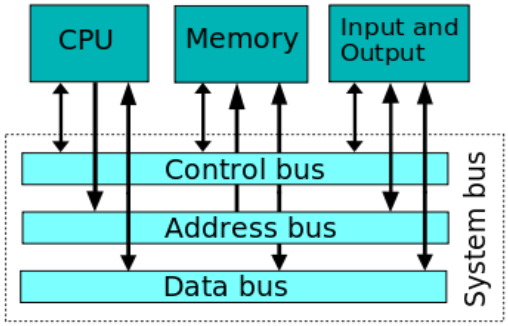


Fig.3 – Schéma des bus utilisés

La mémoire est généralement accessible à travers deux instructions: **load** (charger) & **store** (stocker). Plusieurs variantes existent (load d'entier, load de flottant, …), mais en réalité, le contrôleur mémoire, qui se trouve entre le processeur et la mémoire, ne voit que deux types de traffics: ascendants (**load**) ou descendants (**store**).
Ce contrôleur mémoire opère sur trois bus ou cannaux de communication. Le premier canal est le bus de commande (***Command bus*** ou ***Control bus***) sur lequel le processeur enverra sa requête: **load** ou **store**. Le second canal est le bus d'adressage (***Address bus***) sur lequel circule *l'emplacement* (ou l'adresse) de la zone mémoire ciblée. Le troisième est dernier canal est le bus de données (***Data bus***) utilisé pour transférer la valeur stockée (ou à stocker en cas de **store**) à l'emplacement spécifié sur le bus d'adressage.

En architecture, on évalue généralement la performance de la mémoire en utilisant la notion de bande passante. La bande passante peut être définie comme la quantité de données lues ou écrites depuis une mémoire par un processeur. Elle est souvent exprimée en ***Octets par unités de temps*** (bytes per second, bytes per cycle, …). Autrement dit, la bande passante permet d'évaluer le traffic circulant sur le bus de données.

Exercice 2:

```
for (i = 0; i < 100.000; i++)
    r += t[i];
```

Listing.1 – Code de réduction d'un tableau

```
for (i = 0; i < 100.000; i++)
    a[i] = b[i];
```

Listing.2 – Code de copie de tableaux

```
for (i = 0; i < 100.000; i++)
    c[i] += a[i] * b[i];
```

Listing.3 – Code Multiply-Add

Étant donné les informations suivantes:

Instructions scalaires

LOADW R, B(R, R)	→ 4 cycles CPU
Deux LOAD pipelinés	→ 7 cycles CPU
Quatre LOAD pipelinés	→ 10 cycles CPU
STORW B(R, R), R	→ 7 cycles CPU
Deux STORE pipelinés	→ 7 cycles CPU
Quatre STORE pipelinés	→ 10 cycles CPU

ADD R, R, R → 2 cycles CPU
Deux ADD pipelinés → 2 cycles CPU
Quatre ADD pipelinés → 4 cycles CPU

MOVW R, IMM → 1 cycle CPU
Deux MOV pipelinés → 1 cycle CPU
Quatre MOV pipelinés → 2 cycles CPU

MUL R, R → 1 cycle CPU
Deux ADD pipelinés → 1 cycle CPU
Quatre ADD pipelinés → 2 cycles CPU

Instructions vectorielles

LOADV V, B(R, R) → 6 cycles CPU
Deux VLOAD pipelinés → 10 cycles CPU
Quatre VLOAD pipelinés → 13 cycles CPU

STORV B(R, R), V → 8 cycles CPU
Deux VSTORE pipelinés → 14 cycles CPU
Quatre VSTORE pipelinés → 20 cycles CPU

ADDV V, V, V → 4 cycles CPU
Deux ADD pipelinés → 4 cycles CPU
Quatre ADD pipelinés → 6 cycles CPU

MOVV V, V → 2 cycles CPU
Deux MOV pipelinés → 2 cycles CPU
Quatre MOV pipelinés → 3 cycles CPU

MULV V, V, V → 2 cycles CPU
Deux ADD pipelinés → 2 cycles CPU
Quatre ADD pipelinés → 4 cycles CPU

Instructions de branchement

JMP label → 8 cycles CPU
JEQ label → 10 cycles CPU
JNE label → 10 cycles CPU
JLT label → 10 cycles CPU
JGT label → 10 cycles CPU
JLE label → 10 cycles CPU
JGE label → 10 cycles CPU

– R est un registre de 4 octets (32 bits).
– V est un vecteur de 16 octets (128 bits).

1) Donner le code assembleur scalaire optimal pour chacun des listings précédent.
2) Evaluer le coût, en cycles, de chacun des codes.
3) Calculer la bande passante de chacun des codes.
4) Calculer la bande passante des versions déroulées deux fois et quatre fois.
5) Donner le code vectoriel pour chacun des listings et évaluer sa bande passante sans déroulage et avec un déroulage 2 et 4 pour des éléments de 4 octets seulement.

Exercice 3:

Un cache est une mémoire (SRAM) locale au CPU et est principalement utilisé afin d'exploiter la localité temporelle et spaciale des données. La localité temporelle peut être définie comme le temps (en cycles) entre deux accès mémoires. Par ailleurs, la localité spatiale peut être définie comme la distance entre les addresses de deux accès mémoire.

Le cache-blocking est une technique de programmation permettant de tailler l'accès aux données selon des paramètres relatifs à la configuration des caches du processeur cible.

1) En supposant un cache L1 de 32kB, effectuer un blocking des versions vectorielles déroulée des codes précédents.

Exercice 4:

1) Donner le code C d'une multiplication de deux matrices (GEMM) ainsi que son équivalent en assembleur.
2) Optimiser votre code en applicant: vectorization, déroulage et cache blocking.