Tutoriel d'explications sur les réglages du kernel Android

(Ce tutoriel est réalisé par Mikiya, en se basant sur l'énorme travail de droidphile sur XDA, merci à lui !)

De nombreuses questions reviennent couramment dans les forums suites à un flash :

« Bon maintenant que j'ai flashé ce kernel, je le règle comment ? »

« Je veux gagner en autonomie, je mets quoi comme gouverneur CPU ? »

« C'est quoi le I/O scheduler ? »

Même si les auteurs de ces kernels essayent souvent de mettre des réglages par défaut convenant au plus grand nombre, il convient de mieux comprendre ce qu'il y a derrière tout ça, afin de pouvoir adapter ses réglages à son utilisation, ses priorités, et tout simplement comprendre ce qu'il se passe.

Voici les modestes objectifs de ce petit tutoriel : définir les termes techniques, expliquer de façon simple le rôle des différents paramètres et leurs affectations possibles, et enfin, pour les plus pointilleux, quelques explications pour aider à se créer ses propres scripts de réglages.

Enfin, à la fin de tutoriel, un questionnaire général sur le téléphone est écrit pour répondre aux questions les plus classiques, je vous invite fortement à le lire !

Table des matières

I. Quelques définitions / traductions

Je vais reprendre quelques mots de vocabulaire ou abréviations pour être sûr que tout le monde puisse suivre la suite sans soucis :

- ✓ **CPU** : processeur
- ✓ **GPU** : carte graphique
- ✓ **Gouverneur CPU** (« governor ») : c'est lui qui définit le comportement du CPU. Grossièrement, c'est lui qui lui dit quand être au repos ou être à la fréquence maximale. Selon son réglage, le processeur montera plus ou moins vite en fréquence, donc cela influe directement sur la consommation.
- ✓ **Ordonnanceur mémoire** (« I/O scheduler ») : c'est ce qui contrôle les accès mémoire, l'ordre d'exécution des demandes. C'est lui qui définit quel processus accède à la mémoire pour lire ou écrire à un moment donné. Donc il impact peu la consommation, par contre il peut avoir un rôle dans la fluidité général d'un système.
- ✓ **UV** : UnderVolt, le fait de réduire la tension de fonctionnement d'un composant sans en réduire les performances, pour gagner en consommation et réduire la chauffe. Ceci peut être réalisé sur plusieurs composants, tels que le CPU, le GPU ou le bus. Les tensions stables dépendent de chaque appareil.
- ✓ **OC/UC** : OverClocking / UnderClocking, le fait de monter (over) ou réduire (under) clocking les fréquences d'un composant afin d'en augmenter la performance. Non traité ici.
- ✓ **Tweaks** : modifications / optimisations
- ✓ **I/O Scheduler** : c'est le travail du kernel qui sera traité ici sous le nom **Ordonnanceur mémoire**.
- ✓ **CPU Governor** : c'est le travail du kernel qui sera traité ici sous le nom **gouverneur CPU**.

II. Gouverneur CPU

Il y en a 18 principaux (en anglais afin de les retrouver facilement dans vos réglages) :

1. Ondemand
2. Ondemandx
3. Conservative
4. Interactive
5. InteractiveX
6. Lulzactive (v1 et v2)
7. Smartass
8. SmartassV2
9. Intellidemand
10. Lazy
11. Lagfree
12. Lionheart
13. LionheartX
14. Brazilianwax
15. SavagedZen
16. Userspace
17. Powersave
18. Performance

1) **Ondemand**

Gouverneur par défaut la plupart du temps. Le but de ce gouverneur est de fournir en priorité de la fluidité, dès qu'une activité CPU est détectée, il passe à la fréquence max. Il considère le temps d'utilisation CPU comme critère de choix. Il passe à la fréquence maximale quand le CPU travail, et réduit progressivement quand la charge diminue. Même si d'apparence cela semble un comportement judicieux, cela ne l'est pas en terme de batterie (et parfois même en terme de performance selon ses réglages) . En effet, il se base sur un échantillonnage à un instant pour choisir la prochaine fréquence à définir, or cet instant ne reflète pas forcement une tendance générale. On peut donc souvent se retrouver à alterner sans arrêt fréquence minimale et fréquence maximale, ce qui évidemment consomme.

2) **OndemandX**

Similaire au Ondemand mais avec une notion de « veille/éveil ». Il est supposé plus économe que le Ondemand, car quand l'écran est éteint, il limite la fréquence max (500MHz). Son support dépend du CPU et de sa capacité à varier rapidement en fréquence. Contrairement aux autres gouverneurs, les Ondemand/OndemandX sont relativement dépendants de l'ordonnanceur de mémoire choisi, ils se marient bien avec le SIO.

3) **Conservative**

Proche du Ondemand mais celui-ci ajuste dynamiquement la fréquence par étape, il ne monte pas brusquement à la fréquence maximale. Il augmente progressivement la fréquence du CPU par palier en fonction de l'utilisation du processeur, et fixe la fréquence processeur minimale quand il est en veille.

Pour les experts : Le paramètre sampling_down_factor est un mulitplicateur negatif du paramètre sampling_rate, pour réduire la fréquence d'échantillonnage de l'utilisation CPU. Ainsi, si le sampling_down_factor est fixé à 2 et le sampling_rate, le gouverneur évaluera l'utilisation CPU toutes les 40 000 microsecondes.

4) **Interactive**

Cette fois c'est l'inverse, il s'agit d'une version plus rapide du Ondemand. Objectif : plus rapide donc moins de consommation. Interactive est conçu pour être sensible à la latence, s'adapter à la charge de travail. Au lieu d'échantillonner à intervalles réguliers comme le Ondemand, il détermine la façon de graduer le CPU en sortant de veille. Ses avantages sont :

1) Une hausse plus cohérente car contrairement aux autres gouverneurs qui basent leur évaluation sur la queue de processus en attente de CPU, Interactive lui se base sur une notion de temps en plus, ce qui est plus représentatif de la charge processeur.

2) Une priorité à la hausse de fréquence CPU, donnant un avantage de performance aux tâches à faire plutôt que de baser sa planification de hausse uniquement une fois qu'un manque de performances se fait ressentir.

Le gouverneur Interactive est une sorte de Ondemand plus intelligent et stable. Au lieu d'échantillonner tous les X millisecondes, ce qui peut parfois causer des manques de puissances entre deux échantillonnages, Interactive va regarder de suite en sortie de veille si le CPU a besoin d'être augmenté. Lorsque le CPU est réveillé, il définit un minuteur, qui doit s'activer dans 1 ou 2 périodes. Si le CPU est très chargé entre le début et la fin du minuteur, alors c'est qu'il doit être rehaussé, et Interactive le fixe à la fréquence maximum.

5) **InteractiveX**

Similaire à la différence Ondemand/ OndemandX : cela ajoute une notion de veille pour gagner en batterie.

6) **Lulzactive**

Nouveau gouverneur trouvé par Tegrak, basé sur Interactive et Smartass (voir après), il est un des favoris actuel. Il y a eu 2 versions :

V1 : Quand la charge de travail est supérieure ou égale à 60% pour une fréquence donnée, le CPU passe à la fréquence supérieure. Quand elle est inférieure à 60%, il passe à celle en dessous. Quand l'écran est éteint, la fréquence est bloquée au minimum.

V2 : A ce comportement, il est rajouté 3 paramètres configurables par l'utilisateur, ce qui permet plus d'adaptation.

Pour les experts : ces variables sont inc_cpu_load, pump_up_step et pump_down_step. Quand la charge est supérieure ou égale à inc_cpu_load, le gouverneur augmente le cpu de pump_up_step paliers, et quand elle est inférieure, il diminue de pump_down_step.

7) **Smartass**

Issu du travail de réécriture du gouverneur Interactive par Erasmux. Son objectif était de maximiser la batterie sans compromis de performances. Cependant, il n'est pas aussi économe que smartassv2 car le minimum quand l'écran est allumé est plus grand que les fréquences pour l'écran éteint. De plus smartass va grimper à la fréquence maximale trop souvent aussi.

8) **SmartassV2**

Un autre favori actuel. Le gouverneur vise une « fréquence idéale » , il va grimper rapidement à celle-ci et ensuite être plus souple autours. Il définit des fréquences idéales differentes seon que l'écran soit allumé ou pas (awake_ideal_freq et sleep_ideal_freq). Quand on éteint l'écran il chute rapidement à sleep_ideal_freq, et quand on le rallume il grimpe vite à awake_ideal_freq (500MHz). Contrairement à smartassv1 , il n'y a ici pas de limite maximale quand l'écran est éteint, pour permettre au CPU d'être utilisé à fond même écran éteint. Ce gouverneur est un bon compromis entre performance et autonomie.

9) **Intellidemand**

Encore un dérivé de Ondemand. Cette fois, c'est le GPU qui sert de référence. Quand le GPU est chargé, cela traduit un usage lourd, donc il agit comme le Ondemand classique et grimpe au maximum. Quand le GPU est au repos, Intellidemand fixe une fréquence maximale (selon votre appareil et kernel) pour économiser la batterie. C'est le mode explorateur. Les décisions de montée ou descentes en fréquence se base sur le % de temps de repos du CPU. Quand l'écran est éteint, la fréquence maximale n'est pas utilisée.

10) **Lazy**

Le but de ce gouverneur, écrit par Ezekeel et basé sur le Ondemand, est d'éliminer les changements constants de fréquences qu'introduit le Ondemand. Pour cela, il définit un temps min_time_state qui fixe le temps minimum pour le CPU à une fréquence avant de changer. Lazy va vérifier le CPU plus souvent mais changer moins souvent de fréquences que Ondemand. De plus Lazy définit screenoff_maxfreq qui, si elle est activée, fixe la fréquence à cette fréquence quand l'écran est éteint.

11) **Lagfree**

Un autre de la famille Ondemand (quel succès ! ^^). La principale différence concerne l' économie de batterie. La fréquence CPU est lentement augmentée et diminuée, contrairement_à l'Ondemand qui saute à 100% souvent. Lagfree ne saute aucun pallier quand il augmente la fréquence (ce qui peut poser problème en cas de besoin soudain de puissance par exemple il peut apparaitre un léger lag à l'ouverture d'une vidéo).

12) **Lionheart**

Il s'agit d'un gouverneur de la famille Conservative, et basé sur les sources de l'update3 de samsung. Les tweaks viennt de Knzo et Morfic et l'idée de Netarchy. Au final, il se rapproche plus du gouverneur Performance, au détriment de la batterie car les changements sont agressifs.

Pour les experts : Pour tester le Lionheart depuis le Conservative, réglez : sampling_rate:10000 or 20000 or 50000 (10000 peut être instable) up_threshold:60 down_threshold:30 freq_step:5 Le Lionheart se marie bien avec l'ordonnanceur Deadline. Quand il s'agit de fluidité, et que l'on néglige la consommation de la batterie, un gouverneur Conservative bien réglé est plus efficace qu'un Ondemand bien réglé.

13) **LionheartX**

basé sur le Lionheart mais possède quelques changements et introduit un mode Suspendu, issus du Smartass.

14) **Brazilianwax**

Ressemble au SmartassV2 mais plus agressif donc plus performant mais moins économe.

15) **SavagedZen**

Aussi issu du SmartassV2 mais mieux équilibré entre performances et batterie que le Brazilianwax

16) **Userspace**

C'est l'utilisateur qui définit les fréquences.

17) **Powersave**

Bloque à la fréquence minimum (inutilisable en écran allumé, et même risqué en écran éteint)

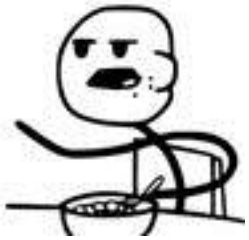
18) **Performance**

Bloque à la fréquence maximale, uniquement pour les benchmark car consomme et chauffe énormément !

On peut donc résumer ces 18 en 4 familles :

- *Famille Ondemand* : travail sur le principe du « Augmente quand une grosse charge arrive » , le temps de calcul CPU est pris en compte : Ondemand, OndemandX, Intellidemand, Lazy, Lagfree.
- *Famille Conservative* : Conservative, Lionheart, LionheartX
- *Famille Interactive* : travail sur le principe du « Choisir la hausse du CPU quand il sort de veille » : Interactive, InteractiveX, Lulzactive, Smartass, SmartassV2, Brazilianwax, SavagedZen.
- *Autres* : Userspace, Powersave, Performance

Questions pour un champion!



Ok bon j'ai assez lu ! En gros lequel est le mieux pour être performant et lequel est le plus économe !

- Bonne question jeune Padawan ! Si le but est la performance, alors un kernel tweaké de type Ondemand ou Conservative, tel que LionheartX ou Brazilianwax, voir même Performance iront très bien mais il ne faudra pas râler sur l'autonomie. Si par contre la batterie à de l'importance, un meilleur équilibre sera trouvé avec Lulzactive (v2) ou SmartassV2.

Bon j'ai fait mon choix, maintenant je l'applique comment ?

- Le plus courant est d'utiliser un script init.d pour le faire. Seulement ce n'est pas toujours supporté ou bien on n'a pas toujours moyen de le faire facilement si on ne connait. C'est pour ça qu'en général, avec le kernel les développeurs fournissent une application pour le paramétrer. Si vous ne la trouvez pas, vous pouvez vous orienter sur des applications connus comme SetCPU ou Voltage Control, qui en plus de ce paramètre, fournissent d'autres réglages pratiques pour faire de l'UV ou de l'OC/UC.

Comment je sais lequel est le mieux pour mon usage ?

- Cela dépend de ce qu'on recherche et de son usage, performance ou autonomie. La plupart du temps on désire un équilibre entre les deux, donc un choix équilibré comme Lulzactive ou Smartassv2 est judicieux. Après si on veut maximiser les performances, on peut se personnaliser un gouverneur à la main pour profiter au mieux de son téléphone mais il faut alors s'armer de temps, de patience et le résultat de sera pas forcément évident.

Raa ! J'ai mis un gouverneur pour écran allumé, un autre quand j'éteins l'écran, et le téléphone plante ! Pourquoi ?

- C'est du à une mauvaise combinaison. Vous avez utilisez deux gouverneurs ayant des profils déjà inclus pour l'écran éteins. C'est ce qu'on appelle le SoD, Sleep Of Death.

J'ai mis un gouverneur qui me plait bien mais parfois j'ai de petits lags, en scroll par exemple, ai –je moyen de les régler sans changer de gouverneur ?

- Oui, ce n'est pas forcément évident mais en rallongeant par exemple le temps durant lequel le gouverneur vérifie le CPU pour réduire la fréquence (quel que soit ce paramètre), cela laisse le processeur plus haut plus longtemps et peut donc éliminer les micros lags.

Bon je veux essayer de paramétrer des valeurs de mon gouverneur, je fais comment ?

- Cela se passe dans un script init.d pour écrire une valeur à : /sys/devices/system/cpu/cpufreq/NOM_DU_GOUVERNEUR/PARAMETRE_A_CHANGER Exemple : echo "20000" /sys/devices/system/cpu/cpufreq/lulzactive/up_sample_time

Je vois que le 100MHz n'est pas utilisé, je vais l'autoriser …

- Doucement ! Il est désactivé et c'est pour de bonnes raisons. Premièrement, selon les tests, il consommé plus que le 200MHz, un comble ! De plus, il s'agit la d'une fréquence de repos, et il vaut mieux alors être à 200MHz, vite finir la tâche, et retourner en veille. Enfin 200Mhz est la fréquence qui a été prévue pour maximiser le rendement pour le Galaxy S II donc autant en profiter !

J'ai déjà mis un gouverneur économe mais je voudrais encore maximiser la batterie, y'a-t-il un moyen ?

- Déjà, premier conseil dans ce cas : ne surtout pas faire d'OC qui consomme beaucoup plus. Au contraire, un UC pourrait être le bon choix. Avec son CPU, le Galaxy S II est déjà bien à l'aise en le bloquant à 800 ou 1000MHz donc vous pouvez tenter de le régler pour aller de 200MHz à 1000MHz et voir le résultat.

Comment je peux rendre mon téléphone encore plus fluide sans trop perdre non plus en autonomie ?

- Définissez un profil avec gestion de veille comme OndemandX ou ConservativeX, et définissez comme plage de fréquences 500 à 1200 en écran allumé et 200 à 500 en écran éteint, c'est un compromis sans excès de consommation ni en veille ni en utilisation et une fluidité absolue.

III. Ordonnanceur mémoire

Commençons par quelques éclaircissements.

Tout d'abord, à quoi sert un ordonnanceur mémoire ?

- Comme je l'ai dit plus haut il répartit les accès à la mémoire entre les processus. Il doit veiller à plusieurs aspects :
  - Minimiser les temps de latences de recherche sur le disque
  - Donner la priorité aux processus interactifs par rapport aux processus de fonds
  - Allouer de la bande passante disque pour les applications en cours
  - Garantir des délais pour certaines demandes

Et quels en sont les objectifs pour bien choisir ?

- Il doit viser 3 optimisations :
  - Equité : permettre à tous d'accéder à la mémoire
  - Performance : Optimiser les lecture/écritures en fonction du disque
  - « Temps réel » : avoir les délais les plus petits possibles pour les requêtes

Ces 3 optimisations entre souvent en conflit c'est pourquoi, comme vous verrez par la suite, aucun n'est parfait, tout dépend des priorités.

On va maintenant passer en revue les différent ordonnanceurs, ceux-ci étant ceux connus dans le monde Linux :

1) **NOOP** : Ajoute chaque requêtes les unes après les autres, premier arrivé premier servi, et implémente la fusion de requêtes. Il peut sembler trivial mais, bien qu'il soit inadapté sur un disque dur, avec des mouvements mécaniques importants à prendre en compte, dans le cas d'un stockage flash où l'ordre importe peu, il est adapté car simple pour le CPU, pas couteux (pas de réorganisation à faire).


- Léger en CPU donc en batterie - Adapté au stockage flash de nos téléphones - Bonne performance sur la base de données	- Réduire le nombre de cycle CPU fait chuter les performances

2) **Deadline** : Son but est de minimiser la latence d'écriture ou l'attente d'une requête. Pour cela il utilise une politique dite de « Round Robin » pour gérer équitablement plusieurs requêtes, ordonnées de façon agressive dans 5 files d'attentes.


- Proche d'un vrai ordonnanceur «temps réel» - Réduit efficacement la latence pour une requête de lecture/écriture - Le meilleur pour gérer les bases de données - Couts facilement calculable - Adapté au mémoire flash	- En cas de grosse charge de travail, on ne peut plus savoir quels processus n'auront pas accès à la mémoire à temps

3) **CFQ** : Il s'agit de l'ordonnanceur le plus commun dans les distributions Linux. Il maintient une file d'attente synchrone par processus, et essaye de distribuer également les accès mémoire entre les files d'attentes des processus. Le « time slice » (temps d'allocution maximum) de chaque file dépend du parent. Il existe une V2 du CFQ apporte quelques amélioration de temps de réponse et corrige quelques problèmes de famines d'accès mémoire.


- Performance de lecture/écriture équilibrée - Facile à personnaliser - Excellent dans les environnements multi-coeurs - Très bon système pour les bases de données	- Certains processus très long en accès mémoires (comme le scanner média au démarrage) peuvent être rallongés du fait du principe d'équité, ils n'ont pas de priorités - Les délais « au pire » sont moins bons qu'avec les autres selon le nombre de tâches du système

4) **BFQ** : Plutôt que de compter sur des temps d'allocutions maximums (le time slice), il compte sur une notion de « budget » par processus, qui sont en fait le nombre de secteurs de disque que le processus accède. Ce budget peut varier selon le comportement du processus.


- Très bon pour les transferts USB - Très bon pour la gestions des videos (HD et enregistrements) car il a un « pire cas » meilleur que les autres - Précis - Rempli 30% de plus de requêtes que le CFQ sur la plupart des charges	- Mauvais en benchmark (ça à la limite …..) - La latence et des lag peuvent apparaitre si un budget élevé est attribué à un processus

5) **SIO** : Ordonnanceur simple qui a pour but de remplir les requêtes mémoire avec une faible latence et un cout système minimum. Il n'y a pas de file par priorités, juste le principe de fusion des requêtes. SIO est un mix entre le NOOP et le Deadline, et il ne réorganise pas les requêtes.


- Simple et fiable - Minimise les « famines » d'accès mémoire	- Les lectures sont plus lentes qu'avec les autres ordonnanceurs sur les mémoires flash

6) **VR** : Contrairement aux autres, il ne traite pas séparément les requêtes synchrones et asynchrones, à la place une limite temporelle est imposée par équité. Il choisit le prochain accès en fonction des distances par rapport à la précédente.


- Très bon en benchamark car dans les situations propices c'est le mieux	- Performances fluctuantes et globalement plutôt moyennes - Le moins fiable

7) **Anticipatory** : Basé sur 2 faits :

a. Les accès disques sont très lents

b. Les écritures arrivent aléatoirement, mais il y a toujours des lectures à faire

En conséquences, il rend prioritaire les lectures sur les écritures et anticipe les opérations de lectures synchrones.

- Les requêtes de lecture ne sont jamais en manque d'accès - Aussi bon que NOOP en performance de lecture sur les mémoires flash	- L'anticipation n'est pas toujours fiable - Réduit les performances d'écriture sur les disques très rapides

Place aux questions !

Ok ok mais lequel est le meilleur ??

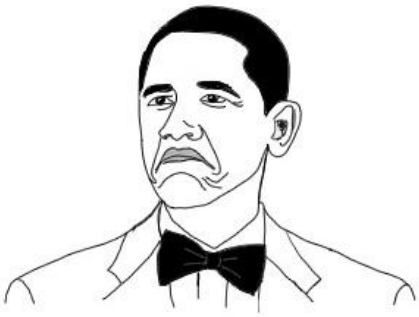
- Il n'y a pas de « meilleur » ordonnanceur. Ça dépend de vos usages et des applications à faire tourner, chacune traitant la mémoire à sa façon. Cependant si on se place pour la téléphonie, donc en terme de batterie, stabilité, fiabilité et faible latence, sur un système flash on peut dégager un classement général de la forme :

  SIO >= NOOP > Deadline > VR > BFQ > CFQ

Et comment je change ça ?

- Encore une fois, comme avant deux possibilités : la plus simple, via des applications comme Volt Control ou dédiées à votre kernel. Sinon, via un script init.d :
  echo "nom_de_l'ordonnanceur" > /sys/block/mmcblk0/queue/scheduler

Voilà nous en avons finis avec les principes généraux permettant de comprendre ce que l'on règle dans un kernel. La suite va s'adresser aux personnes désirant aller plus loin et non plus seulement comprendre mais aussi utiliser ces réglages, via des scripts init.d ! A utiliser avec précaution !



IV. Les scripts init.d

a. Qu'est-ce que c'est ?

Au démarrage, Android boot en 3 étapes de hauts niveaux :

- Le bootloader, qui amorce la suite
- Le kernel qui lance les drivers, prépare le matériel et établit le contact avec lui
- Les programmes utilisateurs sont lancés

Seulement après ces 3 étapes la rom commence vraiment à se lancer. Les scripts dont nous allons parler ici se situent à la 3ème étape, donc avant le lancement de la rom.

La plupart des kernels personnalisés gèrent les scripts init.d. Pour certains, il faut commencer le nom du fichier par « S », pour d'autres il faut juste le placer au bon endroit (/system/etc/init.d).

Ils se lancent selon leur ordre alphabétique (par ordre ASCII pour être exact). On peut utiliser cette propriété si l'ordre est important entre deux scripts.

b. Construction

Première remarque avant de vous faire perdre du temps : **N'UTILISEZ PAS LE BLOC NOTE WINDOWS** ! Celui-ci sera mal lu par votre téléphone car il ajoute des caractères un peu partout, utilisez Notepad++, un éditeur simple, puissant et gratuit sous licence GNU : Notepad ++

Une fois cela fait :

1. En premier ligne, il faut lui donner le langage utilisé, celui-ci devant être lisible par le shell du téléphone. Elle va donc ressembler à ça :
   #!/sbin/busybox sh or #!/system/xbin/busybox sh (le chemin de busybox étant variable selon les rom)
   Ou ça :
   #!/system/bin/sh
2. Ensuite, le script commence, on enchaine les lignes désirées, sans signes particuliers à la fin, chaque ligne pouvant ressembler à quelque chose comme ça :
   echo "200" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
   Si l'on veut écrire des commentaires pour s'y retrouver, ceux –ci sont précédés de #.
3. Quand vous avez fini, vérifiez que vous n'avez pas d'espace ni au début ni à la fin de chaque ligne du script.
4. Sauvegardez le fichier sans extension (sous windows, vous pouvez le mettre en .txt puis, en demandant à afficher les extensions connues, supprimer le « .txt » après).
5. Utilisez un explorateur root (RootExplorer par exemple) pour placer le script dans /etc/init.d, et réglez les permissions ainsi : rwx – r_x – r_x (owner – group – others).
6. Téléchargez script manager depuis le Play Store, et avec lancez le script en tant que root. Cela sert à vérifier que le script se déroule bien du début à la fin. Si il retourne 0 c'est bon, sinon revérifiez le avec Notepad ++ ou Script Manager (surtout les espaces en fin et début de lignes !) .

Et voilà ! Une petit script tout simple pourrait ressembler à ça :

```
#!/sbin/busybox sh

#No of values echoed to freq table and uvmv table should match the no of steps in your kernel
echo "1600 1400 1200 1000 800 500 200 100" > /sys/devices/system/cpu/cpu0/cpufreq/freq_table
echo  "1425 1325 1275 1175 1075 975 950 950"> /sys/devices/system/cpu/cpu0/cpufreq/UV_m_table
echo "200" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
echo "1200" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
echo "ondemandx" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

c. Les options possibles pour les gouverneurs CPU

En fonction du gouverneur choisi, nous avons accès à différents paramètres. Nous allons ici nous intéresser aux principaux que sont Ondemand, Conservative, SmartassV2, Lulzactive et Interative. Les 2 réglages de bases idéalement sont :

- **SamplingTime/Rate**, qui détermine en µS l'intervalle auquel est vérifié le Threshold (voir ci-dessous) afin d'augmenter ou diminuer la fréquence du CPU.
- **Thresholds**, en %. Quand la charge de travail du CPU atteint ce pourcentage, le gouverneur décide qu'il faut monter (ou descendre) la fréquence CPU. La plupart du temps, la montée et la baisse en fréquence sont gérés par des pourcentages distincts (up_threshold pour la montée, down_threshold pour la baisse).

1) **OnDemand**

[PARAMETRES]

- **sampling_rate** qui détermine en µS l'intervalle de mesure pour choisir s'il faut monter ou descendre la fréquence CPU. Une haute valeur signifie que le CPU est moins souvent testé. Pour les faibles fréquences c'est un avantages car ainsi le CPU ne montera pas en permanence par contre en haute fréquence, cela signifie qu'il mettra plus de temps à redescendre.
- **up_threshold** en %, qui détermine à partir de quel % de charge le CPU va monter à la fréquence supérieur. Une valeur élevée signifie moins de répondant alors qu'une faible valeur corrige cela, au prix de plus de batterie.
- **powersave_bias** , par defaut à 0. Mettre une valeur plus élevée va inciter le gouverneur vers le palier de fréquence plus faible. On peut utiliser ce paramètre pour que le CPU ne passe pas trop de temps aux plus hautes fréquences. Une meilleur alternative alors serait alors d'UC pour baisser la fréquence maximale.
- **sampling_down_factor**, qui agit comme un multiplicateur pour évaluer réellement la charge de travail lorsque le CPU est à la fréquence maximale. Par defaut il est à 1 (sauf si le mode OnDemand est modifiée sur votre kernel), et le monter aura pour effet de tester moins souvent la charge CPU quand il est à la fréquence maximale (scaling_max_freq, et uniquement là) car sinon le fait de tester constamment réduit les performances, et cela peut entrainer le CPU à monter / descendre de fréquence constamment, ce qui est encore plus mauvais.

- **down_differential** qui calcul indirectement le « down_threshold », le % limite pour baisser de fréquence. Après avoir fait sampling_down_factor*sampling_rate quand il est à la fréquence maximale à cause de la charge de travail, le gouverneur va à nouveau tester la charge pour calculer la nouvelle fréquence ciblée de façon à ce que la fréquence la plus faible soit choisie et qu'à la fois celle-ci ne soit pas à remonter au prochain échantillonnage. C'est dans ce calcul qu'intervient down_differential, en tant que marge. La fréquence cible est calculée ainsi : max_load_freq/(up_threshold-down_differential). Le résultat est arrondi à la valeur la plus proche de la table de fréquences. max_load_freq est la fréquence théorique maximale pour la charge de travail à 100%, et est en général en dessous de scaling_max_freq.

[TWEAKS]

- **Pour économiser la batterie** 

```
echo "95" /sys/devices/system/cpu/cpufreq/ondemand/up_threshold
echo "120000" > /sys/devices/system/cpu/cpufreq/ondemand/sampling_rate
echo "1" > /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor
echo "5" > /sys/devices/system/cpu/cpufreq/ondemand/down_differential
```

Ce réglage test moins souvent le CPU et monte le CPU moins souvent.

- **Pour les performances** 

```
echo "70" > /sys/devices/system/cpu/cpufreq/ondemand/up_threshold
echo "50000" > /sys/devices/system/cpu/cpufreq/ondemand/sampling_rate
echo "2" > /sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor
echo "15" > /sys/devices/system/cpu/cpufreq/ondemand/down_differential
```

Ce réglage test régulièrement le CPU et le monte assez souvent.

2) Lulzactive V1

[PARAMETRES]

- **down_ampling_time** règle la période d'échantillonnage pour baisser le CPU.
- **up_ampling_time** règle la période d'échantillonnage pour monter le CPU.

[TWEAKS]

La v1 n'autorise que très peu de modifications. Le seule paramètre valant la peine d'être modifié, si vous rencontrer des lags dans le navigateur par exemple, est down_sampling_time. Réglez le de façon proportiennelle à up_sampling_time (celui-ci étant à 24000, cela donne par exemple 2*24000 = 48000 pour down_sampling_time).

3) Lulzactive V2

[PARAMETRES]

- **inc_cpu_load en %.** C'est la charge à partir de laquelle le CPU va monter. Dans la v1 elle était fixée à 60. Si la charge est >= à cette valeur, le CPU monte, sinon il descend.
- **pump_up_step** : c'est le nombre de palier que passe le CPU quand il monte.
- **pump_down_step** : idem mais en descendant.
- **Screen_off_min_step** : Définit les paliers possibles que le gouverneur peut utiliser à la demande quand l'écran est éteint. Ex : si la table de fréquence est ainsi (exemple du Siyah v3) :
  1600 1400 1200 1000 800 500 200 100 (de L0 à L7) et que on le définit à 5, alors le gouverneur pourra utiliser les fréquences de L5 à L7 soit 500, 200 et 100.
- **up_sample_time** idem que v1.
- **down_sample_time** idem que v1.

[TWEAKS]

- **Pour économiser la batterie** 

```
echo "90" > /sys/devices/system/cpu/cpufreq/lulzactive/inc_cpu_load
echo "1" > /sys/devices/system/cpu/cpufreq/lulzactive/pump_up_step
echo "2" > /sys/devices/system/cpu/cpufreq/lulzactive/pump_down_step
echo "50000" > /sys/devices/system/cpu/cpufreq/lulzactive/up_sample_time
echo "40000" > /sys/devices/system/cpu/cpufreq/lulzactive/down_sample_time
echo "5" > /sys/devices/system/cpu/cpufreq/lulzactive/screen_off_min_step
```

Ce réglage permet à Lulzactive de monter graduellement et de redescendre rapidement quand la charge baisse.

- **Pour des performances équilibrées** 

```
echo "90" > /sys/devices/system/cpu/cpufreq/lulzactive/inc_cpu_load
echo "4" > /sys/devices/system/cpu/cpufreq/lulzactive/pump_up_step
echo "1" > /sys/devices/system/cpu/cpufreq/lulzactive/pump_down_step
echo "10000" > /sys/devices/system/cpu/cpufreq/lulzactive/up_sample_time
echo "40000" > /sys/devices/system/cpu/cpufreq/lulzactive/down_sample_time
echo "5" > /sys/devices/system/cpu/cpufreq/lulzactive/screen_off_min_step
```

Ce réglage permet à Lulzactive de tester régulièrement le CPU et le monter de 4 paliers mais seulement quand il passé les 90% de charge. La baisse elle est normale.

Note : Pour le LulzactiveV2 vous pouvez aussi vous servir de l'application du market « lulzactive » de tegrak pour définir ces réglages.

4) **SmartassV2**

[PARAMETRES]

- **awake_ideal_freq** qui est la fréquence « idéale » jusqu'à laquelle le CPU est augmenté rapidement quand le téléphone sort de veille. Après celle-ci, l'augmentation est moins agressive.
- **sleep_ideal_freq** qui est la fréquence jusqu'à laquelle le CPU diminue rapidement quand le l'écran s'éteint. Après celle-ci, la baisse est moins agressive.
- **up_rate_us** qui correspond au temps minimum sur un palier avant que le CPU ne puisse monter plus haut (ignoré quand le gouverneur cherche à atteindre awake_ideal_freq).
- **down_rate_us** qui correspond au temps minimum sur un palier avant que le CPU ne puisse descendre de fréquence (ignoré quand le gouverneur cherche à atteindre sleep_ideal_freq).
- **max_cpu_load** : idem que up_threshold de Ondemand.
- **min_cpu_load** : idem que down_threashold.
- **ramp_down_step** qui représente le pas de baisse de fréquence quand on est en dessous de la fréquence idéale. Mettre 0 désactive ce pas et la baisse se fait alors selon les algorithmes en fonction de la charge. Quand on est au-dessus de la fréquence idéale, on diminue toujours à la fréquence idéale.
- **ramp_up_step** qui représente le pas d'augmentation de fréquence quand on est au-dessus de la fréquence idéale. Mettre 0 désactive ce pas et la hausse se fait alors directement à la fréquence maximale. Quand on est en-dessous de la fréquence idéale, on augmente toujours à la fréquence idéale.
- **sleep_wakeup_freq** qui est la fréquence à appliquer en sortie de veille. Quand sleep_ideal_freq=0 alors elle est sans effet.

[TWEAKS]

- **Pour économiser la batterie** 

```
echo "500000" > /sys/devices/system/cpu/cpufreq/smartass/awake_ideal_freq;
echo "200000" > /sys/devices/system/cpu/cpufreq/smartass/sleep_ideal_freq;
echo "500000" > /sys/devices/system/cpu/cpufreq/smartass/sleep_wakeup_freq
echo "85" > /sys/devices/system/cpu/cpufreq/smartass/max_cpu_load;
echo "70" > /sys/devices/system/cpu/cpufreq/smartass/min_cpu_load;
echo "200000" > /sys/devices/system/cpu/cpufreq/smartass/ramp_up_step;
echo "200000" > /sys/devices/system/cpu/cpufreq/smartass/ramp_down_step;
echo "48000" > /sys/devices/system/cpu/cpufreq/smartass/up_rate_us
echo "49000" > /sys/devices/system/cpu/cpufreq/smartass/down_rate_us
```

- **Pour les performances** 

```
echo "800000" > /sys/devices/system/cpu/cpufreq/smartass/awake_ideal_freq;
echo "200000" > /sys/devices/system/cpu/cpufreq/smartass/sleep_ideal_freq;
echo "800000" > /sys/devices/system/cpu/cpufreq/smartass/sleep_wakeup_freq
echo "75" > /sys/devices/system/cpu/cpufreq/smartass/max_cpu_load;
echo "45" > /sys/devices/system/cpu/cpufreq/smartass/min_cpu_load;
echo "0" > /sys/devices/system/cpu/cpufreq/smartass/ramp_up_step;
echo "0" > /sys/devices/system/cpu/cpufreq/smartass/ramp_down_step;
echo "24000" > /sys/devices/system/cpu/cpufreq/smartass/up_rate_us
echo "99000" > /sys/devices/system/cpu/cpufreq/smartass/down_rate_us
```

5) Conservative

[PARAMETRES]

- **down_threshold, up_threshold, sampling_down_factor, sampling_rate** : voir les gouverneurs precedents, notamment Ondemand.
- **freq_step** définit de combien (en pourcentage de la fréquence maximale du CPU) le gouverneur va augmenter la fréquence quand la charge dépasse le up_threshold.

[TWEAKS]

- **Pour économiser la batterie** 

```
echo "95" > /sys/devices/system/cpu/cpufreq/conservative/up_threshold
echo "120000" > /sys/devices/system/cpu/cpufreq/conservative/sampling_rate
echo "1" > /sys/devices/system/cpu/cpufreq/conservative/sampling_down_factor
echo "40" > /sys/devices/system/cpu/cpufreq/conservative/down_threshold
echo "10" > /sys/devices/system/cpu/cpufreq/conservative/freq_step
```

(On influence sur le freq_step plus faible pour économiser)

- **Pour les performances** 

```
echo "60" > /sys/devices/system/cpu/cpufreq/conservative/up_threshold
echo "40000" > /sys/devices/system/cpu/cpufreq/conservative/sampling rate
echo "5" > /sys/devices/system/cpu/cpufreq/conservative/sampling_down_factor
echo "20" > /sys/devices/system/cpu/cpufreq/conservative/down_threshold
echo "25" > /sys/devices/system/cpu/cpufreq/conservative/freq_step
```

(N'est-ce pas un peu ironique de régler un gouverneur Conservative pour viser des performances de fou ? )

6) Interactive

[PARAMETRES]

- **hispeed_freq** qui est la fréquence à laquelle on va grimper depuis une fréquence faible quand la charge monte (la valeur par défaut est la fréquence maximale).
- **go_hispeed_load** : similaire à up_threshold des autres gouverneurs.
- **min_sample_time** : similaire à down_rate_us de SmartassV2.
- **timer_rate** : le taux d'échantillonnage du timer pour tester s'il faut augmenter la fréquence.

[TWEAKS]

- **Pour économiser la batterie** 

```
echo "95" > /sys/devices/system/cpu/cpufreq/interactive/go_hispeed_load
echo "1000000" > /sys/devices/system/cpu/cpufreq/interactive/hispeed_freq
echo "10000" > /sys/devices/system/cpu/cpufreq/interactive/min_sample_time
echo "40000" > /sys/devices/system/cpu/cpufreq/interactive/timer_rate
```

(On joue que le hispeed_freq pour essayer d'économiser de la battterie)

- **Pour les performances** 

(Dans le cas où votre fréquence maximale a été fixée à 1400MHz ou plus)

```
echo "80" > /sys/devices/system/cpu/cpufreq/interactive/go_hispeed_load
echo "1400000" > /sys/devices/system/cpu/cpufreq/interactive/hispeed_freq
echo "40000" > /sys/devices/system/cpu/cpufreq/interactive/min_sample_time
echo "20000" > /sys/devices/system/cpu/cpufreq/interactive/timer_rate
```

V. Un mot sur les modules

Le kernel Android est un kernel Linux. Comme celui-ci il se présente sous une forme « modulaire ». On peut charger ou décharger des fonctionnalités du kernel, au choix et ce de façon simple par une ligne de commande dans le init.d. Cela permet de réduire l'empreinte mémoire du kernel en enlevant les fonctions inutiles.

La ligne à insérer pour activer un module est :

insmod /lib/modules/nom_du_module.ko

Et pour en décharger un :

rmmod " nom_du_module "

Attention : Tous les modules ne sont pas forcément disponibles selon les kernels.

La liste des modules :

*1) **bthid.ko** - BlueTooth Human Interface Device*

Signifie: Bluetooth

*2) **cifs.ko** - Common Internet File System*

Signifie: Partage internet (notamment avec Windows, il remplace SMB)

*3) **fuse.ko** - File System in Userspace*

Signifie: Système de fichiers (sert pour gérer le ntfs notamment)

*4) **cuse.ko** - Character Devices in User Space*

Signifie: Audio Proxying (sert uniquement pour le OSS)

*5**) dhd.ko** - Dongle Host Driver*

Signifie: Wifi (notamment le Tethering)

*6) **ftdi_sio.ko** - Future Technology Devices International - Serial I/O*

Signifie: USB Serial Devices (pour les appareils utilisant le FTDI)

*7) **usbserial.ko** - USB Serial*

Signifie: Modem série USB (souvent couple avec ftdi_sio)

*8) **gspca_main.ko** - GSPCA Main Driver*

Signifie: Webcams utilisant le protocole GSOCA

*9) **hfs.ko** - Hierarchical File System*

Signifie: Système de fichiers Mac (pour utiliser le HFS)

10) ***hfsplus.ko*** *- Hierarchical File System Plus*

Signifie: Système de fichiers Mac (pour le HFS+ cette fois, utilize sur les iPod notamment)

11) ***j4fs.ko*** *- Jong Jang Jintae Jongmin File System*

Signifie: Système de fichiers J4FS (base sur LFS)

12) ***ld9040_voodoo.ko*** *- LD9040 AMOLED Driver*

Signifie: Voodoo Color(correction de couleurs d'écran par Supercurio)

13) ***scsi_wait_scan.ko*** *- Small Computer System Interface Wait Scan*

Signifie: Attente pendant le boot(des tâches asynchrones)-Ceci peut rallonger le temps de démarrage.

14) ***Si4709_driver.ko*** *- Si4709 FM Radio Driver*

Signifie: Radio FM

15) ***vibrator.ko –*** *Vabration sur le touché tactile*

Signifie: Retour tactile

16) ***logger.ko*** *- Logger Pour Android*

Signifie: Logging/Debugging

17) ***mc1n2_voodoo.ko*** *- mc1n2 Voodoo Sound Driver*

Signifie: Voodoo Sound(améliore et permet de régler l'audio, par Supercurio)

18-25) ***cpufreq_ brazilianwax.ko, cpufreq_ interactive.ko, cpufreq_ interactivex.ko, cpufreq_ lazy.ko, ondemandX.ko, cpufreq_ powersave.ko, cpufreq_ savagedzen.ko, cpufreq_ userspace.ko***

A insérer pour utiliser un de ces gouverneurs CPU.

Les modules essentiels sont déjà chargés par défaut donc ces modules sont à activer ou désactiver avec précautions.

VI. Ouverture sur l'architecture du Galaxy S II et les optimisations possibles

Cette partie sera traitée sous forme de questions/réponses étant donné le caractère vaste et parfois décousu de cette problématique d'optimisation …

Alors c'est parti ! 

Quel est le matériel de base du SGS2 qui fait que l'on aime autant ce téléphone ?

- La version internationale I9100 se base sur une architecture de choc :
  - Processeur : ARC Cortex A9 MPCore processeur sur un SoC Exynos 4210 (System On Chip : système complet intégré sur une puce), gravé en 45 nm. La bande passante mémoire théorique est de 6,4 Go/s pour les tâches lourdes comme l'encodage de vidéos HD.
  - GPU : ARM Mali-400
  - Mémoire : LPDDR2 (ou DDR3)

Quelle est la signification de la fréquence du bus ?

- Pour simplifier, la vitesse du bus est la vitesse qui détermine à la rapidité du transfert des données vers et depuis la mémoire. Le débit mémoire est directement proportionnel à la fréquence du bus. Pour les tâches nécessitant peu de travail du processeur sur chaque donnée, avoir un bus avec une fréquence faible signifie que le CPU attendra longtemps que les données arrivent car il traitera rapidement la donnée arrivant et attendra ensuite la prochaine. Sur le SoC Exynos 4210, le bus est géré par une architecture AMBA (Advanced Micro-controller Bus Architecture).

Qu'est-ce qui modifie la fréquence du bus ? Comment on règle ça ? quels en sont les avantages ?

- Par défaut, le comportement du bus mémoire est régit par une augmentation dynamique, où la fréquence est calculée à la volée pour chaque palier de fréquence CPU, selon les nécessités de l'application. Nous pouvons modifier ce comportement, en fixant des vitesses statiques pour chaque palier de fréquence CPU. Trois valeurs sont possibles : (0) 400MHz , (1) 266 MHz , (2) 133 MHz.
  Cela donnerait par exemple une modification ainsi :
  echo "0 0 0 1 1 1 2 2" > /sys/devices/system/cpu/cpu0/cpufreq/busfreq_static
  echo "enabled" > /sys/devices/system/cpu/cpu0/cpufreq/busfreq_static
  Alors pour les 3 paliers CPU les plus hauts, le bus est à 400MHz, ensuite à 266 pour les 3 suivants et à 133 pour les 2 derniers.
  Les avantages de ce changement pour un mode statique sont : Une économie de batterie en fixant une vitesse faible de bus sur les faibles fréquences CPU, et une prévention de surchauffe.

Parfois j'ai des lags durant la lecture de vidéos HD ou sur des jeux 3D en utilisant ces fréquences de bus statiques, pourquoi ?

- La lecture HD ainsi que certaines jeux lourds demandent au moins un bus à 400 ou 266 MHz quelque soit la fréquence CPU. Pour régler ça, on peut soit tout simplement désactiver le comportement statique (*echo "disabled" > /sys/devices/system/cpu/cpu0/cpufreq/busfreq_static*) soit partir du fait que en fonctionnement le CPU sera au moins à 500MHz et donc régler une fréquence de bus supérieure dès le palier de 500MHz.

Le Galaxy S II a deux cœurs, comment sont-ils utilisés ? Sont-ils actifs en permanence ?

- Par défaut, le mode est sur « Dynamic Hot Plug Mode », c'est-à-dire que selon la charge, le deuxième cœur est activé ou non. Si elle peut être gérée par un seul cœur, le second est coupé. Ce comportement peut être géré par l'application « Tegrak Second Core » si votre kernel le permet. Il y a 3 modes :
  - Dynamic Hot Plug Mode : Mode par défaut et c'est celui recommandé.
  - Single Core Mode : Quoi qu'il arrive, un seul cœur est actif, cela augmente la batterie mais réduit les performances quand il y a des tâches lourdes ou multiples (jeux, vidéos).
  - Dual Core Mode : (à réserver aux benchmark et jeux 3D très lourds) Ce mode active les 2 cœurs en permanence, pour gagner un peu en réactivité mais cela fait chuter l'autonomie.

  Le mode Dynamic Hot Plug est suffisamment rapide dans son activation du 2ème cœur pour permettre une bonne fluidité. De plus, seul le premier cœur (cpu0) peut entrer en état de vieille donc avoir les 2 activés entraine une surconsommation même en vieille.

Ok j'utilise bien le mode Dynamic Hot Plug, mais y'a-t-il moyen de le paramétrer ?

- Oui, on peut régler les paliers d'activation et coupure du 2ème cœur pour l'écran allumé et éteint.
  Exemple (dans un init.d) :
  echo "70" > /sys/module/pm_hotplug/parameters/loadh ***(2ème actif dès 70% de charge, écran allumé)***
  echo "25" > /sys/module/pm_hotplug/parameters/loadl ***(2ème inactif en dessous de 25%, écran allumé)***
  echo "90" > /sys/module/pm_hotplug/parameters/loadh_scroff ***(2ème actif dès 90%, écran éteint)***
  echo "35" > /sys/module/pm_hotplug/parameters/loadl_scroff ***(2ème inactive sous 35%, écran éteint)***

  Ce type de réglage peut permettre d'éviter que le 2ème cœur ne s'active quand l'écran est éteint et que l'on écoute de la musique par exemple.

Pour les gouverneurs CPU, on a pu régler la fréquence de mesure, peut-on faire pareil ici ?

- Théoriquement oui. Cependant la plupart des kernels ne laissent pas cette possibilité et le fixe. Si vous y avez accès vous pouvez le régler en sachant que un taux d'échantillonnage plus élevé peut permettre d'activer moins longtemps le 2ème cœur et donc gagner un petit peu de batterie.

Quels sont ces modes : IDLE, LPA et AFTR ?

- Entre avoir l'écran éteint et un état de «sommeil profond », il y a différents modes de veille du téléphone qui sont gérés par le driver CPU. Il y a le IDLE, c'est-à-dire la vieille normale, il y a le LPA, c'est-à-dire la vieille profonde et enfin le AFTR (Arm off Top Running). Le passage en vieille par le CPU est implémenté pour économiser de la batterie.
  - Dans l'état IDLE, le CPU n'est plus cadencé mais aucun composant n'est coupé.
  - Dans l'état LPA, après IDLE donc, le CPU n'est toujours pas cadencé mais en plus certaines parties matérielles sont coupées. Cette veille profonde économise réellement de l'énergie et il n'y a pas besoin de fixer manuellement de limite de fréquence pour l'écran éteint, une bonne pratique est plutôt de se servir de gouverneur CPU gérant un profil d'écran éteint. La veille profonde n'est plus utilisée quand le téléphone passe en sommeil profond ou alors quand le téléphone se réactive. Quand le téléphone entre en veille profond, le CPU est fixé à la fréquence SLEEP_FREQ et n'est plus cadencé en plus ou en moins jusqu'à ce qu'il quitte cet état.
  - Le mode AFTR est un patch du IDLE permettant de gérer le mode Top = Off pour la veille profonde. Le cache L2 garde ses données dans ce mode.

On peut combiner les modes IDLE ou AFTR avec le LPA (mais il n'est pas possible de combiner IDLE et AFTR, ceux-ci se trouvant au même « niveau »). Les valeurs possibles sont donc :

0 :IDLE

1 : AFTR

2 : IDLE+LPA

3 : AFTR+LPA

Quel est le mode recommandé parmi ces 4 et comment le choisit-on ?

- Le mode recommandé est le mode 3, AFTR+LPA. Pour le fixer, soit une application permet de régler votre kernel, soit vous passez par un script init.d :
  echo "3" > /sys/module/cpuidle/parameters/enable_mask

Qu'est ce que "sched_mc" ?

- L'équipe Linaro ont inventé le sched_mc, ou encore schedule multi core, pour rendre l'ordonnancement de tâches adapté aux multi-cœurs, c'est-à-dire ici en utilisant les 2 cœurs judicieusement pour économiser de la batterie et avoir des performances équilibrées. Même si sched_mc est une alternative avec la coupure cpu vu avec le mode Dynamic Hot Plug, les deux peuvent être combinés.
  Les valeurs possibles sont :
  0 : Pas d'économie d'énergie en fonction de la charge, valeur par défaut sur le Exynos 4210.
  1 : Remplir un cœur en premier pour les tâches longues. Sur un processeur unique muni de 2 cœurs (comme le SGS II) le multi-threading n'est pas pris en compte donc c'est plus ou moins équivalent à la gestion dynamique du 2ème cœur.
  2 : En plus privilégier l'attribution des tâches pour remplir la 1ère moitié des CPU disponibles, et laisser l'autre partie en veille, pour économiser de la batterie. Ici cela correspond à saturer CPU0 en premier avant d'activer CPU1.

Quelle est la valeur recommandée pour le sched_mc ?

- Si vous trouvez des avantages à utiliser le sched_mc, alors vous pouvez fixer sched_mc à 1 pour un possible gain de batterie. Mais puisque ce mode est plus ou moins redondant avec la gestion dynamique du 2ème cœur vu avant, il se peut qu'il n'y ait ici aucun avantage.

  Si vous voulez privilégiez les performances, mettez-lui une valeur à 2. Mais gardez en mémoire que seul le CPU0 peut entrer en veille profonde, donc quelques soit les performances, fixer la valeur 2 consommera plus de batterie car le CPU0 ne sera pas en veille profonde.

  Pour laisser la main à la gestion dynamique du 2ème cœur, mettez 0.

  echo "VOTRE_VALEUR_CHOISIE" /sys/devices/system/cpu/sched_mc_power_savings

Qu'est-ce que le « MALI aggressive policy » sur le GPU ?

- C'est simplement le fait de diminuer le up_threshold, palier pour le lequel le GPU va augmenter, pour éviter que le GPU ne saute à la deuxième fréquence trop souvent. Cela prend son sens quand en parallèle le palier inférieur est réduit de fréquence.

Qu'est-ce que le « tree rcu », le « fast nohz » et le « jrcu » ?

- Read-Copy Update (RCU) est un mécanisme de synchronisation ajouté au kernel Linux. Il améliore la souplesse lors des montées en charge en allouant des lecteurs pour s'exécuter simultanément avec des processus d'écriture.
  Le Tree RCU est une nouvelle implémentation du RCU original pour atteindre encore une meilleure adaptation à la charge puisque le nombre de CPU augmente.
  Fast NoHz est une version optimisée du Tree RCU. Beaucoup de nouveaux kernels utilisent la conception dites « Tickless NoHz ». Ce RCU est fait pour travailler de pair avec le nouveau kernel NoHz .

  JRCU est un mécanisme pour exécuter les opérations de routine sur un seul CPU, soulageant les autres CPU de cette tâche répétitive. C'est un mécanisme important pour les systèmes en temps réel mais pour le SGS 2, cela entre en conflit avec le gestion dynamique des 2 cœurs, c'est pourquoi le Tree RCu est utilisé (avec ou sans le Fast NoHz : CONFIG_RCU_FAST_NO_HZ) dans nos kernels.

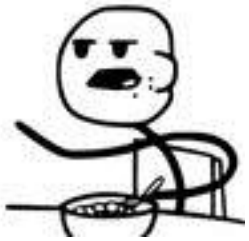
Qu'est-ce que le SLAB, SLUB, SLQB ?

- Ce sont 3 mécanismes d'allocation mémoire.
  - SLAB est un gestionnaire prévu pour l'allocation efficace d'objets du kernel, qui a la propriété d'éliminer la fragmentation dut aux allocations/désallocations. Il retient le type d'un objet de données en mémoire pour réutiliser cette espace avec le même type plus tard.
  - SLUB promet de meilleurs performances et plus de souplesse en laissant tomber la plupart des files d'attentes du SLAB et en en simplifiant la structure globale. SLUB permet d'aligner les objets et de vider les caches plus simplement que le SLAB.
  - SLQB est un SLAB avec une file d'attente. C'est un SLAB qui se concentre sur la gestion par CPU. Il est adapté aux systèmes possédant un petit nombre de CPU, et est conçu pour être simple, alors que su un système avec un nombre de CPU important, SLUB reste le plus pertinent, même si SLAB a l'avantage de la simplicité.

Puis-je changer la synchronisation RCU et l'allocation mémoire ?

- Non, ces choix se font au moment de la compilation du kernel.

Encore quelques questions plus générales mais UTILES ! …



Qu'est-ce que le « Top-off current » ?

- C'est quelque chose qui intervient dans un cycle de charge batterie. Un cycle se compose de deux phases :
  - Première étape : On fournit un courant constant jusqu'à ce que la batterie atteigne son voltage nominal, constant, situé vers 4,1V.
  - Deuxième étape : Une fois ce voltage atteint, une tension constante est appliquée jusqu'à ce que le courant de charge passe sous le « top-off current ». Par défaut celui-ci est à 200mA. Si on le diminue, la batterie sera rempli un peu plus (mais au détriment de sa durée de vie, il faut éviter de la recharger réellement à 100% trop souvent, pas plus que une fois par mois environ).

Parfois je perds énormément de batterie d'un coup lors d'un reboot ou d'un flash. J'eteints avec 50% et en rallumant je suis à 20% ! Que puis-je faire ?

- En fait votre batterie n'a pas réellement perdu ce %. C'est l'indicateur batterie qui donne des valeurs excentriques. Au démarrage, une grosse charge de travail est demandé, donc le voltage de la batterie peut varier plus que d'habitude et cela induit l'indicateur en erreur.
  Vous pouvez essayer de rentrer ce code (pour le I9100) :
  echo "1" > /sys/devices/platform/i2c-gpio.9/i2c-9/9-0036/power_supply/fuelgauge/fg_reset_soc
  Cela va réinitialiser la jauge batterie, et quelques heures après le % devrait être bon.

Qu'est ce qui consomme le plus dans le téléphone ?

- Sans conteste il s'agit de l'écran ! Surtout pour les couleurs claires et une forte luminosité. L'écran pompe 370mW en moyenne, et jusqu'à 960 mW avec la luminosité à 100% et un ecran blanc ! Evitez donc les fonds d'écrans clairs et réduisez la luminosité ☺

Pour le reste des consommations cela donne :

- ✓ Ecran AMOLED: Moyenne - **370mW**. Ecran blanc, 1% luminosité - **450mW** Ecran blanc, 100% luminosité - **960mW**. Chaque % de luminosité augmente la consommation d'environ 5.2mW.
- ✓ Boutons allumés - **40mW**
- ✓ Flash LED - **1.3W**
- ✓ Camera - **700mW**
- ✓ Bluetooth et GPS - 110 to **180mW**
- ✓ Passage de 2G à 3G - **800mW** pour 8 secondes
- ✓ CPU 1.4 Ghz pleine charge, 100% de luminosité - **4W+**
- ✓ CPU 1.4 Ghz en moyenne - **3.2W**
- ✓ CPU 1.6 Ghz à pleine charge - **5.9W**
- ✓ BLN (Boutons allumés pour les notifications) - **200mW** (8mW en veille sans BLN).
- ✓ Téléchargement Wifi - **1.51W**
- ✓ Téléchargement 2G - **1.598W**
- ✓ Envoi 2G - **853mW**
- ✓ Téléchargement 3G - **1.603W**
- ✓ Envoi 3G - **2.136W (!)**

Parfois le téléphone signale une batterie faible puis s'éteint, alors qu'en le rallumant il reste 20 ou 30%, pourquoi ?

- ➢ Dans des cas extrêmes de charge, comme le passage rapide à une grosse charge avec un CPU réglé à 1.6GHz, le voltage de la batterie peut varier énormément et passer sous les 3.3V ce qui est (à tort) interprété par le téléphone comme une batterie vide.

C'est quoi le « bug » du voltage de 500MHz ?

- ➢ Ce n'est pas un bug en effet, il s'agit simplement d'une sécurité. Quand la fréquence passe à 500MHz depuis une fréquence inférieure, le voltage utilisé est celui du palier à 800MHz, alors que lorsqu'il passe à 500 MHz depuis un palier supérieur, il utilise bien le voltage du palier à 500. Maintenant vous le savez, faites vos tables d'UV en conséquence. ☺

Et voilà !!! Ceci est la fin de ce tutoriel. J'espère qu'il vous aura servi, pour toutes vos remarques ou les éventuelles corrections à effectuer, merci de laisser vos messages sur le forum FrAndroid dans le sujet dédié ☺

Bonne continuation et bon courage dans le chemin de l'optimisation …