UEFI (Unified Extensible Firmware Interface)

Le BIOS nous vient d'IBM et des années 1980. Selon les responsables de l'UEFI, Big Blue ne pensait pas que son Basic Input Output System dépasserait les 250 000 machines. C'est clairement chose faite, mais 30 ans après ses limites technologiques l'ont finalement rattrapé.

La réponse à l'anachronisme du BIOS commence dans les laboratoires d'Intel qui se retrouva face à une impasse lors de la conception de son architecture Itanium (IA64). Le fondeur a renoncé à toute rétrocompatibilité avec les architectures 32 bits, ce qui signifiait que le BIOS, avec son bloc de 512 octets destiné à l'amorçage, était tout simplement trop étriqué et incompatible avec la nouvelle puce.

Intel a donc apporté sa réponse en publiant l'Intel Boot Initiative, installé sur les machines HP-Itanium dans le milieu des années 90. Il a ensuite renommé son projet EFI (Extensible Firmware Interface) en 2000 avec la sortie de l'EFI 1.02 qui fut retiré quasi immédiatement pour des raisons juridiques. Deux ans plus tard, la firme revint sur le devant de la scène avec la version 1.10, compatible avec les processeurs x86 32 bits. C'est le premier pas de l'EFI vers une autre architecture que l'IA64.

Le fondeur se retrouve néanmoins devant un problème : il n'est pas le seul au monde à fabriquer des processeurs et cartes mères et les limitations qu'il a rencontré avec les BIOS existent aussi pour ses concurrents. Il court donc le grand risque de voir fleurir des solutions annexes à son EFI et incompatibles avec son système.

C'est pour éviter le chaos qu'en 2005, il a décidé d'ouvrir son projet en créant le Forum UEFI (Unified Extensible Firmware Interface), un consortium en charge de définir les standards, développer les mises à jour et promouvoir son installation. Regroupant AMD, American Megatrends (AMI), Apple, Dell, HP, IBM, Insyde Software, Intel, Lenovo, Microsoft, et Phoenix Technologies, ce consortium a marqué les premiers pas de l'UEFI 1.0 qui était en fait la version 1.10 de l'EFI. Cette étape marque la fin du monopole d'Intel sur son projet. Il savait que la démocratisation de sa technologie devait passer par une organisation regroupant aussi ses concurrents.

Depuis 2008, l'UEFI 2.0 a apporté la gestion des architectures x86-64 AMD et Intel qui ont la grande particularité d'être rétrocompatible x86-32 bits contrairement à l'IA64. On notera aussi qu'ARM a rejoint les rangs de l'UEFI. Nous en sommes aujourd'hui à la version 2.3 qui a été approuvée en mai 2009.

Depuis 2005, force est de constater que la révolution UEFI tant annoncée n'a pas eu lieu.

Selon le consortium, l'UEFI est un ensemble de spécifications détaillant une interface qui facilite la prise de contrôle de la machine allumée par le système d'exploitation. Il reprend donc le flambeau du BIOS moderne sans réellement apporter de nouveautés.

Réduction des fonctions du BIOS

En effet, du temps de MS-DOS et des OS 16 bits, le système d'exploitation appelait le BIOS au lieu d'accéder aux composants directement. La séparation du matériel et du logiciel était d'ailleurs un des esprits ayant animé la création de ce système. Les OS 32 bits ont néanmoins aboli cela et utilisent des pilotes qui leur permettent de communiquer directement avec les diverses puces et cartes. Le BIOS n'est donc plus qu'un système d'initialisation qui dispose de quelques fonctionnalités de gestion de la consommation (ACPI) et d'initialisation vidéo. Il se limite à lancer le POST (Power-on self-test) qui s'assure de la présence et du bon fonctionnement des composants et périphériques importants et attribue les ressources nécessaires. Il permet enfin de modifier certains paramètres de la carte mère comme la fréquence du bus ou le coefficient multiplicateur (à condition qu'il soit débloqué sur le CPU), etc.

La définition officielle de l'UEFI montre clairement que les choses n'ont pas beaucoup évoluées. Pourtant

l’UEFI apporte évidemment des nouveautés. Il est modulaire, permettant aux constructeurs d’ajouter leurs propres pilotes, tandis que le BIOS est figé. Le premier est plus facilement portable, car écrit en C, contrairement au dernier qui repose toujours sur l'assembleur. Enfin, la gestion de bytecode par l’EFI signifie que les pilotes peuvent être compilés pour plusieurs architectures CPU à la fois. Néanmoins, le but avoué de cette nouvelle interface est simplement le transfert de la machine à l’OS qui s’occupe alors de tout.

De la grande difficulté à composer sans le BIOS

Cette équivalence de fonction pousse d’ailleurs le Forum UEFI à expliquer sur son site que son interface ne remplace pas complètement le BIOS. Une des grandes idées reçues sur l’EFI consiste à penser que sa présence extermine l’ancienne interface. Or dans de nombreux cas, le système UEFI intègre un mode 16-bit similaire à un BIOS classique pour rester compatible avec les systèmes d’exploitation ne le prenant pas en charge. De plus, le POST n’est pas défini par les spécifications de l’UEFI et les fabricants n’hésitent pas faire appel à un BIOS rudimentaire pour cette fonction avant de passer à l’UEFI, même si Intel le déconseille fortement. Enfin, il est frappant de constater que les premières cartes UEFI avaient des interfaces identiques aux BIOS classiques et certaines en avaient même les limitations.

Les cartes mères Sandy Bridge seraient les premières à retrouver une interface UEFI en version 2.1, principalement pour dépasser la fameuse limite des 2 Tio.

Le grand avantage de l’EFI est qu’il ne demande pas une architecture processeur spécifique, contrairement au BIOS qui demande une interface 16 bits real mode, que tous les CPU x86 prennent en charge.

L’UEFI System Tables

L’UEFI repose sur une table des systèmes UEFI (UEFI Systems Table) qui est la structure fondamentale de l’interface. Tous les exécutables (que l’on appelle aussi image UEFI) disposent d’un pointeur vers cette table. Cette adresse est le point d’entrée du logiciel qui lui permettra d’accéder aux services UEFI qui sont répartis en deux catégories principales : les services du moteur d’exécution (runtime services) et les services d’amorçage (boot services). Une image accède aux services définis par le Forum ou le développeur qui dispose d’une certaine marge de manoeuvre, en définissant la section de la table qu'il souhaite accéder.

Le tableau regroupant les services de démarrage qu'il est possible d'appeler, comprend aussi les protocoles UEFI qui sont, entre autres, des pointeurs de fonction. Très commun en C, ils permettent de simplifier le code en offrant un moyen de plus facilement invoquer une fonction. En l'espèce, le Forum définit un certain nombre de protocoles, mais les développeurs peuvent accroître la liste des fonctionnalités disponibles en définissant les leurs. Les protocoles sont gérés par une base de données et sont accessibles à l’aide de handle, un pointeur intelligent ou plus précisément une référence qui va permettre de retrouver le protocole dans la base. Cette dernière est importante, car elle regroupe aussi les handle attribués à tous les exécutables et aux composants, ces derniers ayant tous un système d’adressage spécifique qui permet de les identifier.

Les diverses images UEFI

Les programmes ou pilotes UEFI ont un entête définissant le type de processeur (IA64 ou x86-64 ou UEFI Byte Code pour une application neutre) et le type du programme, par exemple une application UEFI, un pilote d’amorçage ou un pilote runtime.

Les deux premiers types se lancent avant que l’OS soit appelé. Ils permettent d’établir des connexions réseau, faire tourner toute sorte de programmes, lancer une interface graphique, etc. Dans le cas des applications UEFI, la mémoire allouée est libérée une fois que le programme s’éteint. Dans celui des pilotes d’amorçage, le programme qui va charger l’OS (le bootloader) récupère la mémoire en lançant la commande « ExitBootServices() ». Une chose est sûre, ces deux programmes ne persistent pas une fois l’OS lancé.

Ce n’est par contre pas le cas des pilotes du moteur d’exécution qui maintiennent leur état et leurs allocations mémoires au-delà du chargement du système d’exploitation. Concrètement, ces pilotes sont désignés en mémoire comme « EfiRuntimeServicesCode » et la structure des données est réservée sous « EfiRuntimeServicesData ». Un OS compatible UEFI peut invoquer ces pilotes, même après avoir exécuté la fonction ExitBootServices(). Comme nous le verrons plus tard, cette particularité permet la création d’hyperviseurs UEFI.

Intel ouvre une partie du code source

Avec la création de l’UEFI, Intel a publié une partie de ce code sous licence open source et l’a nommé TianoCore ou EFI Development Kit (EDK). Intel le désigne comme une espèce de noyau de l’EFI, même s'il s’agit de codes standards permettant l’utilisation de diverses versions de l’UEFI. Le TianaCore ne contient aucune référence à des composants spécifiques ou à des phases d’amorçage. Comme Intel le précise sur son site, écrire une interface UEFI à partir du TianaCore est aussi complexe qu’écrire un BIOS à partir d’une feuille blanche.

Le reste de Tiano est connu sous le nom de Framework et est publié sous licence BSD. Il est généralement le fruit des développements des éditeurs de BIOS. Par exemple, Pheonix Technologies gère l’UEFI 2.0 dans son Framework SecureCore. AMI utilise son Framework Aptio, récemment mis à jour pour prendre en charge les processeurs Intel Atom E6xx. C’est d’ailleurs le Framework utilisé pour les interfaces UEFI de MSI et Asus. On note que le standard UEFI permet aux pilotes d’être facilement portés d’un framework à l’autre et toutes les versions de l’interface sont rétrocompatibles.

Outils à la disposition du public

Le développement d’applications UEFI requiert le téléchargement de l’EFI Developement Kit. Il contient les codes open source ainsi que des exemples et des binaires pour un Shell EFI qui permet la création d’un environnement assurant le lancement d’applications et commandes. Il est aussi conseillé de télécharger l’EFI Toolkit. Même s’il n’est pas nécessaire, il offre des en-têtes de C intéressantes. Enfin, il vous faudra un compilateur C 64 bits. Windows Visual C++ est disponible dans le Windows Driver Development Kit.

La documentation portant sur l’interface est disponible gratuitement sur le site du forum et des informations sur les étapes d’amorçage d’un système UEFI sont disponibles dans le Platform Initialization Specification.

La nécessité d’ouverture

Si les technologies sont fermées et la propriété du Forum, les spécifications sont accessibles gratuitement par les acteurs intéressés qui ont une marge de manoeuvre relativement large, le but étant de favoriser sa propagation auprès des éditeurs de systèmes d’exploitation, de BIOS et les fabricants de cartes mères et cartes filles.

En effet, l’UEFI n’est pas seulement l’apanage de Microsoft, Apple et les fabricants de cartes mères. Pour être réellement efficace, il faut que tout l’écosystème soit compatible avec les spécifications UEFI. C’est pour cela que l’on voit apparaître des serveurs lames, des stations de travail, mais aussi des imprimantes et scanners compatibles avec ce standard.

La gestion de l’EFI par les composants n’est qu’une partie de l’équation, les systèmes d’exploitation devant aussi prendre en charge l’interface. Vu son origine, il est facile de comprendre que tous les OS pour Itanium sont compatibles avec cette technologie. Par contre, la situation des systèmes grand public est plus complexe.

Windows

En ce qui concerne les systèmes d’exploitation de Redmond, la gestion de l’UEFI commence avec

Windows XP en version 64 bits seulement. C'est d'ailleurs logique puisque cette version de Windows fut conçue pour tourner sur des processeurs IA64. Les versions 64 bits des OS grand public de Redmond (XP 64 bits pour CPU x86, Vista SP1 64 bits et 7 64 bits) sont aussi compatibles. Concrètement, cela signifie qu'il faut un processeur 64 bits (AMD64 ou EM64T) et au minimum une interface UEFI 2.x, les versions antérieures ne prenant pas en charge ces architectures CPU.

Malheureusement, dans les faits, même la réunion de tous ces critères ne garantit pas une installation sans problème. En effet, les premières cartes mères Intel UEFI 2.x refusaient d'installer Vista SP1 64 bits en raison de l'absence d'un module ACPI dans le Framework.

Comme nous l'avons précisé entre parenthèses, l'UEFI est aussi géré par Windows Server 2003 x64 et Windows XP Professional x64, mais ils ne peuvent pas démarrer sur une partition GPT. Pour cela, il faudra utiliser Server 2008 x64 et les versions 64 bits de Vista et 7.

Mac OS X et Linux

De leur côté, les ordinateurs à la pomme ont apporté la gestion de cette interface avec les premiers Mac Intel. Ils embarquaient alors l'EFI 1.10, ce qui signifiait que les machines, qui demandaient Mac OS X 10.4 minimum, étaient limitées à la version 32 bits de l'EFI, même sur les ordinateurs possédant des CPU Intel compatibles EM64T. Cela voulait d'ailleurs dire que l'installation de Vista SP1 64 bits sur un Mac ne permettait pas l'exploitation de l'EFI, l'OS de Microsoft n'étant compatible qu'avec les versions 2.0 et plus.

À notre connaissance, et selon cette page du site d'Apple, les derniers MacBook Pro sortis à la mi-2010 ne prennent en charge que l'EFI 1.9. Néanmoins, la numérotation de Cupertino est particulièrement confuse puisque nous savons que les Mac Pro de 2008 et les MacBook et les iMac sortis de 2009 gèrent la version 64 bits de l'UEFI, à condition d'utiliser Snow Leopard (Mac OS X 10.6), ce qui signifie qu'ils sont compatibles avec la norme 2.x du Forum.

Elilo et grub sont les boot loader Linux pour systèmes UEFI. Ils sont compatibles avec les processeurs x86 32 bits et 64 bits ainsi que les Itanium. Ironiquement, c'est le système d'exploitation que les consommateurs moyens jugent compliqué qui apporte la gestion la plus simple de l'interface.

Avantages et inconvénients

Il existe un réel débat autour de la pertinence de l'UEFI. À la question « futur ou flop », il n'y a pas réellement de réponse, car même s'il est évident que certaines limitations du BIOS obligeront les marques à privilégier la nouvelle solution, il n'est pas dit que l'interface et le nombre de fonctionnalités changent. L'UEFI pourrait rester une technologie sous-exploitée avec une interface ressemblant à un BIOS classique. Il est vrai que c'est un élément de l'ordinateur que peu d'utilisateurs connaissent et configurent, ce qui limite les investissements et les innovations.

Plus de capacités

Les avantages de l'UEFI sont relativement connus et certains d'entre eux ont déjà été mentionnés. Le plus populaire est la possibilité d'utiliser des partitions GPT et de démarrer sur des volumes de plus de 2 Tio. Nous nous sommes suffisamment étendus sur la question pour pouvoir passer notre chemin. Ceux qui n'auraient pas suivi la succession d'articles sur le sujet sont vivement encouragés à lire « 2 To ça va, 3 To bonjour les dégâts ! ».

L'autre atout est la présence d'un système 32 bits ou 64 bits, contrairement au BIOS limité au 16 bits et à 1 Mo d'espace adressable. Comme nous l'avons vu, certaines applications UEFI ont accès à toute la mémoire disponible, même après le lancement de l'OS et une interface 64 bits offre des possibilités d'adressage incommensurablement plus élevées.

Les accès mémoires n'auraient pas de sens s'il n'y avait pas de programmes pour les exploiter et le fait que

l'EFI utilise un environnement shell facilite l'exécution d'applications. De même, la gestion d'extensions pouvant être lancées depuis n'importe quel support de stockage ouvre la voie à de nombreuses possibilités. Enfin, le gestionnaire de démarrage simplifie la gestion de multiples OS présents sur une seule machine.

Trop de complexité pour pas grand-chose

Néanmoins, L'UEFI est loin de faire l'unanimité. On se souvient d'un email incendiaire de Linus Torvalds en 2006 caractérisant l'interface de « *dégénération cérébrale d'Intel* ». Il critiquait alors le manque d'originalité de l'EFI par rapport au BIOS et ses origines IA64 clairement inadaptées aux systèmes grand public. Il se plaignait d'une complexité beaucoup plus importante pour un résultat qui n'est pas si différent d'un BIOS classique.

En effet, il faut toujours deux types de pilotes, un pour l'EFI, l'autre pour l'OS, comme du temps du BIOS. Une partie des fonctions de l'EFI est juste un calque de ce que l'on trouvait sur le BIOS (ACPI par exemple) ou de ce qui sera géré par l'OS et on peut s'interroger sur la pertinence de cette redondance. Enfin, les promesses de temps de démarrage sans précédent restent encore insatisfaites. Il n'y a pas de différence notable par rapport à un BIOS optimisé, pour le moment.

Le père de Linux a écrit ses lignes en 2006, mais malheureusement, il semblerait que les choses n'aient pas beaucoup changé en quatre ans. De plus, si l'utilisation du C offre des avantages par rapport à l'Assembleur, le langage n'est pas exempt de défaut et les éditeurs de BIOS reprochent principalement la taille du code, largement plus imposante, qui demande des puces mémoires d'une capacité plus importante, ce qui tend à augmenter les coûts de fabrication.

Malgré les critiques, les spécifications de l'UEFI ont beaucoup séduit les spécialistes de la virtualisation. En effet, nous avons ici un mécanisme qui permet d'ordonner les commandes envoyés au CPU en fonction de leur priorité, qui est capable de communiquer avec les composants de la machine, de gérer la mémoire, d'établir des connexions réseau, d'offrir une interface graphique intéressante, le tout avant même d'appeler le système d'exploitation. Très rapidement, les éditeurs ont compris que l'EFI pouvait servir à lancer un hyperviseur natif pour prendre le contrôle de la machine bien plus tôt qu'auparavant.

Nous ne reviendrons pas sur les types d'hyperviseurs que nous détaillons dans le chapitre « Définitions et évolution de la virtualisation » dans *VMWorld 2009 : l'avenir de la virtualisation se dessine*.
L'omniprésence des hyperviseurs de type 1 qui contrôlent tous les OS qui sont installés dessus et les ressemblances avec l'UEFI ont rapidement motivé les développeurs à écrire des hyperviseurs UEFI.

L'interface est un atout, car elle se pose entre les firmwares des composants et les systèmes d'exploitation et qu'elle favorise la création d'environnements riches. Utilisant les fonctionnalités inhérentes aux pilotes runtime que nous avons détaillé en page 4, l'UEFI est capable d'offrir des hyperviseurs natifs qui facilitent la gestion du parc de l'entreprise.

La dépendance matérielle de la virtualisation

Dans les faits, la virtualisation sur architectures x86 est particulièrement difficile, car selon les principes énoncés par Popek et Goldberg en 1974, dix-sept instructions x86 ISA ne peuvent pas être correctement virtualisées, ce qui oblige l'hyperviseur à trouver des parades. L'une d'entre elles se nomme para-virtualisation et en schématisant, elle accède au noyau pour réécrire les parties difficiles du code. Si cela fonctionne pour Linux, cette solution est incompatible avec Windows. L'autre solution est une traduction dynamique convertissant les codes binaires originels pour les rendre compatibles. Le gros problème est que ces deux solutions coûtent très cher en performance. C'est la raison pour laquelle AMD et Intel ont tout deux apporté la gestion de la virtualisation dans leur architecture en ajoutant 10 nouvelles instructions x86 ISA qui éliminent le besoin de passer par la paravirtualisation ou la traduction dynamique de binaires.

Le processeur a donc une place fondamentale dans la gestion des environnements virtuels et le fait que l'UEFI soit si proche des composants est un atout pour exploiter ses performances. Les solutions actuelles portent sur l'utilisation d'un hyperviseur qui se lance depuis le shell. Néanmoins, les machines de ce type

restent rares. Dell a proposé quelques modèles, mais les éditeurs d'hyperviseurs ne semblent pas encore vouloir embrasser cette solution.

Conclusion

L'EFI n'a pas séduit. C'est aujourd'hui un fait qu'il est impossible de nier. Les raisons restent par contre mal comprises et même les acteurs en haut de l'affiche ont du mal à mettre le doigt sur ce qui ne va pas.

Il est indéniable que l'UEFI a des atouts et qu'il remplacera le BIOS un jour où l'autre principalement parce que c'est le seul moyen de dépasser la limite des 2 Tio. Ses fondations open source, son langage de programmation en C, la gestion des bytecodes ou la possibilité de créer des applications avancées en font une interface intéressante. Même si elle n'est pas exempte de défauts, l'EFI a des atouts non négligeables et l'idée d'un hyperviseur de type 1 a un sérieux potentiel.

Le problème est ailleurs et réside dans le fait qu'aujourd'hui, il n'apporte pas suffisamment de fonctionnalités pour se distinguer du BIOS classique. À l'heure où les consommateurs cherchent un OS rapide leur permettant d'accéder immédiatement aux fonctions primaires de leurs machines pour éviter d'avoir à attendre que le système d'exploitation classique se lance, l'UEFI ne fait que s'occuper du démarrage de la machine. De plus, malgré ses possibilités, les temps d'amorçage restent similaires à ceux du BIOS.

Bref, l'UEFI est un BIOS moderne et c'est probablement là tout le problème. Le BIOS est une fonctionnalité nécessaire, mais très mal connue. Très peu d'utilisateurs vont régulièrement y faire un tour pour changer leur configuration. Même les plus experts ont tendance à oublier ce composant une fois qu'ils ont personnalisé leurs réglages. Si une interface plus accueillante est un point positif pour le profane, il laisse souvent l'expert indifférent. Pire encore, on peut soupçonner que les habitués n'aient pas aimé réapprendre à naviguer dans des menus pour retrouver les options qu'ils avaient l'habitude d'accéder sans même regarder l'écran, ce qui explique que beaucoup aient flashé leur carte mère ou que les éditeurs privilégient les interfaces ressemblant au BIOS classique.

Bref, le passage à l'UEFI est inévitable, mais son arrivée ne révolutionnera pas le monde de l'informatique et les restes du temps du BIOS ne sont pas prêt de s'effacer.