NEON : a ten for multimedia processing

A quick summary of NEON illuminated with some experimental values for comparisons.

Antoine Braut

Isep - 92130 Issy-les-moulineaux - France

Abstract

This article aim to provide a quick overview of NEON a SIMD from ARM. The overview will try to cover and explain the main features of NEON. By the end, this article will presents a protocol for a small benchmark of NEON on I.MX8 .

Cet article à pour objectif de faire une revue de NEON de chez ARM. Cette revue essaiera d'expliquer les principales caractéristiques et fonctionnalités du SIMD. A la fin de l'article, des résultats de test seront présenté afin d'avoir des valeurs numérique pour comparaison.

Index

Processeur : système éléctronique conçu pour effectuer des calculs et intérargir avec de la mémoire.
Coprocesseur : Processeur secondaire spécialisé dans une fonction précise (cryptographie, calcul flottant ...)
Jeu d'instruction : Ensemble des commandes assembleur compréhensible par un processeur.
FPU (Floating Point Unit/Unité de calcul flottant) : Coprocesseur spécialisé dans les calculs de nombres à virgules.
DSP : Coprocesseur spécialisé dans les opérations de traitement de signal, tels que la FFT.
SIMD (Single Instruction/Multiple Data) : Processeur spécialisé dans l'exécution d'une même instruction sur plusieurs données en paralléle.
Pipeline : Architecture permettant de découper une instruction en plusieurs "étages" pouvant être effectués parallèlement.
OS (Système d'exploitation) : Programme permettant de faire le lien entre le matériel hardware et les logiciels. Il est aussi responsable de l'ordonnancement des programmes.

1 NEON : Qu'est ce que c'est ?

Si le neon est bien connue pour être l'élément de numéro atomique 10, c'est aussi le nom donnée au SIMD acoompagnant les processeurs ARM depuis ARMV7. Un SIMD est définie dans la taxonomie de Flynn comme un type de processeur capable d'excéuter une même instruction sur plusieurs données. Cette capacité à parallélisé une même opération fait des SIMD d'excellents processeurs lorsqu'il s'agit de traiter de grandes matrices ou tableaux, type d'opérations que l'on retrouve énormément dans le traitement de flux multimédia. Ce type de processeurs peut aussi s'avérer utile lors de traitement de signaux ou de calcul à virgule.

NEON a été introduit en 2009 avec la gamme ARMV7-A[1]. Aussi appelé *Advanced SIMD* ou *MPE* pour *Media Process Engine*, il est conçu pour accélérer le traitement de contenue multimédia. Selon le coeur, il peut être intégré de base ou en option. Par exemple dans la gamme ARMV7, il est par défaut présent dans le cortex-A8, mais optionnel dans le cortex-A9. Pareil dans la gamme ARV8, où il est implémenté de base sur le cortex-A72, mais optionnel dans le cortex-A53. A l'instar du FPU VFPv3, NEON sert de coprocesseur pour accélérer le traitement de certaines données dans l'architecture ARM.

2 Spécificités techniques des SIMD NEON

Comme indiqué dans la documentation d'ARM[3], les processeurs NEON peuvent se combiner à toute autre processeur depuis la gamme ARMV7-A[1]. En tant que coprocesseur, il étend le pipeline du coeur ARM en plus de bénéficié de son propre pipeline optimisé. Il peut aussi lire les registres du coeur principale. NEON propose ses propres registres vectorielles, qu'il partage avec VFPv3 si celui-ci est présent, sous la forme de 32 registres de 64 bits. Ces registres peuvent aussi être utilisé comme 16 registres de 128 bits. Il est capable d'opérer à la fois en scalaires et en vectorielle. Le nombre maximum d'opérations réalisable en parallèle est de 16 sur des opérations de 8 bits. Dans le cadre de l'utilisation avec un OS, NEON peut être optimisé en multithreading si l'os prends en charge la regénétation des registres. Pour pouvoir utiliser NEON, il est nécessaire de charger les données depuis la mémoires dans les registres de NEON. Une fois les registres chargés, il est possible d'effectuer chaque opérations sur un à plusieurs registres à la fois. les processeurs NEON possèdent un jeu d'instruction spécifique permettant de déclencher ces opérations vectorielles. L'exemple ci-dessous montre le chargement de registres de NEON depuis des registres ARM :

```
VLD 2 .16 {d0,d1}, [r0]
```

Le V devant le mnémonique de l'instruction LD (load) indique une opération vectorielle. Les instructions de NEON sont très souvent préfacé d'un V afind de les différencier. Cette instructions va charger la mémoire du registre ARM r0 et remplir les registres de d0 à d1 avec. Les éléments à traiter son spécifié comme étant des éléments de 16 bits. Le 2 après l'opération indique le nombre de registre à entrelacé. NEON supportant l'entrelacement, il est possible de disperser une données sur plusieurs registres pour diviser le temps de lecture/écriture, le système traitant le registre entrelacé suivant le temps que celui venant d'être utilisé se rafraîchisse.
Utiliser le jeu d'instruction de NEON constitue la première méthode d'optimisation. C'est la plus bas niveau puisqu'elle consiste à faire appel directement au processeur en codant soi même son comportement. Une autre méthode consiste à passer par des librairies spécialisé comme OpenCL et ARMCL, qui permettent de faire appel aux différents coprocesseurs (ainsi qu'aux GPU) directement depuis le code C. D'autre librairies comme FFMPEG, libJPG-Turbo, OpenSSL, Cairo peuvent aussi être compilé avec les optimisations NEON afin d'en faire profiter les programmes qui l'utilise.
La dernière solution consiste à utiliser les options de compilation de GCC[4] afin de laisser le compilateur automatiquement utilisé les instructions NEON si possible. Il est possible d'activer la compilation optimisé avec NEON en rajoutant l'option *–mcpu=neon* ou

[1] Un coprocesseur NEON par coeur sur les architectures MPCORE

de spécifié un coeur avec simd comme par exemple *–march=cortex-a7+simd*. La qualité de l'optimisation par le compilateur dépends cependant de la qualité du code, une optimisation en assembleur ou avec des librairies spécialisé restera plus efficace dans la majeur partie des cas.

3 Test de performance et analyses

Cette partie va tenter de présenter l'impact de l'utilisation de NEON sur un processeur ARM. Les premiers résultats seront issue d'une étude expérimentale de l'auteur sur cortex-A53.

3.1 Benchmark ffmpeg

Pour permettre de mésurer l'impact de l'utilisation de NEON dans le traitement d'élément multimédia, nous allons mesurer le temps pris par la librairies FFMPEG pour décoder un élément multimédia. FFMPEG a été choisi car il facilement configurable avec ou sans l'optimisation pour NEON[2] en plus d'être le package le plus utilisé pour la lecture de flux multimédia.
Les mesures ont été effectuées sur I.MX8MQuad EVK avec des images Linux produite spécialement à l'aide du Yocto IMX version hardknot. Les paramètres de la plateforme sont présenté ci dessous :

- Carte : I.MX8MQEVK
- CPU : 4x Cortex-A53 @ 1.5Ghz
- Mémoire : 4GB DDR4 @ 3200MT
- Yocto : Meta-imx - hardknot (imx-image-core + ffmpeg)
- version de GCC : 10.2.0
- version de ffmpeg : 4.3.2

La configuration utilisé pour ffmpeg est donné en Annexe. Trois version différentes de l'image Linux ont été produite: une version ".nosimd" sans optimisation pour ffmpeg, une version "vfp" avec l'optimisation fpu de ffmpeg pour mesurer l'impact de l'utilisation d'une FPU, et une version "NEON" qui utilise l'optimisation NEON de ffmpeg. Pour ces 3 versions, un décodage de flux multimédia est effectué puis passer sur un fichier servant de "framebuffer leurre" en sortie afin de ne mesurer que le temps de décodage. Les deux fichiers multimédia utilisés sont : une image .png et une courte vidéo .mp4. Les paramètres des deux images sont les suivants

Table 1: Paramétres des flux multimédia

	Giraffe.png	Animation.mp4
format	.png	.mp4
codec	png	h264 + aac
taille	560*560	1920*1080
FPS	-	29.97
Durée(s)	-	9.66

La mesure est aussi effectué sur l'image girafe.png afin d'évaluer l'impact de NEON lors de traitement peu demandant.

Figure 1: girafe.png

Pour ce benchmark, les mesures de temps d'exécution (réel, système et utilisateur) seront données ainsi que la consommation maximum de mémoire. Les mesures sont effectuées grâce à l'option -benchmark de ffmpeg. La commande est la suivante :

```
ffmpeg -i <file> -benchmark -f null -
```

Les mesures ont été effectuées en QuadCore ainsi qu'en Monocore.

3.2 Résultats et analyse

Les tableaux suivants présente les résultats obtenue pour le benchmark ffmpeg sur I.MX8Mquad EVK. Les valeurs de temps sont données en secondes :

[2]FFMPEG utilise du code assembleur spécifique à NEON pour son optimisation

Table 2: Résultats girafe.png - Quadcore

valeur	.nosimd	vfp	NEON
Utime	0.015	0.12	0.019
Stime	0.006	0.01	0.003
Rtime	0.052	0.052	0.052
Max memory (kb)	11396	11528	11560

Table 3: Résultats animation.mp4 - Quadcore

valeur	.nosimd	vfp	NEON
Utime	11.528	11.552	7.681
Stime	0.145	0.149	0.157
Rtime	3.140	3.153	2.135
Max memory (kb)	95076	98028	97800
FPS	93	93	137
Megapixels/seconde	178	178	262

Premiérement, on peut remarquer que l'option -benchmark de ffmpeg manque de précision pour les petites valeurs (un real time supérieur au user time est plutôt suspect). Cependant, même imprécise, ces valeurs nous permettent de voir que l'utilisation de NEON pour des petites données n'entraîne pas un changement de performance majeur. A noter toute fois que l'impact de NEON étant proportionelle à la taille des données à traiter, le gain apporté par l'optimisation à pû se perdre dans l'imprécision.
Sur le traitement de la vidéo, on voit bien le gain de performance ammené par NEON avec des résultats presque 1.5x meilleur. On peut aussi remarquer que l'utilisation de VFP, l'autre option d'optimisation hardware, n'apporte pas de bénéfice. Les flux vidéo utilisant rarement des calculs avec virgule flottante, il est logique que l'utilisation d'une FPU n'apporte pas de bénéfice. Néanmoins, les résultats avec VFP seul confirme que l'amélioration de performance viens de l'utilisation de NEON.

Table 4: Résultats girafe.png - Monocore

valeur	.nosimd	vfp	NEON
Utime	0.010	0.010	0.010
Stime	0.000*	0.000*	0.000*
Rtime	0.010	0.0010	0.010
Max memory (kb)	N/A	10568	10604

*Le timer n'a probablement pas eu le temps de se déclencher correctement

Table 5: Résultats animation.mp4 - Monocore

valeur	.nosimd	vfp	NEON
Utime	10.473	10.442	6.178
Stime	0.062	0.078	0.091
Rtime	10.679	10.666	6.360
Max memory (kb)	80276	80408	80232
FPS	27	27	46

Les données obtenue pour le traitement de l'image en monocoeur semble fortement imprécise. FFMPEG utilisant un timer soft pour son option benchmark, il est possible que le passage en monocoeur l'ai rendu encore plus imprécis pour de très faible valeur.
Sans surprise on observe un effondrement des performances lors du passage en monocoeur, les opérations multimédia étant fortement optimisé par le multiprocess. Néanmoins si la basse de performance est quasiment la même entre la version non optimisé et NEON (augmentation du temps processeur par 3 environ), le User time est légèrement plus faible que la version QuadCore, potentiellement dû à la réduction de l'impact de l'os lors de la mesure.

4 Conclusion

L'extension NEON propose une optimisation très intéressante pour du traitement vidéo. Que ce soit avec l'optimisation du compilateur ou des instructions assembleur spécifique, utilisé NEON réduit les temps de calcul des éléments tels que les matrices ou les vecteur. Cependant son efficacité devient limité lors de son utilisation sur des éléments de petite taille, ou du calcul flottant. Sur les cortex basse et moyenne gamme qui ne l'inplémente pas par défaut, son ajout n'est pas obligatoire pour certaines applications. Cepenant, les cortex-A étant maintenant principalement utilisé pour du multimédia, il est logique de retrouver NEON dans presque toutes les variantes proposées par les fondeurs.
Pour compléter cette étude, des données sur d'autre type de mesures tels que la consommation, d'autre type de calcul ainsi que du calcul flottant aurait été bienvenue. Des comparaisons numérique avec d'autre SIMD aurait aussi été intéressant.

5 References

[1] Cortex-A9 Technical Reference Manual r4p1 - ARM - 15/06/12

[2] ARM Architecture Reference Manual I.3 - ARM - 19/08/22

[3] NEON programmer's guide V1 - ARM - 28/06/13

[4] ffmpeg manual - ffmpeg - consulté le 17/09/22 - https://www.ffmpeg.org/ffmpeg.html

[5] Gnu C Compiler options summary - GNU - consulté le 17/09/22 - https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html

Annexes

Configuration de ffmpeg

La configuration suivante a été utilisé pour ffmpeg, le mode VFP retire –disable-vfp et le mode NEON retire aussi –disable-neon :

```
–mcpu=cortex–a53 –march=armv8–a+crc+crypto –fstack–protector–strong
    –O2 –D_FORTIFY_SOURCE=2 –Wformat –Wformat–security –Werror=format
    –security ––sysroot=/recipe–sysroot ' ––cc='aarch64–poky–linux–gcc
     –mcpu=cortex–a53 –march=armv8–a+crc+crypto –fstack–protector–
    strong –O2 –D_FORTIFY_SOURCE=2 –Wformat –Wformat–security –Werror
    =format–security ––sysroot=/recipe–sysroot ' ––cxx='aarch64–poky–
    linux–g++ –mcpu=cortex–a53 –march=armv8–a+crc+crypto –fstack–
    protector–strong –O2 –D_FORTIFY_SOURCE=2 –Wformat –Wformat–
    security –Werror=format–security ––sysroot=/recipe–sysroot ' ––
    arch=aarch64 ––target–os=linux ––enable–cross–compile ––extra–
    cflags=' –O2 –pipe –g –feliminate–unused–debug–types –fmacro–
    prefix–map==/usr/src/debug/ffmpeg/4.3.2–r0 –fdebug–prefix–map==/
    usr/src/debug/ffmpeg/4.3.2–r0 –fdebug–prefix–map=/recipe–sysroot=
     –fdebug–prefix–map=/recipe–sysroot–native= –mcpu=cortex–a53 –
    march=armv8–a+crc+crypto –fstack–protector–strong –O2 –
    D_FORTIFY_SOURCE=2 –Wformat –Wformat–security –Werror=format–
    security ––sysroot=/recipe–sysroot ' ––extra–ldflags='–Wl,–O1 –Wl
    ,––hash–style=gnu –Wl,––as–needed –Wl,–z,relro,–z,now' ––sysroot
    =/recipe–sysroot ––libdir=/usr/lib ––shlibdir=/usr/lib ––datadir
    =/usr/share/ffmpeg ––cpu=cortex–a53 ––pkg–config=pkg–config ––
    disable–vfp ––disable–neon ––disable–static ––enable–alsa ––
    disable–altivec ––enable–avcodec ––enable–avdevice ––enable–
    avfilter ––enable–avformat ––enable–avresample ––enable–bzlib ––
    disable–libfdk–aac ––disable–gpl ––disable–libgsm ––disable–indev
    =jack ––disable–libvorbis ––enable–lzma ––disable–libmfx ––
    disable–mipsdsp ––disable–mipsdspr2 ––disable–libmp3lame ––
    disable–openssl ––enable–pic ––enable–postproc ––enable–pthreads
    ––disable–sdl2 ––enable–shared ––disable–libspeex ––disable–
    libsrt ––disable–stripping ––enable–swresample ––enable–swscale
    ––enable–libtheora ––disable–vaapi ––disable–vdpau ––disable–
    libvpx ––disable–libx264 ––disable–libx265 ––disable–libxcb ––
    disable–outdev=xv ––enable–zlib
```