Etude WCET de programmes Faust avec a3

Benoit Pin
Pierre Jouvelot

MINES ParisTech, PSL Research University, France

28 avril 2017

1. Motivation

Faust[1] est un langage de programmation spécialisé dans la synthèse sonore temps-réel à des fins musicales et artistiques. L'utilisateur typique de Faust est un réalisateur en informatique musicale (RIM) pour qui l'informatique est un outil de travail essentiel, sans être sa discipline première. Avec Faust, un musicien peut créer de toute pièce des instruments logiciels ou des effets, en se concentrant en premier lieu sur la facture instrumentale et les aspects esthétiques des traitements sonores. Ces instruments ou effets vont être décrits sous forme d'un code source Faust. Par la suite, ce code est compilé et devient utilisable par les musiciens interprètes. Le compilateur Faust offre un vaste choix d'architectures matérielles ou logicielles pour son exécution. De cette façon, il est possible d'élaborer des chaînes de traitement sonores complexes où des composants décrits avec Faust interagissent avec des machines du commerce, des stations audio-numérique (DAW), etc.

Par conception, l'implémentation la plus concrète des traitements audio-numériques est effectuée par le compilateur Faust. À l'origine, ce choix de conception a été motivé par l'observation empirique des algorithmes écrits « à la main » par des musiciens développeurs : les performances étaient généralement moindres que des algorithmes écrits par des experts. Par extension, les premiers travaux sur le langage Faust ont également mis en évidence qu'il est possible de générer des implémentations d'algorithmes de bonne facture et que seuls des experts confirmés pouvaient en améliorer les performances. Le langage Faust apporte donc un compromis pour rendre la création de traitements sonores accessibles aux néophytes tout en garantissant des performances accrues. On remarquera aussi que le critère de performance est particulièrement important à la vue des contraintes d'exécution temps-réel imposées par le concert.

Au delà des observations constatées et d'une longue expérience d'œuvres élaborées avec Faust et éprouvées dans des conditions réelles d'interprétation en public, qui donnent déjà un haut niveau de confiance dans la technologie, il est possible d'aller plus loin et de déterminer analytiquement les performances d'un programme. En particulier, il est possible de calculer le temps d'exécution d'un traitement sur une architecture définie.

Calculer le temps exact d'exécution d'un programme est un problème indécidable. Aussi, une approche approximative est nécessaire. Dans le cas présent, nous avons cherché à étudier les logiciels actuellement disponibles pour mesurer le WCET (*Worst Case Execution Time*) d'un programme, qui est une borne supérieure sur tous ses temps d'exécution possibles. Cette mesure est particulièrement pertinente pour un langage comme Faust, qui suit l'approche synchrone [2]. En effet, dans ce cas, les traitements effectués sont effectués par pas de temps réglés sur la fréquence d'échantillonnage ; ils doivent être terminés à la fin de chaque période, ce qui peut être assuré par un analyse WCET.

2. Cadre de l'étude

Classiquement, un programme Faust est compilé vers un code source C++. Ce dernier se divise en deux parties (sans s'y limiter) :
1- le traitement audio ;
2- l'affichage de l'interface homme-machine.

1 http://faust.grame.fr/

On peut également mentionner d'autres éléments qui composent un code C++ généré par le compilateur Faust, notamment des fonctions d'initialisation et d'interface vers l'architecture souhaitée ou encore le nécessaire pour le contrôle-commande extérieur par OSC[2] ou MIDI[3].

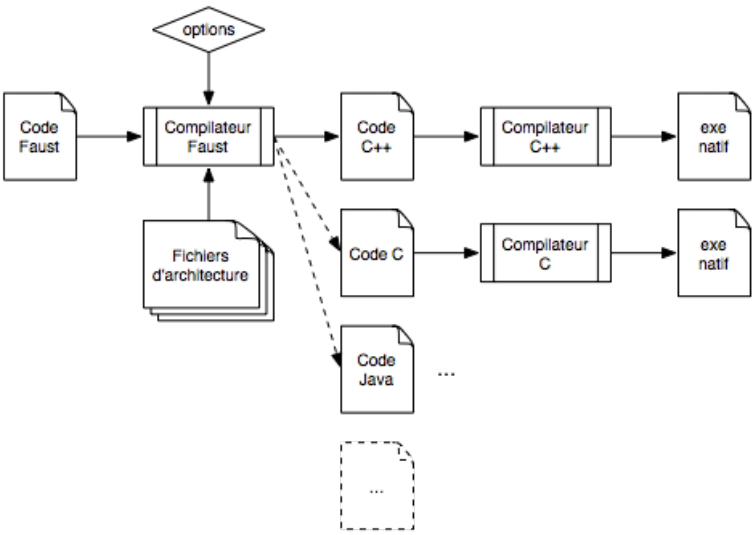

Illustration 1: Chaine de compilation Faust

Le code C++ est enfin compilé sous forme d'un exécutable au format natif de l'architecture cible. D'autres options techniques peuvent être sollicitées à partir d'un code Faust, mais la chaîne de compilation suivra toujours le schéma de la figure 1.

Pour effectuer les analyses de WCET, nous avons utilisé le logiciel *a3* de la société AbsInt[4]. Nous avons utilisé une version de ce logiciel conçue pour analyser des exécutables compilés pour les architectures ARM et originellement écrits en C.

Pour que les résultats obtenus aient une signification concrète nous nous sommes intéressés à la pédale d'effets programmable OWL[5] qui fait partie des architectures supportées. Cette pédale dispose d'un processeur ARM Cortex M4 cadencé à 168 MHz avec support matériel des nombres flottants (FPU) et 192 ko de mémoire RAM. Elle constitue un bon exemple d'étude à la vue de ses ressources en calcul relativement modestes (comparées à un PC actuel) et pour lequel des programmes optimisés sont importants.

Une fois que l'architecture et le langage ont été définis, nous avons pu mettre en place le workflow de traitement pour calculer le WCET à partir d'un code Faust. Pour mener notre étude, toujours dans l'optique d'obtenir les résultats concrets, nous avons eu recours aux codes Faust des modules présent dans l'application Faust Playground[6].

3. Chaîne de compilation

Dans le code source qui est généré par le compilateur Faust, la partie concernant le DSP[7] est générique (indépendante de l'architecture logicielle ou matérielle cible). Cette partie est facilement repérable à la lecture : elle est placée en fin de fichier et est implémentée dans une unique fonction nommée typiquement « computemydsp ». On notera également que le corps de cette fonction est implémenté de manière identique, si l'on utilise indifféremment le langage C ou C++ comme option. La seule dépendance que l'on peut rencontrer dans l'implémentation de cette fonction est la bibliothèque *math*. L'implémentation de cette dernière influe sur le WCET, en particulier sur les architectures « embarquées », qui peuvent être dotées ou démunies d'unité de traitement des flottants native.

Pour produire le code source des exécutables donnés au logiciel d'analyse, nous avons compilé les codes Faust comme ceci :

2 Open Sound Control
3 Musical Instrument Digital Interface
4 http://www.absint.com/
5 https://hoxtonowl.com/
6 http://www.grame.fr/mediation/faust-audio-playground
7 Digital Signal Processing

```
faust -lang c <fichier.dsp>
```

De cette façon, nous obtenons un code écrit en langage C, qui ne comporte pas de spécificité d'une architecture. Ce code ne peut être compilé en l'état ; la seule option -lang n'est pas suffisante et il faudrait préciser une architecture pour y parvenir, mais c'est précisément ce que nous évitons. Pour obtenir un code C apte à être compilé, il nous faut écrire une fonction « main ».

Nous avons implémenté cette fonction *main* de la manière la plus minimale, à savoir :
1- allocation mémoire de la structure de données du DSP ;
2- définition de la fréquence d'échantillonnage ;
3- allocation des tampons d'échantillons d'entrée et de sortie ;
4- appel du DSP (fonction « computemydsp » sus-mentionnée) ;
5- désallocation de la mémoire.

La différence entre cette implémentation minimale par rapport à une architecture logicielle réelle (par exemple ALSA, Jack…) résidera sur l'appel de la fonction de calcul du DSP : normalement, c'est le driver de l'architecture audio qui se charge de l'appeler, selon ses propres caractéristiques.

Le processus pour créer un code C prêt à être compilé a été automatisé. Nous avons développé un script en python et utilisé un *parser* C pour effectuer quelques opérations de transformation source à source. Ces transformations consistent, d'une part, à implémenter automatiquement la fonction *main* et, d'autre part à supprimer certaines fonctions non utiles à notre étude, et qui peuvent amener des dépendances problématiques à la compilation. Pour la manipulation du code C, nous avons utilisé la bibliothèque pycparser[8] : un *parser* C « pur python ». Vu la faible étendue des transformations à effectuer (filtrage de fonctions ou de directives de pré-processeur), il aurait été envisageable d'opérer par reconnaissance de motifs sur le code source. Cependant, la facilité de mise en œuvre de *pycparser* et la souplesse de python, en tant que langage de script, ont permis d'adopter une technique plus académique et plus fiable.

Nous avons utilisé le compilateur GCC pour générer les exécutables sur architecture ARM. Plus précisément, nous avons utilisé des chaînes de compilation croisées (*cross compiling*) sur un ordinateur Intel x86 Linux. Pour les exécutables avec support natif des flottants, nous avons employé la chaîne « GNU ARM Embedded Toolchain », disponible sur arm.com[9] – une version officielle, maintenue par ARM. Pour les versions avec flottants logiciels, nous avons utilisé le générateur de chaînes de compilation *croostool-NG*[10] et le compilateur d'ARM.

4. Évaluation du WCET

Mise en oeuvre

Dès lors que l'exécutable est créé, il reste quelques éléments à caractériser pour obtenir un résultat chiffré. Il faut, en premier lieu, informer le logiciel d'analyse *a3* des caractéristiques du matériel cible. Pour notre cas d'étude portant sur la pédale OWL, nous précisons les éléments suivants :
— la fréquence d'horloge du processeur ;
— le compilateur utilisé ;
— le jeu d'instructions ;
— les caractéristiques de la mémoire RAM.

Ces informations sont transcrites dans les fichiers de configuration du projet et de l'analyse WCET. Nous présentons ci-dessous, successivement, ces fichiers. Pour pouvoir analyser un lot d'exécutables, nous avons généré par programmation les fichiers xml de projet. On précise que, par conception, le logiciel *a3* ne traite qu'un exécutable par projet. Ensuite, il peut y avoir plusieurs analyses au sein d'un projet, mais, en ce qui nous concerne, nous avons configuré chaque projet pour une unique analyse du WCET de la fonction *computemydsp*.

```
<project xmlns="http://www.absint.com/apx" version="16.10" build="269677" target="arm">
 <options xmlns="http://www.absint.com/apx">
  <arm_options xmlns="http://www.absint.com/apx">
   <general xmlns="http://www.absint.com/apx">
    <target xmlns="http://www.absint.com/apx">Cortex-M4</target>
```

8 https://pypi.python.org/pypi/pycparser
9 https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
10 https://crosstool-ng.github.io

```
   </general>
   <instruction_set>THUMB</instruction_set>
  </arm_options>
  <analyses_options xmlns="http://www.absint.com/apx">
   <extract_annotations_from_source_files xmlns="http://www.absint.com/apx">true</ex-
tract_annotations_from_source_files>
  </analyses_options>
 </options>
 […]
```

Fichier de configuration du projet (extrait)

```
ais2 {
    compiler: "arm-gcc";
    clock: 168 MHz;

    area 0x00000000 to 0x1fffffff {
         attribute "bus": "Private";
         attribute "memory_type": "CODE";
         attribute "port_width": 32;
         access time: 1cycle;
}
```

Fichier de configuration de l'analyse WCET (extrait)

Pour finir, et peut-être est-ce la partie la plus délicate, il faut caractériser les boucles présentes dans le code source en fournissant, pour chaque boucle, le nombre d'itérations maximum (s'il ne peut être inféré par *a3*). Le logiciel *a3* dispose d'outils efficients pour détecter les boucles, naviguer entre un *call-graph* interactif, le code source, le code assembleur, le journal d'analyse et les fichiers de configuration. La caractérisation des boucles peut être définie directement dans le code source C, via l'ajout de commentaires dûment formatés, ou encore dans le fichier de configuration de l'analyse. Un système d'étiquetage des boucles est intégré au logiciel *a3.* L'utilisation typique consiste à lancer une première fois l'analyse du WCET. Le journal indique alors, sous forme de *warnings*, les boucles détectées où le nombre d'itérations doit être défini.

Les boucles présentes dans le code C issu du compilateur Faust ne posent pas de difficulté à caractériser : il s'agit de boucles *for* avec incrémentation ou décrémentation d'un compteur parfaitement lisibles. En revanche, les boucles présentes dans le code de bibliothèques externes peuvent être bien plus difficiles à borner. La compréhension du code source d'une bibliothèque et du domaine qu'elle concerne peut être nécessaire à un bornage correct.

Les codes générés par Faust n'ont recours qu'à la bibliothèque *math* concernant la partie DSP sur laquelle nous nous focalisons. Ceci est certainement un avantage, cependant le domaine concerné est loin d'être le plus trivial.

Exemple : Birds.dsp

Nous nous intéressons dans cette exemple au générateur audio *Birds* présent dans la bibliothèque de Faustplayground. Nous exposons ci-dessous des extraits du code C de la fonction de traitement audio. Nous faisons apparaitre les bornes de deux boucles *for*, exprimées sous forme de commentaires.

```
void computemydsp(mydsp *dsp, int count, FAUSTFLOAT **inputs, FAUSTFLOAT **outputs)
{
[…]
  int i;
  for (i = 0; i < count; i = i + 1)
  {
    /* ai: ais2 { loop here bound: 0 .. 1; } */
    […]
    dsp->fRec11[0] =
              fmodf((float) (((int) (2994.2312f * (dsp->fRec11[2] + dsp->fRec11[3])))
                             + 38125),
                    22.0f);
```

```
    {
     int j0;
     for (j0 = 3; j0 > 0; j0 = j0 - 1)
     {
       /* ai: ais2 { loop here bound: 0 .. 3; } */
[…]
}
```

Parmi les fonctions de la bibliothèque math, *fmodf*, contient 4 boucles dans son implémentation. Nous devons analyser *de visu* son code source (ef_fmodf.c) pour déterminer les bornes. Nous analysons la première boucle :

```
float __ieee754_fmodf(float x, float y) {
…
if(FLT_UWORD_IS_SUBNORMAL(hx)) { /* subnormal x */
       for (ix = -126,i=(hx<<8); i>0; i<<=1) ix -=1;
} else ix = (hx>>23)-127;
…
}
```

Pour borner cette boucle, il faut savoir que *ix* et *hx* sont, respectivement, l'exposant et la mantisse de l'argument *x* de la fonction. On rappelle les éléments de la norme ieee754 (représentation en mémoire des nombres flottants) nécessaires à la compréhension :
— nous (i.e., Faust) travaillons en simple précision soit des mots de 32 bits ;
— du bit de poids fort au bit de poids faible, on a :
  • 1 bit de signe (le signe du flottant représenté) ;
  • 8 bits pour l'exposant (entier signé)
  • 23 bits pour la mantisse (entier non signé).
— un flottant « dénormalisé » se caractérise par ses 8 bits d'exposant à zéro ;
— pour un flottant dénormalisé, la valeur de l'exposant résultant est la soustraction du nombre de bits à zéro avant le premier bit à un (présent sur la mantisse) - 126 - 1.

À la lumière de ces spécifications, on peut comprendre que l'opération réalisée par la boucle consiste à calculer l'exposant de l'argument *x* (sachant qu'il est dénormalisé à cet endroit du programme). Au départ, la mantisse est décalée de huit bits à gauche. Dès lors, à chaque itération, un décalage à gauche est effectué. Pour peu que ce décalage positionne à 1 le MSB de la variable *i*, alors, la boucle s'arrête, car, *i* étant un entier signé, il sera réputé négatif. On en arrive à la conclusion que, dans le pire des cas, il peut y avoir 23 itérations (cas où la mantisse vaut 1).

La compréhension des autres boucles présentes dans l'implémentation de *fmodf* nécessite elles aussi une connaissance de la norme ieee754 et des idiomes de programmation bit à bit (test du « signe », addition d'un nombre par lui-même pour réaliser un décalage à gauche, etc.).

Cet exemple illustre quelles peuvent être les difficultés rencontrées dans la caractérisation d'un code bas niveau et modérément commenté. Il montre aussi que les flottants dénormalisés provoquent un surcoût de calcul que nous allons pouvoir mesurer en calculant le WCET.

Pour *fmodf*, voici comment sont bornées les quatre boucles lorsque les flottants dénormalisés sont supportés :

```
    loop "__ieee754_fmodf.L1" { bound: 0 .. 23; }
    loop "__ieee754_fmodf.L2" { bound: 0 .. 23; }
    loop "__ieee754_fmodf.L3" { bound: 0 .. 276; }
    loop "__ieee754_fmodf.L4" { bound: 0 .. 23; }
```

et non supportés :

```
    loop "__ieee754_fmodf.L1" { bound: 0 .. 0; }
    loop "__ieee754_fmodf.L2" { bound: 0 .. 0; }
    loop "__ieee754_fmodf.L3" { bound: 0 .. 253; }
    loop "__ieee754_fmodf.L4" { bound: 0 .. 0; }
```

Sans prise en charge des flottants dénormalisés, le WCET de *computemydsp* est de 36 144 cycles ; avec, 40 104 cycles, soit un surcoût de près de 11%.

Pour une application audio, il peut être opportun de ne pas utiliser les nombres dénormalisés, car il s'agit de valeurs très proches de zéro, et, concrètement, il peut s'agir de signaux audio tellement faibles qu'ils sont inaudibles.

On trouve dans le fichier d'en-têtes *fdlibm.h* une directive de préprocesseur, `_FLT_NO_DENORMALS`, qui permet de désactiver les nombres dénormalisés. Il faut noter que la seule présence (ou l'absence) de cette directive ne pourra influer directement sur les résultats de calcul du WCET, en l'état actuel de l'implémentation de la bibliothèque *math*. En effet, les routines de traitement spécifiques aux nombres dénormalisés ne sont pas soumises à des directives de compilation conditionnelle. Il y a donc à chaque fois un test pour déterminer si le code doit être exécuté. La compilation conditionnelle engendrée par `_FLT_NO_DENORMALS` ne porte que sur l'expression du test. La conséquence vis-à-vis l'analyse du WCET est qu'il faut alors fournir des valeurs adaptées pour les bornes des boucles concernées, car le graphe d'appels reste identique.

Par extension, si l'on souhaite mesurer l'influence d'options de compilation différentes pour un programme donné, il faut avoir conscience que la mesure du WCET ne sera pas nécessairement modifiée, quand bien même l'exécution pourrait montrer des écarts de performance. Autrement-dit, un changement des options de compilation doit être suivi d'une nouvelle analyse des boucles, sans quoi, le WCET serait, au mieux, imprécis et, au pire faux.

Le logiciel *a3* apporte une suite d'outils d'analyse qui aide à ce propos. Il reste néanmoins une part incompressible d'analyse sémantique du code qui est sous la responsabilité de l'utilisateur.

5. Constitution des lots pour les mesures

Nous avons effectué 6 lots de mesures sur les codes Faust standards de Faustplayground.
Les programmes ont été compilés avec gcc, la version librement diffusée sur le site arm.com : arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 6.3.1 20170215 (release) [ARM/embedded-6-branch revision 245512]. Nous avons utilisé les options de compilation suivantes :

— Cortex-M4, No FP
`-mthumb -mcpu=cortex-m4`

— Cortex-M4, Soft FP
`-mthumb -mcpu=cortex-m4 -mfloat-abi=softfp -mfpu=fpv4-sp-d16`

— Cortex-M4 Hard FP
`-mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16`

Pour ces deux derniers jeux d'options, on rappelle que l'unité FPU est utilisée (la dénomination « Soft FP » n'est pas très heureuse). Pour l'option `-mfloat-abi=softfp`, le code assembleur généré sera conforme au standard Linux Application Binary Interface (ABI). Pour l'option `-mfloat-abi=hard`, la séquence d'instructions assembleur est optimisée pour une utilisation plus importante des éléments d'architecture matériels dédiés aux flottants.

Et pour chacune de ces trois variantes de compilation, nous avons mesuré le WCET avec et sans support de la dénormalisation – soit six lots de résultats.

6. Commentaires sur les résultats

Nous discutons ci-dessous deux points qui nous ont paru importants : la gestion du type d'interace Floating Point et la vérification de l'hypothèse synchrone.

Interfacage FP

Un résultat peut sembler surprenant : pour une majorité de cas, l'option `-mfloat-abi=softfp` donne un WCET plus faible qu'avec l'option `-mfloat-abi=hard`. Nous allons maintenant expliquer cette différence en examinant l'exemple ASREnvelope. Ci-après, on trouvera les graphes d'appels, dans cet ordre `softp` puis `hard`. Nous constatons que la fonction *min*, (minimum de deux flottants) s'exé-

cute sensiblement plus lentement dans le second cas, malgré l'option *hard* (138 cycles contre 129

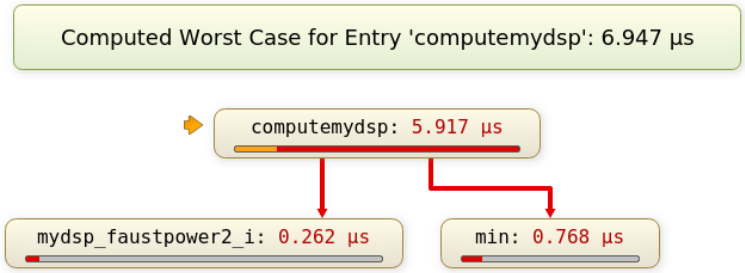

Illustration 2: Analyse en mode `softfp`

pour la version *softfp*).

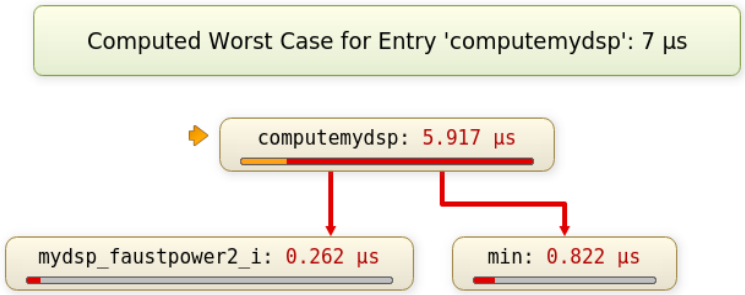

Illustration 3: Analyse en mode hard

Le fin mot de l'histoire se trouve dans le code assembleur affiché ci-après.

Version *softfp*

```
min:
   float min(float a, float b) {
thumb::0x814e push   {r7}
thumb::0x8150 sub    sp, sp, #0xc
thumb::0x8152 add    r7, sp, #0
thumb::0x8154 str    r0, [r7, #4]
thumb::0x8156 str    r1, [r7]
   return (a < b) ? a : b;
thumb::0x8158 vldr   s14, [r7, #+4]
thumb::0x815c vldr   s15, [r7]
thumb::0x8160 vcmpe.f32 s14, s15
thumb::0x8164 vmrs   apsr, fpscr
thumb::0x8168 bpl    0x816e <0x816e>

thumb::0x816a ldr    r3, [r7, #4]
thumb::0x816c b      0x8170 <0x8170>

thumb::0x816e ldr    r3, [r7]

   }
thumb::0x8170 mov    r0, r3
thumb::0x8172 adds   r7, #0xc
thumb::0x8174 mov    sp, r7
thumb::0x8176 pop.w  r7
thumb::0x817a bx     lr
```

Version *hard*

```
min:
   float min(float a, float b) {
thumb::0x8158 push   {r7}
thumb::0x815a sub    sp, sp, #0xc
thumb::0x815c add    r7, sp, #0
thumb::0x815e vstr   s0, [r7, #+4]
thumb::0x8162 vstr   s1, [r7]
   return (a < b) ? a : b;
thumb::0x8166 vldr   s14, [r7, #+4]
thumb::0x816a vldr   s15, [r7]
thumb::0x816e vcmpe.f32 s14, s15
thumb::0x8172 vmrs   apsr, fpscr
thumb::0x8176 bpl    0x817c <0x817c>

thumb::0x8178 ldr    r3, [r7, #4]
thumb::0x817a b      0x817e <0x817e>

thumb::0x817c ldr    r3, [r7]
thumb::0x817e vmov   s15, r3

   }
thumb::0x8182 vmov.f32      s0, s15
thumb::0x8186 adds   r7, #0xc
thumb::0x8188 mov    sp, r7
```

```
thumb::0x818a pop.w  r7                    thumb::0x818e bx     lr
```

La version *hard* se distingue par une instruction supplémentaire (vmov) et l'emploi des registres s0 et s1 au lieu de r0 et r1, lors des manipulations de la pile préalables à l'exécution de la comparaison. On rappelle que les registres r sont à usage général et les registres tandis que les registres s sont spécifiques au flottants. Ces opérations de chargement de registres de natures différentes sont plus coûteuses que pour des chargements entre registres de même natures et provoquent une baisse de performance.

Seuls 6 des 87 exemples analysés présentent un gain de performance avec l'option *Hard FP*. Nous observons le détail du WCET du programme Faust ZitaReverb (tableau ci-dessous) – un cas où l'option Hard FP provoque un gain de performance. Le tableau indique que, sur les 18 fonctions appelées, 7 s'exécutent plus lentement et 5 plus vite. Le gain de performance est alors obtenu si le programme appelle plus souvent les fonctions plus rapides.

Soft FP							Hard FP			
Routine	#	Cumul. (cycle)	Cumul.	Contrib. (cycle)	Contrib.	Speedup	Cumul. (cycles)	Cumul.	Contrib. (cycle)	Contrib
computemydsp	1	17888	0.107ms	157	0.935us	0	17732	0.106ms	157	0.935us
Computemydsp.L1	2	17731	0.106ms	5683	33.828us	26	17575	0.105ms	5709	33.983us
expf	8	5672	33.762us	2120	12.620us	-56	5632	33.524us	2064	12.286us
sqrtf	4	5376	32.000us	992	5.905us	-20	5184	30.858us	972	5.786us
__ieee754_sqrtf	4	3208	19.096us	300	1.786us	16	3036	18.072us	316	1.881us
__ieee754_sqrtf.L1	100	2064	12.286us	2064	12.286us	-188	1876	11.167us	1876	11.167us
__ieee754_expf	8	1776	10.572us	1776	10.572us	-64	1712	10.191us	1712	10.191us
__aeabi_f2d	12	1164	6.929us	1164	6.929us	40	1204	7.167us	1204	7.167us
__aeabi_d2f	12	960	5.715us	960	5.715us	32	992	5.905us	992	5.905us
__ieee754_sqrtf.L2	104	844	5.024us	844	5.024us	0	844	5.024us	844	5.024us
mydsp_faustpower2_f	12	516	3.072us	516	3.072us	4	520	3.096us	520	3.096us
max	6	354	2.108us	354	2.108us	46	400	2.381us	400	2.381us
__aeabi_ddiv	4	340	2.024us	120	0.715us	0	340	2.024us	120	0.715us
__errno	24	300	1.786us	300	1.786us	-8	292	1.739us	292	1.739us
Anon_0xbe1e	4	220	1.310us	220	1.310us	0	220	1.310us	220	1.310us
min	2	130	0.774us	130	0.774us	0	130	0.774us	130	0.774us
finitef	8	104	0.620us	104	0.620us	16	120	0.715us	120	0.715us
matherr	12	84	0.500us	84	0.500us	0	84	0.500us	84	0.500us

L'utilisation de l'option h*ard* pourrait s'avérer gagnante si le code assembleur généré mettait à profit les quelques instructions vectorielles disponibles sur les flottants. Sur le processeur ARM Cortex-M4, ces instructions se limitent aux opérations de manipulation de la mémoire, les instructions d'arithmétique n'étant pas vectorielles. La présentation marketing du Cortex-M4 mentionne des instructions SIMD, mais, dans le détail, elles ne s'appliquent qu'aux entiers.

Quoi qu'il en soit, les gains de performance obtenus par l'utilisation du FPU sont très significatifs, et, en comparaison, les différences de performances entre les deux options d'utilisation du FPU restent marginales.

Hypothèse synchrone

Le traitement de signal audio dans un langage tel que Faust doit satisfaire à ce que l'on nomme l'hypothèse synchrone [2] : le temps de traitement d'un échantillon doit être inférieur à la période du CPU. La fréquence maximum d'échantillonnage admissible pour chaque programme correspond ainsi à l'inverse du temps cumulé (colonne « Cumul. Time » des tableaux fournis en annexe). Pour une lecture plus directe, on peut rappeler que le temps de calcul maximum d'un échantillon ne doit pas dépasser 24,4 µs à 44,1 kHz (qualité CD) ou 20,8 µs à 48 kHz (autre standard audio fréquemment utilisé dans le monde professionnel). On pourra alors identifier rapidement les programmes qui pourront s'exécuter sans encombre sur la pédale OWL.

On voit ainsi rapidement que de nombreux programmes Faust nécessitent, apparemment, une puissance de calcul supérieure à celle fournie par la plateforme étudiée. Une caractéristique souvent rencontrée dans les programmes qui ne peuvent s'exécuter à une fréquence d'échantillonnage conventionnelle est l'usage important des fonctions *powf* ou *expf* dont les WCET sont, respectivement de 4,8 µs et 1,3 µs. Pour peu que le programme ait recours quelques dizaines de fois à ces fonctions, la limite d'une vingtaine de micro-secondes est alors vite atteint. Le programme AtonalSoftHarp en est un exemple typique avec 20 appels à *powf* et 16 appels à *expf*.

7. Perspectives

Pour cette étude, nous avons utilisé le logiciel *a3* qui dispose de tous les attributs d'un IDE dédié à l'analyse de programmes, le WCET étant une fonction majeure. Un tel logiciel, avec un haut niveau d'intégration et de finition, a permis d'accéder facilement aux problématiques d'analyses qui, dans le détail, sont assez pointues.

D'autres logiciels du même type existent, même s'ils sont moins intégrés. Nous pouvons mentionner le framework OTAWA[11], un logiciel libre, d'origine universitaire, également dédié à l'analyse du WCET. Ce logiciel peut être utilisé de deux façons : soit, à l'instar de a3, en mode IDE, au moyen d'un plugin greffé dans l'IDE Eclipse, soit comme bibliothèque C++ pour créer des programmes d'analyse.

OTAWA dispose également d'une syntaxe spécifique pour caractériser les boucles des programmes, sous forme de commentaires dans le code C à analyser. Il serait envisageable d'étendre le compilateur Faust pour que ces annotations soient automatiquement ajoutées. En effet, le compilateur Faust, lorsqu'il génère le code du DSP, a une connaissance complète des boucles et l'impression de commentaires à destination d'OTAWA ne devraient être qu'une question de mise en forme. Enfin, il serait également possible d'écrire un programme d'analyse avec la bibliothèque OTAWA suffisamment générique pour analyser un exécutable généré à partir d'un code Faust quelconque.

L'analyse des résultats nous a montré que l'hypothèse synchrone risque de ne pas être vérifiée sur certains programmes lourds de traitement audio. Ceux qui sont ainsi détectés par *a3* doivent donc être analysés plus finement pour, d'une part, vérifier que l'analyse WCET est conforme à la réalité et, d'autre part et tel est le cas, proposer des solutions (changement de plateforme, réécriture d'algorithmes moins gourmands en puissance de calcul,...).

8. Conclusion

Nous avons réalisé une étude sur l'applicabilité des outils d'analyse WCET au langage Faust. L'outil *a3* nous a permis d'analyser une large palette de programmes Faust et de valider la pertinence de ce type d'outil pour les langages synchrones. En particulier, nous avons pu détecter automatiquement que certains programmes Faust ne peuvent être traités sur la plateforme OWL qui nous a servi de plateforme de test. Ceci illustre l'importance d'une analyse fine de performances pour les applications audio-numériques.

11 http://www.otawa.fr/

9. Remerciements

Nous remercions la société AbsInt pour la fourniture d'une licence *a3*, ainsi que pour la formation en ligne qui nous a été donnée afin de nous permettre d'utiliser efficacement cet environnement riche. Nous remercions également Yann Orlary (Grame) pour son aide concernant le processus de compilation Faust.

10. Bibliographie

[1] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-Case Execution Time Problem: Overview of Methods and Survey of Tools.* ACM Transactions on Embedded Computing Systems (TECS), Volume 7, Issue 3, April 2008.

[2] Karim Barkati, Pierre Jouvelot. *Synchronous programming in audio processing: A lookup table oscillator case study.* ACM Computing Surveys (CSUR), Volume 46, Issue 2, November 2013.

Annexe : Benchmarking de programmes Faust

Nous présentons ci-après les résultats obtenus sur les six lots d'exécutables. On rappelle que les mesures concernent exclusivement la fonction *computemydsp* avec une unique itération de la boucle principale. La colonne « Cycles » donne la contribution du code placé juste avant la boucle tandis que la valeur valeur « Cumul. cycles » donne la contribution totale de la fonction. On peut donc facilement recalculer d'autres valeurs de WCET pour un nombre d'itération supérieur, ce qui a du sens pour des drivers audio qui fournissent un tableau d'échantillons lors de l'appel de la fonction `computemydsp`.

a. Hard FP

	Hard FP subnormal			
Name	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**
ASREnvelope	246	1176	1,465 us	7,000 us
AtonalSoftHarp	5210	121296	31,012 us	0,722 ms
BandPassFilter	143	106151	0,852 us	0,632 ms
Birds	214	40104	1,274 us	0,239 ms
BlowhistleBottle	3984	87921	23,715 us	0,524 ms
BouncyHarp	4895	110195	29,137 us	0,656 ms
Brass	3852	193070	22,929 us	1,150 ms
ChromaticSoftHarp	5517	138273	32,840 us	0,824 ms
Clarinet	3894	37943	23,179 us	0,226 ms
CMajDryHarp	6944	187566	41,334 us	1,117 ms
CMajFlute	4035	100422	24,018 us	0,598 ms
CMajSoftHarp	5435	138202	32,352 us	0,823 ms
CombFilter	161	1156	0,959 us	6,881 us
Echo	173	606	1,030 us	3,608 us
Flanger	203	105148	1,209 us	0,626 ms
FlappyFlute	3773	149821	22,459 us	0,892 ms
Flute	4028	248364	23,977 us	1,479 ms
Freeverb	138	6625	0,822 us	39,435 us
Granulator	148	5622	0,881 us	33,465 us
HighPassFilter	119	52834	0,709 us	0,315 ms

InstrReverb	3528	34738	21,000 us	0,207 ms
Kisana	1857	77941	11,054 us	0,464 ms
Loop	228	3371	1,358 us	20,066 us
LowPassFilter	127	52788	0,756 us	0,315 ms
Meow	468	222481	2,786 us	1,325 ms
Modulations	4408	563341	26,239 us	3,354 ms
Notch	359	104655	2,137 us	0,623 ms
PeakEqualizer	282	107808	1,679 us	0,642 ms
PentatonicDryHarp	5324	105677	31,691 us	0,630 ms
PentatonicFlute	4041	89060	24,054 us	0,531 ms
PentatonicSoftHarp	5106	120976	30,393 us	0,721 ms
Phaser	210	520600	1,250 us	3,099 ms
Pulsaxophone	3639	355359	21,661 us	2,116 ms
RandomFlute	3608	148852	21,477 us	0,887 ms
RandomRingModulation	151	5341	0,899 us	31,792 us
RandomVibrato	180	109103	1,072 us	0,650 ms
RingModulation	125	663	0,745 us	3,947 us
SAtonalSoftHarp	1596	85702	9,500 us	0,511 ms
SBird	182	25159	1,084 us	0,150 ms
SBlowhistleBottle	209	49660	1,245 us	0,296 ms
SBouncyHarp	1007	72398	5,995 us	0,431 ms
SBrass	366	158451	2,179 us	0,944 ms
SBrassMulti	433	161421	2,578 us	0,961 ms
SCameleonKeyboard	613	230004	3,649 us	1,370 ms
SChromaticSoftHarp	1793	102404	10,673 us	0,610 ms
SChromaticTunedBars	214	46064	1,274 us	0,275 ms
SClarinet	331	2773	1,971 us	16,506 us
SCMajBlowBottle	249	54854	1,483 us	0,327 ms
SCMajDryHarp	3315	151290	19,733 us	0,901 ms
SCMajFlute	270	61530	1,608 us	0,367 ms
SCMajSoftHarp	1793	102771	10,673 us	0,612 ms
SCMajTunedBars	222	58566	1,322 us	0,349 ms
Seaside	3557	141170	21,173 us	0,841 ms
SFlute	479	212800	2,852 us	1,267 ms
SLimiter	127	638	0,756 us	3,798 us
SModulation1	255	209760	1,518 us	1,249 ms
SModulation2	271	106182	1,614 us	0,633 ms
SModulation3	302	210169	1,798 us	1,252 ms
SModulation4	273	210032	1,625 us	1,251 ms
SNoise	80	238	0,477 us	1,417 us
SNoiseburst	137	4552	0,816 us	27,096 us
SNoiseS	184	565	1,096 us	3,364 us
SOscillator	88	104161	0,524 us	0,621 ms
SPentatonicBlowBottle	245	51423	1,459 us	0,307 ms
SPentatonicDryHarp	1728	68635	10,286 us	0,409 ms

SPentatonicFlute	271	50865	1,614 us	0,303 ms
SPentatonicSoftHarp	1559	85671	9,280 us	0,510 ms
SPulsaxophone	168	319837	1,000 us	1,904 ms
SRandomAndHold	168	209338	1,000 us	1,247 ms
SRandomFlute	157	113348	0,935 us	0,675 ms
SRandomFrequencyGenerator	171	105535	1,018 us	0,629 ms
StalactiteHarp	4771	101966	28,399 us	0,607 ms
STibetanBowl	1430	784445	8,512 us	4,670 ms
STinkle	1374	575579	8,179 us	3,427 ms
STunedBar	4775	2096937	28,423 us	12,482 ms
STunedBar3	282	34019	1,679 us	0,203 ms
STunedBar6	462	85864	2,750 us	0,512 ms
SWhistles2	289	635786	1,721 us	3,785 ms
SWhistles3	289	635786	1,721 us	3,785 ms
TibetanBowl	5213	820501	31,030 us	4,884 ms
TibetanBowlMulti	5212	820303	31,024 us	4,883 ms
TunedBars	8533	2132449	50,792 us	12,694 ms
VibratoEnvelope	200	104996	1,191 us	0,625 ms
Volume	128	342	0,762 us	2,036 us
WahWah	182	61611	1,084 us	0,367 ms
Whistles	543	2022830	3,233 us	12,041 ms
WoodenKeyboard	505	216919	3,006 us	1,292 ms
ZitaReverb	157	17732	0,935 us	0,106 ms

b. Soft FP

	Hard FP no subnormal				Speedup (%) vs Hard FP subnormal
Name	Cycles	Cumul. cycles	Time	Cumul. Time	
ASREnvelope	246	1176	1,465 us	7,000 us	0,00
AtonalSoftHarp	5210	121296	31,012 us	0,722 ms	0,00
BandPassFilter	143	106151	0,852 us	0,632 ms	0,00
Birds	214	36144	1,274 us	0,216 ms	9,87
BlowhistleBottle	3984	87921	23,715 us	0,524 ms	0,00
BouncyHarp	4895	110195	29,137 us	0,656 ms	0,00
Brass	3852	193070	22,929 us	1,150 ms	0,00
ChromaticSoftHarp	5517	138273	32,840 us	0,824 ms	0,00
Clarinet	3894	37943	23,179 us	0,226 ms	0,00
CMajDryHarp	6944	187566	41,334 us	1,117 ms	0,00
CMajFlute	4035	100422	24,018 us	0,598 ms	0,00
CMajSoftHarp	5435	138202	32,352 us	0,823 ms	0,00
CombFilter	161	1156	0,959 us	6,881 us	0,00
Echo	173	606	1,030 us	3,608 us	0,00
Flanger	203	105148	1,209 us	0,626 ms	0,00
FlappyFlute	3773	148831	22,459 us	0,886 ms	0,66
Flute	4028	248364	23,977 us	1,479 ms	0,00

Freeverb	138	6625	0,822 us	39,435 us	0,00
Granulator	148	4632	0,881 us	27,572 us	17,61
HighPassFilter	119	52834	0,709 us	0,315 ms	0,00
InstrReverb	3528	34738	21,000 us	0,207 ms	0,00
Kisana	1857	77941	11,054 us	0,464 ms	0,00
Loop	228	3371	1,358 us	20,066 us	0,00
LowPassFilter	127	52788	0,756 us	0,315 ms	0,00
Meow	468	222481	2,786 us	1,325 ms	0,00
Modulations	4408	563341	26,239 us	3,354 ms	0,00
Notch	359	104655	2,137 us	0,623 ms	0,00
PeakEqualizer	282	107808	1,679 us	0,642 ms	0,00
PentatonicDryHarp	5324	105677	31,691 us	0,630 ms	0,00
PentatonicFlute	4041	89060	24,054 us	0,531 ms	0,00
PentatonicSoftHarp	5106	120976	30,393 us	0,721 ms	0,00
Phaser	210	520600	1,250 us	3,099 ms	0,00
Pulsaxophone	3639	354369	21,661 us	2,110 ms	0,28
RandomFlute	3608	147862	21,477 us	0,881 ms	0,67
RandomRingModulation	151	4351	0,899 us	25,899 us	18,54
RandomVibrato	180	108113	1,072 us	0,644 ms	0,91
RingModulation	125	663	0,745 us	3,947 us	0,00
SAtonalSoftHarp	1596	85702	9,500 us	0,511 ms	0,00
SBird	182	21199	1,084 us	0,127 ms	15,74
SBlowhistleBottle	209	49660	1,245 us	0,296 ms	0,00
SBouncyHarp	1007	72398	5,995 us	0,431 ms	0,00
SBrass	366	158451	2,179 us	0,944 ms	0,00
SBrassMulti	433	161421	2,578 us	0,961 ms	0,00
SCameleonKeyboard	613	230004	3,649 us	1,370 ms	0,00
SChromaticSoftHarp	1793	102404	10,673 us	0,610 ms	0,00
SChromaticTunedBars	214	46064	1,274 us	0,275 ms	0,00
SClarinet	331	2773	1,971 us	16,506 us	0,00
SCMajBlowBottle	249	54854	1,483 us	0,327 ms	0,00
SCMajDryHarp	3315	151290	19,733 us	0,901 ms	0,00
SCMajFlute	270	61530	1,608 us	0,367 ms	0,00
SCMajSoftHarp	1793	102771	10,673 us	0,612 ms	0,00
SCMajTunedBars	222	58566	1,322 us	0,349 ms	0,00
Seaside	3557	141170	21,173 us	0,841 ms	0,00
SFlute	479	212800	2,852 us	1,267 ms	0,00
SLimiter	127	638	0,756 us	3,798 us	0,00
SModulation1	255	209760	1,518 us	1,249 ms	0,00
SModulation2	271	106182	1,614 us	0,633 ms	0,00
SModulation3	302	210169	1,798 us	1,252 ms	0,00
SModulation4	273	210032	1,625 us	1,251 ms	0,00
SNoise	80	238	0,477 us	1,417 us	0,00
SNoiseburst	137	3562	0,816 us	21,203 us	21,75
SNoiseS	184	565	1,096 us	3,364 us	0,00

SOscillator	88	104161	0,524 us	0,621 ms	0,00
SPentatonicBlowBottle	245	51423	1,459 us	0,307 ms	0,00
SPentatonicDryHarp	1728	68635	10,286 us	0,409 ms	0,00
SPentatonicFlute	271	50865	1,614 us	0,303 ms	0,00
SPentatonicSoftHarp	1559	85671	9,280 us	0,510 ms	0,00
SPulsaxophone	168	318847	1,000 us	1,898 ms	0,31
SRandomAndHold	168	209338	1,000 us	1,247 ms	0,00
SRandomFlute	157	112358	0,935 us	0,669 ms	0,87
SRandomFrequencyGenerator	171	105535	1,018 us	0,629 ms	0,00
StalactiteHarp	4771	100976	28,399 us	0,602 ms	0,97
STibetanBowl	1430	784445	8,512 us	4,670 ms	0,00
STinkle	1374	575579	8,179 us	3,427 ms	0,00
STunedBar	4775	2096937	28,423 us	12,482 ms	0,00
STunedBar3	282	34019	1,679 us	0,203 ms	0,00
STunedBar6	462	85864	2,750 us	0,512 ms	0,00
SWhistles2	289	635786	1,721 us	3,785 ms	0,00
SWhistles3	289	635786	1,721 us	3,785 ms	0,00
TibetanBowl	5213	820501	31,030 us	4,884 ms	0,00
TibetanBowlMulti	5212	820303	31,024 us	4,883 ms	0,00
TunedBars	8533	2132449	50,792 us	12,694 ms	0,00
VibratoEnvelope	200	104996	1,191 us	0,625 ms	0,00
Volume	128	342	0,762 us	2,036 us	0,00
WahWah	182	61611	1,084 us	0,367 ms	0,00
Whistles	543	2021840	3,233 us	12,035 ms	0,05
WoodenKeyboard	505	216919	3,006 us	1,292 ms	0,00
ZitaReverb	157	17732	0,935 us	0,106 ms	0,00

	Soft FP subnormal				Speedup (%) vs Hard FP subnormal
Name	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	
ASREnvelope	246	1167	1,465 us	6,947 us	0,77
AtonalSoftHarp	4982	118206	29,655 us	0,704 ms	2,55
BandPassFilter	143	105513	0,852 us	0,629 ms	0,60
Birds	213	32300	1,268 us	0,193 ms	19,46
BlowhistleBottle	3932	88002	23,405 us	0,524 ms	-0,09
BouncyHarp	4844	108107	28,834 us	0,644 ms	1,89
Brass	3811	192429	22,685 us	1,146 ms	0,33
ChromaticSoftHarp	5254	134501	31,274 us	0,801 ms	2,73
Clarinet	3855	38287	22,947 us	0,228 ms	-0,91
CMajDryHarp	6679	180665	39,756 us	1,076 ms	3,68
CMajFlute	3992	100638	23,762 us	0,600 ms	-0,22
CMajSoftHarp	5256	134468	31,286 us	0,801 ms	2,70
CombFilter	163	1155	0,971 us	6,875 us	0,09
Echo	173	598	1,030 us	3,560 us	1,32

Flanger	203	104503	1,209 us	0,623 ms	0,61
FlappyFlute	3731	147532	22,209 us	0,879 ms	1,53
Flute	3989	247332	23,745 us	1,473 ms	0,42
Freeverb	139	6580	0,828 us	39,167 us	0,68
Granulator	148	3712	0,881 us	22,096 us	33,97
HighPassFilter	117	52469	0,697 us	0,313 ms	0,69
InstrReverb	3501	35090	20,840 us	0,209 ms	-1,01
Kisana	1693	74403	10,078 us	0,443 ms	4,54
Loop	220	3220	1,310 us	19,167 us	4,48
LowPassFilter	127	52436	0,756 us	0,313 ms	0,67
Meow	468	220696	2,786 us	1,314 ms	0,80
Modulations	4362	560248	25,965 us	3,335 ms	0,55
Notch	343	103931	2,042 us	0,619 ms	0,69
PeakEqualizer	279	106965	1,661 us	0,637 ms	0,78
PentatonicDryHarp	5134	102435	30,560 us	0,610 ms	3,07
PentatonicFlute	3998	89141	23,798 us	0,531 ms	-0,09
PentatonicSoftHarp	4964	117944	29,548 us	0,703 ms	2,51
Phaser	199	517241	1,185 us	3,079 ms	0,65
Pulsaxophone	3606	351772	21,465 us	2,094 ms	1,01
RandomFlute	3577	146641	21,292 us	0,873 ms	1,49
RandomRingModulation	151	3419	0,899 us	20,352 us	35,99
RandomVibrato	180	106512	1,072 us	0,634 ms	2,37
RingModulation	125	663	0,745 us	3,947 us	0,00
SAtonalSoftHarp	1430	82320	8,512 us	0,490 ms	3,95
SBird	182	17427	1,084 us	0,104 ms	30,73
SBlowhistleBottle	211	49527	1,256 us	0,295 ms	0,27
SBouncyHarp	1007	70223	5,995 us	0,418 ms	3,00
SBrass	368	157452	2,191 us	0,938 ms	0,63
SBrassMulti	433	160455	2,578 us	0,956 ms	0,60
SCameleonKeyboard	613	228646	3,649 us	1,361 ms	0,59
SChromaticSoftHarp	1674	98383	9,965 us	0,586 ms	3,93
SChromaticTunedBars	214	46042	1,274 us	0,275 ms	0,05
SClarinet	331	2780	1,971 us	16,548 us	-0,25
SCMajBlowBottle	242	54760	1,441 us	0,326 ms	0,17
SCMajDryHarp	3099	144089	18,447 us	0,858 ms	4,76
SCMajFlute	263	61274	1,566 us	0,365 ms	0,42
SCMajSoftHarp	1674	98750	9,965 us	0,588 ms	3,91
SCMajTunedBars	222	58537	1,322 us	0,349 ms	0,05
Seaside	3513	140860	20,911 us	0,839 ms	0,22
SFlute	479	211458	2,852 us	1,259 ms	0,63
SLimiter	127	624	0,756 us	3,715 us	2,19
SModulation1	259	208400	1,542 us	1,241 ms	0,65
SModulation2	275	105530	1,637 us	0,629 ms	0,61
SModulation3	304	208837	1,810 us	1,244 ms	0,63
SModulation4	277	208716	1,649 us	1,243 ms	0,63

SNoise	80	238	0,477 us	1,417 us	0,00
SNoiseburst	138	2642	0,822 us	15,727 us	41,96
SNoiseS	184	565	1,096 us	3,364 us	0,00
SOscillator	88	103488	0,524 us	0,616 ms	0,65
SPentatonicBlowBottle	238	51339	1,417 us	0,306 ms	0,16
SPentatonicDryHarp	1621	65108	9,649 us	0,388 ms	5,14
SPentatonicFlute	264	50758	1,572 us	0,303 ms	0,21
SPentatonicSoftHarp	1399	82301	8,328 us	0,490 ms	3,93
SPulsaxophone	168	315900	1,000 us	1,881 ms	1,23
SRandomAndHold	168	207966	1,000 us	1,238 ms	0,66
SRandomFlute	159	110775	0,947 us	0,660 ms	2,27
SRandomFrequencyGenerator	173	104832	1,030 us	0,624 ms	0,67
StalactiteHarp	4611	98009	27,447 us	0,584 ms	3,88
STibetanBowl	1430	779413	8,512 us	4,640 ms	0,64
STinkle	1374	571894	8,179 us	3,405 ms	0,64
STunedBar	4775	2083523	28,423 us	12,402 ms	0,64
STunedBar3	282	34019	1,679 us	0,203 ms	0,00
STunedBar6	462	85864	2,750 us	0,512 ms	0,00
SWhistles2	289	631806	1,721 us	3,761 ms	0,63
SWhistles3	289	631806	1,721 us	3,761 ms	0,63
TibetanBowl	5139	815748	30,590 us	4,856 ms	0,58
TibetanBowlMulti	5152	815562	30,667 us	4,855 ms	0,58
TunedBars	8475	2119308	50,447 us	12,615 ms	0,62
VibratoEnvelope	199	104316	1,185 us	0,621 ms	0,65
Volume	128	342	0,762 us	2,036 us	0,00
WahWah	182	60750	1,084 us	0,362 ms	1,40
Whistles	543	2008326	3,233 us	11,955 ms	0,72
WoodenKeyboard	505	215556	3,006 us	1,284 ms	0,63
ZitaReverb	157	17888	0,935 us	0,107 ms	-0,88

c. No FP

	Soft FP no subnormal				Speedup (%) vs Hard FP no subnormal
Name	Cycles	Cumul. cycles	Time	Cumul. Time	
ASREnvelope	246	1167	1,465 us	6,947 us	0,77
AtonalSoftHarp	4982	118206	29,655 us	0,704 ms	2,55
BandPassFilter	143	105513	0,852 us	0,629 ms	0,60
Birds	213	29832	1,268 us	0,178 ms	17,46
BlowhistleBottle	3932	88002	23,405 us	0,524 ms	-0,09
BouncyHarp	4844	108107	28,834 us	0,644 ms	1,89
Brass	3811	192429	22,685 us	1,146 ms	0,33
ChromaticSoftHarp	5254	134501	31,274 us	0,801 ms	2,73
Clarinet	3855	38287	22,947 us	0,228 ms	-0,91
CMajDryHarp	6679	180665	39,756 us	1,076 ms	3,68
CMajFlute	3992	100638	23,762 us	0,600 ms	-0,22

CMajSoftHarp	5256	134468	31,286	us	0,801	ms	2,70
CombFilter	163	1155	0,971	us	6,875	us	0,09
Echo	173	598	1,030	us	3,560	us	1,32
Flanger	203	104503	1,209	us	0,623	ms	0,61
FlappyFlute	3731	146906	22,209	us	0,875	ms	1,29
Flute	3989	247332	23,745	us	1,473	ms	0,42
Freeverb	139	6580	0,828	us	39,167	us	0,68
Granulator	148	3086	0,881	us	18,370	us	33,38
HighPassFilter	117	52469	0,697	us	0,313	ms	0,69
InstrReverb	3501	35090	20,840	us	0,209	ms	-1,01
Kisana	1693	74403	10,078	us	0,443	ms	4,54
Loop	220	3220	1,310	us	19,167	us	4,48
LowPassFilter	127	52436	0,756	us	0,313	ms	0,67
Meow	468	220696	2,786	us	1,314	ms	0,80
Modulations	4362	560248	25,965	us	3,335	ms	0,55
Notch	343	103931	2,042	us	0,619	ms	0,69
PeakEqualizer	279	106965	1,661	us	0,637	ms	0,78
PentatonicDryHarp	5134	102435	30,560	us	0,610	ms	3,07
PentatonicFlute	3998	89141	23,798	us	0,531	ms	-0,09
PentatonicSoftHarp	4964	117944	29,548	us	0,703	ms	2,51
Phaser	199	517241	1,185	us	3,079	ms	0,65
Pulsaxophone	3606	351146	21,465	us	2,091	ms	0,91
RandomFlute	3577	146015	21,292	us	0,870	ms	1,25
RandomRingModulation	151	2793	0,899	us	16,625	us	35,81
RandomVibrato	180	105886	1,072	us	0,631	ms	2,06
RingModulation	125	663	0,745	us	3,947	us	0,00
SAtonalSoftHarp	1430	82320	8,512	us	0,490	ms	3,95
SBird	182	14959	1,084	us	89,042	us	29,44
SBlowhistleBottle	211	49527	1,256	us	0,295	ms	0,27
SBouncyHarp	1007	70223	5,995	us	0,418	ms	3,00
SBrass	368	157452	2,191	us	0,938	ms	0,63
SBrassMulti	433	160455	2,578	us	0,956	ms	0,60
SCameleonKeyboard	613	228646	3,649	us	1,361	ms	0,59
SChromaticSoftHarp	1674	98383	9,965	us	0,586	ms	3,93
SChromaticTunedBars	214	46042	1,274	us	0,275	ms	0,05
SClarinet	331	2780	1,971	us	16,548	us	-0,25
SCMajBlowBottle	242	54760	1,441	us	0,326	ms	0,17
SCMajDryHarp	3099	144089	18,447	us	0,858	ms	4,76
SCMajFlute	263	61274	1,566	us	0,365	ms	0,42
SCMajSoftHarp	1674	98750	9,965	us	0,588	ms	3,91
SCMajTunedBars	222	58537	1,322	us	0,349	ms	0,05
Seaside	3513	140860	20,911	us	0,839	ms	0,22
SFlute	479	211458	2,852	us	1,259	ms	0,63
SLimiter	127	624	0,756	us	3,715	us	2,19
SModulation1	259	208400	1,542	us	1,241	ms	0,65

SModulation2	275	105530	1,637	us	0,629	ms	0,61
SModulation3	304	208837	1,810	us	1,244	ms	0,63
SModulation4	277	208716	1,649	us	1,243	ms	0,63
SNoise	80	238	0,477	us	1,417	us	0,00
SNoiseburst	138	2016	0,822	us	12,000	us	43,40
SNoiseS	184	565	1,096	us	3,364	us	0,00
SOscillator	88	103488	0,524	us	0,616	ms	0,65
SPentatonicBlowBottle	238	51339	1,417	us	0,306	ms	0,16
SPentatonicDryHarp	1621	65108	9,649	us	0,388	ms	5,14
SPentatonicFlute	264	50758	1,572	us	0,303	ms	0,21
SPentatonicSoftHarp	1399	82301	8,328	us	0,490	ms	3,93
SPulsaxophone	168	315274	1,000	us	1,877	ms	1,12
SRandomAndHold	168	207966	1,000	us	1,238	ms	0,66
SRandomFlute	159	110149	0,947	us	0,656	ms	1,97
SRandomFrequencyGenerator	173	104832	1,030	us	0,624	ms	0,67
StalactiteHarp	4611	97383	27,447	us	0,580	ms	3,56
STibetanBowl	1430	779413	8,512	us	4,640	ms	0,64
STinkle	1374	571894	8,179	us	3,405	ms	0,64
STunedBar	4775	2083523	28,423	us	12,402	ms	0,64
STunedBar3	282	34019	1,679	us	0,203	ms	0,00
STunedBar6	462	85864	2,750	us	0,512	ms	0,00
SWhistles2	289	631806	1,721	us	3,761	ms	0,63
SWhistles3	289	631806	1,721	us	3,761	ms	0,63
TibetanBowl	5139	815748	30,590	us	4,856	ms	0,58
TibetanBowlMulti	5152	815562	30,667	us	4,855	ms	0,58
TunedBars	8475	2119308	50,447	us	12,615	ms	0,62
VibratoEnvelope	199	104316	1,185	us	0,621	ms	0,65
Volume	128	342	0,762	us	2,036	us	0,00
WahWah	182	60750	1,084	us	0,362	ms	1,40
Whistles	543	2007700	3,233	us	11,951	ms	0,70
WoodenKeyboard	505	215556	3,006	us	1,284	ms	0,63
ZitaReverb	157	17888	0,935	us	0,107	ms	-0,88

	No FPU subnormal						
Name	**Cycles**	**Cumul. cycles**	**Time**		**Cumul. Time**		**Time / Time with FPU**
ASREnvelope	307	5754	1,828	us	34,250	us	4,9
AtonalSoftHarp	5024	327765	29,905	us	1,951	ms	2,8
BandPassFilter	146	1054290	0,870	us	6,276	ms	10,0
Birds	238	108693	1,417	us	0,647	ms	3,4
BlowhistleBottle	4033	401676	24,006	us	2,391	ms	4,6
BouncyHarp	5386	400754	32,060	us	2,386	ms	3,7
Brass	4140	1723711	24,643	us	10,261	ms	9,0
ChromaticSoftHarp	5241	362735	31,197	us	2,160	ms	2,7
Clarinet	4191	166210	24,947	us	0,990	ms	4,3

CMajDryHarp	6727	397382	40,042	us	2,366	ms	2,2
CMajFlute	4146	426403	24,679	us	2,539	ms	4,2
CMajSoftHarp	5245	362383	31,221	us	2,158	ms	2,7
CombFilter	154	4494	0,917	us	26,750	us	3,9
Echo	196	1773	1,167	us	10,554	us	3,0
Flanger	232	1047696	1,381	us	6,237	ms	10,0
FlappyFlute	4046	1227788	24,084	us	7,309	ms	8,3
Flute	4406	2260039	26,227	us	13,453	ms	9,1
Freeverb	177	27406	1,054	us	0,164	ms	4,2
Granulator	146	12559	0,870	us	74,756	us	3,4
HighPassFilter	118	528481	0,703	us	3,146	ms	10,1
InstrReverb	3838	150079	22,846	us	0,894	ms	4,3
Kisana	1718	137658	10,227	us	0,820	ms	1,9
Loop	248	3773	1,477	us	22,459	us	1,2
LowPassFilter	118	528011	0,703	us	3,143	ms	10,1
Meow	548	2149008	3,262	us	12,792	ms	9,7
Modulations	4771	5401234	28,399	us	32,151	ms	9,6
Notch	368	1043557	2,191	us	6,212	ms	10,0
PeakEqualizer	309	1052922	1,840	us	6,268	ms	9,8
PentatonicDryHarp	5317	262384	31,649	us	1,562	ms	2,6
PentatonicFlute	4147	378289	24,685	us	2,252	ms	4,2
PentatonicSoftHarp	5004	327096	29,786	us	1,947	ms	2,8
Phaser	228	5213716	1,358	us	31,035	ms	10,1
Pulsaxophone	3881	3300368	23,102	us	19,646	ms	9,4
RandomFlute	3873	1224040	23,054	us	7,286	ms	8,3
RandomRingModulation	148	9378	0,881	us	55,822	us	2,7
RandomVibrato	186	1049905	1,108	us	6,250	ms	9,9
RingModulation	143	3206	0,852	us	19,084	us	4,8
SAtonalSoftHarp	1240	176454	7,381	us	1,051	ms	2,1
SBird	191	57299	1,137	us	0,342	ms	3,3
SBlowhistleBottle	285	248503	1,697	us	1,480	ms	5,0
SBouncyHarp	1368	247411	8,143	us	1,473	ms	3,5
SBrass	419	1575045	2,495	us	9,376	ms	10,0
SBrassMulti	535	1593992	3,185	us	9,489	ms	9,9
SCameleonKeyboard	943	2178106	5,614	us	12,965	ms	9,5
SChromaticSoftHarp	1463	211633	8,709	us	1,260	ms	2,2
SChromaticTunedBars	182	195434	1,084	us	1,164	ms	4,2
SClarinet	420	15374	2,500	us	91,512	us	5,5
SCMajBlowBottle	271	277781	1,614	us	1,654	ms	5,1
SCMajDryHarp	2943	245343	17,518	us	1,461	ms	1,7
SCMajFlute	304	271738	1,810	us	1,618	ms	4,4
SCMajSoftHarp	1463	211668	8,709	us	1,260	ms	2,1
SCMajTunedBars	193	206915	1,149	us	1,232	ms	3,5
Seaside	3838	1205995	22,846	us	7,179	ms	8,6
SFlute	598	2108603	3,560	us	12,552	ms	10,0

SLimiter	115	1616	0,685	us	9,620	us	2,6
SModulation1	310	2094243	1,846	us	12,466	ms	10,0
SModulation2	325	1055298	1,935	us	6,282	ms	10,0
SModulation3	353	2096045	2,102	us	12,477	ms	10,0
SModulation4	321	2095531	1,911	us	12,474	ms	10,0
SNoise	95	604	0,566	us	3,596	us	2,5
SNoiseburst	144	6242	0,858	us	37,155	us	2,4
SNoiseS	256	3574	1,524	us	21,274	us	6,3
SOscillator	100	1041701	0,596	us	6,201	ms	10,1
SPentatonicBlowBottle	271	260783	1,614	us	1,553	ms	5,1
SPentatonicDryHarp	1564	109088	9,310	us	0,650	ms	1,7
SPentatonicFlute	325	224478	1,935	us	1,337	ms	4,4
SPentatonicSoftHarp	1207	176504	7,185	us	1,051	ms	2,1
SPulsaxophone	199	3149609	1,185	us	18,748	ms	10,0
SRandomAndHold	188	2086590	1,120	us	12,421	ms	10,0
SRandomFlute	174	1073558	1,036	us	6,391	ms	9,7
SRandomFrequencyGenerator	183	1046037	1,090	us	6,227	ms	10,0
StalactiteHarp	4752	289359	28,286	us	1,723	ms	3,0
STibetanBowl	1742	7836298	10,370	us	46,645	ms	10,1
STinkle	1638	5747199	9,750	us	34,210	ms	10,0
STunedBar	5682	20899632	33,822	us	0,125	s	10,0
STunedBar3	333	128131	1,983	us	0,763	ms	3,8
STunedBar6	576	269973	3,429	us	1,607	ms	3,1
SWhistles2	353	6326756	2,102	us	37,660	ms	10,0
SWhistles3	353	6326756	2,102	us	37,660	ms	10,0
TibetanBowl	5714	7988715	34,012	us	47,552	ms	9,8
TibetanBowlMulti	5721	7987705	34,054	us	47,546	ms	9,8
TunedBars	9548	21051506	56,834	us	0,126	s	9,9
VibratoEnvelope	248	1046459	1,477	us	6,229	ms	10,0
Volume	142	1108	0,846	us	6,596	us	3,2
WahWah	199	558310	1,185	us	3,324	ms	9,2
Whistles	712	20049742	4,239	us	0,120	s	10,0
WoodenKeyboard	762	2111787	4,536	us	12,571	ms	9,8
ZitaReverb	182	78055	1,084	us	0,465	ms	4,4

	No FPU no subnormal				
Name	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	**Time / Time with FPU**
ASREnvelope	307	5754	1,828 us	34,250 us	4,9
AtonalSoftHarp	5024	327765	29,905 us	1,951 ms	2,8
BandPassFilter	146	1054290	0,870 us	6,276 ms	10,0
Birds	238	107485	1,417 us	0,640 ms	3,3
BlowhistleBottle	4033	401676	24,006 us	2,391 ms	4,6
BouncyHarp	5386	400754	32,060 us	2,386 ms	3,7

Brass	4140	1723711	24,643 us	10,261 ms	9,0
ChromaticSoftHarp	5241	362735	31,197 us	2,160 ms	2,7
Clarinet	4191	166210	24,947 us	0,990 ms	4,3
CMajDryHarp	6727	397382	40,042 us	2,366 ms	2,2
CMajFlute	4146	426403	24,679 us	2,539 ms	4,2
CMajSoftHarp	5245	362383	31,221 us	2,158 ms	2,7
CombFilter	154	4494	0,917 us	26,750 us	3,9
Echo	196	1773	1,167 us	10,554 us	3,0
Flanger	232	1047696	1,381 us	6,237 ms	10,0
FlappyFlute	4046	1227502	24,084 us	7,307 ms	8,3
Flute	4406	2260039	26,227 us	13,453 ms	9,1
Freeverb	177	27406	1,054 us	0,164 ms	4,2
Granulator	146	12273	0,870 us	73,054 us	3,3
HighPassFilter	118	528481	0,703 us	3,146 ms	10,1
InstrReverb	3838	150079	22,846 us	0,894 ms	4,3
Kisana	1718	137658	10,227 us	0,820 ms	1,9
Loop	248	3773	1,477 us	22,459 us	1,2
LowPassFilter	118	528011	0,703 us	3,143 ms	10,1
Meow	548	2149008	3,262 us	12,792 ms	9,7
Modulations	4771	5401234	28,399 us	32,151 ms	9,6
Notch	368	1043557	2,191 us	6,212 ms	10,0
PeakEqualizer	309	1052922	1,840 us	6,268 ms	9,8
PentatonicDryHarp	5317	262384	31,649 us	1,562 ms	2,6
PentatonicFlute	4147	378289	24,685 us	2,252 ms	4,2
PentatonicSoftHarp	5004	327096	29,786 us	1,947 ms	2,8
Phaser	228	5213716	1,358 us	31,035 ms	10,1
Pulsaxophone	3881	3300082	23,102 us	19,644 ms	9,4
RandomFlute	3873	1223754	23,054 us	7,285 ms	8,3
RandomRingModulation	148	9092	0,881 us	54,120 us	2,7
RandomVibrato	186	1049619	1,108 us	6,248 ms	9,9
RingModulation	143	3206	0,852 us	19,084 us	4,8
SAtonalSoftHarp	1240	176454	7,381 us	1,051 ms	2,1
SBird	191	56091	1,137 us	0,334 ms	3,2
SBlowhistleBottle	285	248503	1,697 us	1,480 ms	5,0
SBouncyHarp	1368	247411	8,143 us	1,473 ms	3,5
SBrass	419	1575045	2,495 us	9,376 ms	10,0
SBrassMulti	535	1593992	3,185 us	9,489 ms	9,9
SCameleonKeyboard	943	2178106	5,614 us	12,965 ms	9,5
SChromaticSoftHarp	1463	211633	8,709 us	1,260 ms	2,2
SChromaticTunedBars	182	195434	1,084 us	1,164 ms	4,2
SClarinet	420	15374	2,500 us	91,512 us	5,5
SCMajBlowBottle	271	277781	1,614 us	1,654 ms	5,1
SCMajDryHarp	2943	245343	17,518 us	1,461 ms	1,7
SCMajFlute	304	271738	1,810 us	1,618 ms	4,4
SCMajSoftHarp	1463	211668	8,709 us	1,260 ms	2,1

SCMajTunedBars	193	206915	1,149	us	1,232	ms	3,5
Seaside	3838	1205995	22,846	us	7,179	ms	8,6
SFlute	598	2108603	3,560	us	12,552	ms	10,0
SLimiter	115	1616	0,685	us	9,620	us	2,6
SModulation1	310	2094243	1,846	us	12,466	ms	10,0
SModulation2	325	1055298	1,935	us	6,282	ms	10,0
SModulation3	353	2096045	2,102	us	12,477	ms	10,0
SModulation4	321	2095531	1,911	us	12,474	ms	10,0
SNoise	95	604	0,566	us	3,596	us	2,5
SNoiseburst	144	5956	0,858	us	35,453	us	2,3
SNoiseS	256	3574	1,524	us	21,274	us	6,3
SOscillator	100	1041701	0,596	us	6,201	ms	10,1
SPentatonicBlowBottle	271	260783	1,614	us	1,553	ms	5,1
SPentatonicDryHarp	1564	109088	9,310	us	0,650	ms	1,7
SPentatonicFlute	325	224478	1,935	us	1,337	ms	4,4
SPentatonicSoftHarp	1207	176504	7,185	us	1,051	ms	2,1
SPulsaxophone	199	3149323	1,185	us	18,746	ms	10,0
SRandomAndHold	188	2086590	1,120	us	12,421	ms	10,0
SRandomFlute	174	1073272	1,036	us	6,389	ms	9,7
SRandomFrequencyGenerator	183	1046037	1,090	us	6,227	ms	10,0
StalactiteHarp	4752	289073	28,286	us	1,721	ms	2,9
STibetanBowl	1742	7836298	10,370	us	46,645	ms	10,1
STinkle	1638	5747199	9,750	us	34,210	ms	10,0
STunedBar	5682	20899632	33,822	us	0,125	s	10,0
STunedBar3	333	128131	1,983	us	0,763	ms	3,8
STunedBar6	576	269973	3,429	us	1,607	ms	3,1
SWhistles2	353	6326756	2,102	us	37,660	ms	10,0
SWhistles3	353	6326756	2,102	us	37,660	ms	10,0
TibetanBowl	5714	7988715	34,012	us	47,552	ms	9,8
TibetanBowlMulti	5721	7987705	34,054	us	47,546	ms	9,8
TunedBars	9548	21051506	56,834	us	0,126	s	9,9
VibratoEnvelope	248	1046459	1,477	us	6,229	ms	10,0
Volume	142	1108	0,846	us	6,596	us	3,2
WahWah	199	558310	1,185	us	3,324	ms	9,2
Whistles	712	20049456	4,239	us	0,120	s	10,0
WoodenKeyboard	762	2111787	4,536	us	12,571	ms	9,8
ZitaReverb	182	78055	1,084	us	0,465	ms	4,4