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.

1 http://faust.grame.fr/

2. Cadre de l'étude

Classiquement, un programme Faust est compilé vers un code source C++. Ce dernier se divise en trois parties, dont deux sont importantes pour nous ici :
1- le traitement audio ;
2- l'affichage de l'interface homme-machine.

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].
Le code C++ est enfin compilé sous forme d'un exécutable au format natif de l'architecture cible.

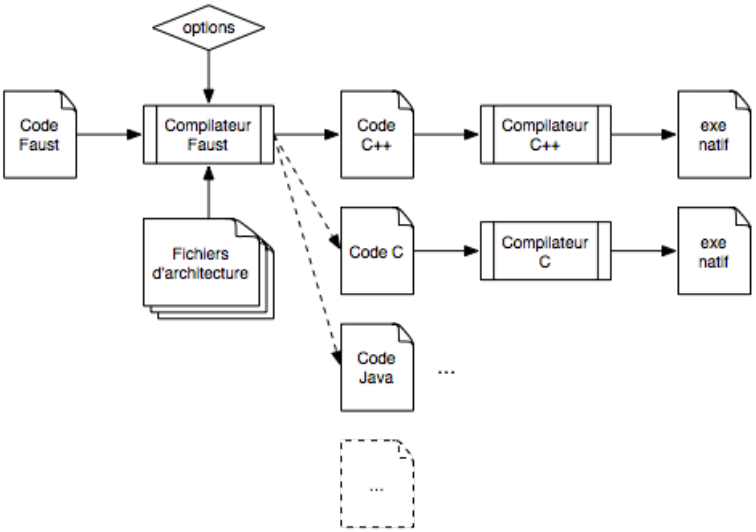

Illustration 1: Chaine de compilation Faust

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].

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

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 :

```
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[9].

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[10] – une version offi-

7 Digital Signal Processing
8 https://pypi.python.org/pypi/pycparser
9 Une autre approche possible serait de définir une architecture spécifique au WCET dans le compilateur Faust. Dans le cas d'une utilisation récurrente de l'outil a3, ce serait sans doute la méthode à privilégier.
10 https://developer.arm.com/open-source/gnu-toolchain/gnu-rm

cielle, maintenue par ARM. Pour les versions avec flottants logiciels, nous avons utilisé le générateur de chaînes de compilation *croostool-NG*[11] 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>
   </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 confi-

11 https://crosstool-ng.github.io

guration 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’interface 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'exécute sensiblement plus lentement dans le second cas, malgré l'option *hard* (138 cycles contre 129 pour la version *softfp*).

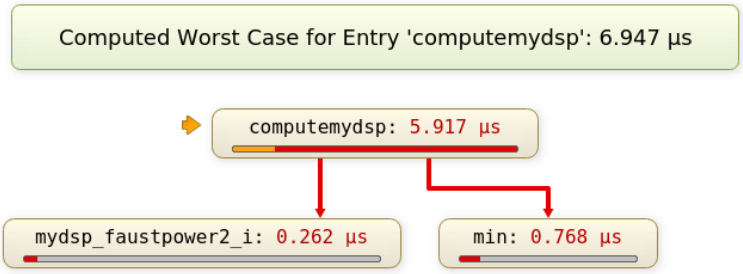

Illustration 2: Analyse en mode `softfp`

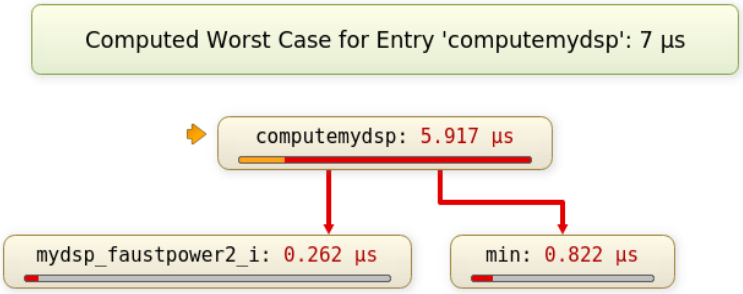

Illustration 3: Analyse en mode hard

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 comparai-

son. 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[12], 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,...).

12http://www.otawa.fr/

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.

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 et sa relecture d'une version précédente de ce document. Ce travail a été partiellement financé par le projet ANR FEEVER.

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.* 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 pour une itération. 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` ; ainsi, nous donnons les temps cumulés par bloc de 256 échantillons, et la charge « Load » afférente (en pourcentage d'utilisation du CPU).

Bien évidemment, il convient de se souvenir, en analysant les résultats présentés dans cette annexe, que ceux-ci sont fondés sur des analyses statiques, qui fournissent donc des temps d'exécution dans le pire cas.

a. Hard FP

	Hard FP subnormal					
Name	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	**Cumul. T block 256 (µs)**	**Load (block 256)**
ASREnvelope	246	1176	1,465 us	7,000 us	5,541	0,244
AtonalSoftHarp	5210	121296	31,012 us	0,722 ms	691,109	30,478
BandPassFilter	143	106151	0,852 us	0,632 ms	631,151	27,834

Birds	214	40104	1,274 us	0,239 ms	237,731	10,484
BlowhistleBottle	3984	87921	23,715 us	0,524 ms	500,378	22,067
BouncyHarp	4895	110195	29,137 us	0,656 ms	626,977	27,650
Brass	3852	193070	22,929 us	1,150 ms	1127,161	49,708
ChromaticSoftHarp	5517	138273	32,840 us	0,824 ms	791,288	34,896
Clarinet	3894	37943	23,179 us	0,226 ms	202,912	8,948
CMajDryHarp	6944	187566	41,334 us	1,117 ms	1075,827	47,444
CMajFlute	4035	100422	24,018 us	0,598 ms	574,076	25,317
CMajSoftHarp	5435	138202	32,352 us	0,823 ms	790,774	34,873
CombFilter	161	1156	0,959 us	6,881 us	5,926	0,261
Echo	173	606	1,030 us	3,608 us	2,582	0,114
Flanger	203	105148	1,209 us	0,626 ms	624,796	27,553
FlappyFlute	3773	149821	22,459 us	0,892 ms	869,629	38,351
Flute	4028	248364	23,977 us	1,479 ms	1455,117	64,171
Freeverb	138	6625	0,822 us	39,435 us	38,616	1,703
Granulator	148	5622	0,881 us	33,465 us	32,587	1,437
HighPassFilter	119	52834	0,709 us	0,315 ms	314,294	13,860
InstrReverb	3528	34738	21,000 us	0,207 ms	186,082	8,206
Kisana	1857	77941	11,054 us	0,464 ms	452,989	19,977
Loop	228	3371	1,358 us	20,066 us	18,713	0,825
LowPassFilter	127	52788	0,756 us	0,315 ms	314,247	13,858
Meow	468	222481	2,786 us	1,325 ms	1322,225	58,310
Modulations	4408	563341	26,239 us	3,354 ms	3327,863	146,759
Notch	359	104655	2,137 us	0,623 ms	620,871	27,380
PeakEqualizer	282	107808	1,679 us	0,642 ms	640,328	28,238
PentatonicDryHarp	5324	105677	31,691 us	0,630 ms	598,433	26,391
PentatonicFlute	4041	89060	24,054 us	0,531 ms	507,040	22,360
PentatonicSoftHarp	5106	120976	30,393 us	0,721 ms	690,726	30,461
Phaser	210	520600	1,250 us	3,099 ms	3097,755	136,611
Pulsaxophone	3639	355359	21,661 us	2,116 ms	2094,424	92,364
RandomFlute	3608	148852	21,477 us	0,887 ms	865,607	38,173
RandomRingModulation	151	5341	0,899 us	31,792 us	30,897	1,363
RandomVibrato	180	109103	1,072 us	0,650 ms	648,932	28,618
RingModulation	125	663	0,745 us	3,947 us	3,205	0,141
SAtonalSoftHarp	1596	85702	9,500 us	0,511 ms	501,537	22,118
SBird	182	25159	1,084 us	0,150 ms	148,920	6,567
SBlowhistleBottle	209	49660	1,245 us	0,296 ms	294,760	12,999
SBouncyHarp	1007	72398	5,995 us	0,431 ms	425,028	18,744
SBrass	366	158451	2,179 us	0,944 ms	941,830	41,535
SBrassMulti	433	161421	2,578 us	0,961 ms	958,432	42,267
SCameleonKeyboard	613	230004	3,649 us	1,370 ms	1366,365	60,257
SChromaticSoftHarp	1793	102404	10,673 us	0,610 ms	599,369	26,432
SChromaticTunedBars	214	46064	1,274 us	0,275 ms	273,731	12,072
SClarinet	331	2773	1,971 us	16,506 us	14,543	0,641
SCMajBlowBottle	249	54854	1,483 us	0,327 ms	325,523	14,356

SCMajDryHarp	3315	151290	19,733 us	0,901 ms	881,344	38,867
SCMajFlute	270	61530	1,608 us	0,367 ms	365,398	16,114
SCMajSoftHarp	1793	102771	10,673 us	0,612 ms	601,369	26,520
SCMajTunedBars	222	58566	1,322 us	0,349 ms	347,683	15,333
Seaside	3557	141170	21,173 us	0,841 ms	819,910	36,158
SFlute	479	212800	2,852 us	1,267 ms	1264,159	55,749
SLimiter	127	638	0,756 us	3,798 us	3,045	0,134
SModulation1	255	209760	1,518 us	1,249 ms	1247,488	55,014
SModulation2	271	106182	1,614 us	0,633 ms	631,392	27,844
SModulation3	302	210169	1,798 us	1,252 ms	1250,209	55,134
SModulation4	273	210032	1,625 us	1,251 ms	1249,381	55,098
SNoise	80	238	0,477 us	1,417 us	0,942	0,042
SNoiseburst	137	4552	0,816 us	27,096 us	26,283	1,159
SNoiseS	184	565	1,096 us	3,364 us	2,272	0,100
SOscillator	88	104161	0,524 us	0,621 ms	620,478	27,363
SPentatonicBlowBottle	245	51423	1,459 us	0,307 ms	305,547	13,475
SPentatonicDryHarp	1728	68635	10,286 us	0,409 ms	398,754	17,585
SPentatonicFlute	271	50865	1,614 us	0,303 ms	301,392	13,291
SPentatonicSoftHarp	1559	85671	9,280 us	0,510 ms	500,756	22,083
SPulsaxophone	168	319837	1,000 us	1,904 ms	1903,004	83,922
SRandomAndHold	168	209338	1,000 us	1,247 ms	1246,004	54,949
SRandomFlute	157	113348	0,935 us	0,675 ms	674,069	29,726
SRandomFrequencyGenerator	171	105535	1,018 us	0,629 ms	627,986	27,694
StalactiteHarp	4771	101966	28,399 us	0,607 ms	578,712	25,521
STibetanBowl	1430	784445	8,512 us	4,670 ms	4661,521	205,573
STinkle	1374	575579	8,179 us	3,427 ms	3418,853	150,771
STunedBar	4775	2096937	28,423 us	12,482 ms	12453,688	549,208
STunedBar3	282	34019	1,679 us	0,203 ms	201,328	8,879
STunedBar6	462	85864	2,750 us	0,512 ms	509,261	22,458
SWhistles2	289	635786	1,721 us	3,785 ms	3783,286	166,843
SWhistles3	289	635786	1,721 us	3,785 ms	3783,286	166,843
TibetanBowl	5213	820501	31,030 us	4,884 ms	4853,091	214,021
TibetanBowlMulti	5212	820303	31,024 us	4,883 ms	4852,097	213,977
TunedBars	8533	2132449	50,792 us	12,694 ms	12643,406	557,574
VibratoEnvelope	200	104996	1,191 us	0,625 ms	623,814	27,510
Volume	128	342	0,762 us	2,036 us	1,277	0,056
WahWah	182	61611	1,084 us	0,367 ms	365,920	16,137
Whistles	543	2022830	3,233 us	12,041 ms	12037,780	530,866
WoodenKeyboard	505	216919	3,006 us	1,292 ms	1289,006	56,845
ZitaReverb	157	17732	0,935 us	0,106 ms	105,069	4,634

Dans le tableau suivant, « SU » indique l'accélération (*speedup*) entre la version « Hard FP subnormal » et celle sans nombres dénormalisés (« no subnormal »), avec gestion des flottants identique à la version précédente (gestion matérielle des nombres flottants).

	Hard FP no subnormal					
SU%	Cycles	Cumul. cycles	Time	Cumul. Time	Cumul. T block 256 (µs)	Load (block 256)
0,00	246	1176	1,465 us	7,000 us	5,541	0,244
0,00	5210	121296	31,012 us	0,722 ms	691,109	30,478
0,00	143	106151	0,852 us	0,632 ms	631,151	27,834
9,87	214	36144	1,274 us	0,216 ms	214,731	9,470
0,00	3984	87921	23,715 us	0,524 ms	500,378	22,067
0,00	4895	110195	29,137 us	0,656 ms	626,977	27,650
0,00	3852	193070	22,929 us	1,150 ms	1127,161	49,708
0,00	5517	138273	32,840 us	0,824 ms	791,288	34,896
0,00	3894	37943	23,179 us	0,226 ms	202,912	8,948
0,00	6944	187566	41,334 us	1,117 ms	1075,827	47,444
0,00	4035	100422	24,018 us	0,598 ms	574,076	25,317
0,00	5435	138202	32,352 us	0,823 ms	790,774	34,873
0,00	161	1156	0,959 us	6,881 us	5,926	0,261
0,00	173	606	1,030 us	3,608 us	2,582	0,114
0,00	203	105148	1,209 us	0,626 ms	624,796	27,553
0,66	3773	148831	22,459 us	0,886 ms	863,629	38,086
0,00	4028	248364	23,977 us	1,479 ms	1455,117	64,171
0,00	138	6625	0,822 us	39,435 us	38,616	1,703
17,61	148	4632	0,881 us	27,572 us	26,694	1,177
0,00	119	52834	0,709 us	0,315 ms	314,294	13,860
0,00	3528	34738	21,000 us	0,207 ms	186,082	8,206
0,00	1857	77941	11,054 us	0,464 ms	452,989	19,977
0,00	228	3371	1,358 us	20,066 us	18,713	0,825
0,00	127	52788	0,756 us	0,315 ms	314,247	13,858
0,00	468	222481	2,786 us	1,325 ms	1322,225	58,310
0,00	4408	563341	26,239 us	3,354 ms	3327,863	146,759
0,00	359	104655	2,137 us	0,623 ms	620,871	27,380
0,00	282	107808	1,679 us	0,642 ms	640,328	28,238
0,00	5324	105677	31,691 us	0,630 ms	598,433	26,391
0,00	4041	89060	24,054 us	0,531 ms	507,040	22,360
0,00	5106	120976	30,393 us	0,721 ms	690,726	30,461
0,00	210	520600	1,250 us	3,099 ms	3097,755	136,611
0,28	3639	354369	21,661 us	2,110 ms	2088,424	92,099
0,67	3608	147862	21,477 us	0,881 ms	859,607	37,909
18,54	151	4351	0,899 us	25,899 us	25,004	1,103
0,91	180	108113	1,072 us	0,644 ms	642,932	28,353
0,00	125	663	0,745 us	3,947 us	3,205	0,141
0,00	1596	85702	9,500 us	0,511 ms	501,537	22,118
15,74	182	21199	1,084 us	0,127 ms	125,920	5,553
0,00	209	49660	1,245 us	0,296 ms	294,760	12,999
0,00	1007	72398	5,995 us	0,431 ms	425,028	18,744

0,00	366	158451	2,179 us	0,944 ms	941,830	41,535
0,00	433	161421	2,578 us	0,961 ms	958,432	42,267
0,00	613	230004	3,649 us	1,370 ms	1366,365	60,257
0,00	1793	102404	10,673 us	0,610 ms	599,369	26,432
0,00	214	46064	1,274 us	0,275 ms	273,731	12,072
0,00	331	2773	1,971 us	16,506 us	14,543	0,641
0,00	249	54854	1,483 us	0,327 ms	325,523	14,356
0,00	3315	151290	19,733 us	0,901 ms	881,344	38,867
0,00	270	61530	1,608 us	0,367 ms	365,398	16,114
0,00	1793	102771	10,673 us	0,612 ms	601,369	26,520
0,00	222	58566	1,322 us	0,349 ms	347,683	15,333
0,00	3557	141170	21,173 us	0,841 ms	819,910	36,158
0,00	479	212800	2,852 us	1,267 ms	1264,159	55,749
0,00	127	638	0,756 us	3,798 us	3,045	0,134
0,00	255	209760	1,518 us	1,249 ms	1247,488	55,014
0,00	271	106182	1,614 us	0,633 ms	631,392	27,844
0,00	302	210169	1,798 us	1,252 ms	1250,209	55,134
0,00	273	210032	1,625 us	1,251 ms	1249,381	55,098
0,00	80	238	0,477 us	1,417 us	0,942	0,042
21,75	137	3562	0,816 us	21,203 us	20,390	0,899
0,00	184	565	1,096 us	3,364 us	2,272	0,100
0,00	88	104161	0,524 us	0,621 ms	620,478	27,363
0,00	245	51423	1,459 us	0,307 ms	305,547	13,475
0,00	1728	68635	10,286 us	0,409 ms	398,754	17,585
0,00	271	50865	1,614 us	0,303 ms	301,392	13,291
0,00	1559	85671	9,280 us	0,510 ms	500,756	22,083
0,31	168	318847	1,000 us	1,898 ms	1897,004	83,658
0,00	168	209338	1,000 us	1,247 ms	1246,004	54,949
0,87	157	112358	0,935 us	0,669 ms	668,069	29,462
0,00	171	105535	1,018 us	0,629 ms	627,986	27,694
0,97	4771	100976	28,399 us	0,602 ms	573,712	25,301
0,00	1430	784445	8,512 us	4,670 ms	4661,521	205,573
0,00	1374	575579	8,179 us	3,427 ms	3418,853	150,771
0,00	4775	2096937	28,423 us	12,482 ms	12453,688	549,208
0,00	282	34019	1,679 us	0,203 ms	201,328	8,879
0,00	462	85864	2,750 us	0,512 ms	509,261	22,458
0,00	289	635786	1,721 us	3,785 ms	3783,286	166,843
0,00	289	635786	1,721 us	3,785 ms	3783,286	166,843
0,00	5213	820501	31,030 us	4,884 ms	4853,091	214,021
0,00	5212	820303	31,024 us	4,883 ms	4852,097	213,977
0,00	8533	2132449	50,792 us	12,694 ms	12643,406	557,574
0,00	200	104996	1,191 us	0,625 ms	623,814	27,510
0,00	128	342	0,762 us	2,036 us	1,277	0,056
0,00	182	61611	1,084 us	0,367 ms	365,920	16,137
0,05	543	2021840	3,233 us	12,035 ms	12031,780	530,601

0,00	505	216919	3,006 us	1,292 ms	1289,006	56,845
0,00	157	17732	0,935 us	0,106 ms	105,069	4,634

b. Soft FP

Dans le tableau suivant, « SU » indique l'accélération (*speedup*) entre la version « Hard FP subnormal » et celle gérant les flottants en mode « soft fp » (utilisation des registres classiques et non des registres dédiés aux flottants) tout en prenant en compte la dénormalisation.

	Soft FP subnormal					
SU%	Cycles	Cumul. cycles	Time	Cumul. Time	Cumul. T block 256 (µs)	Load (block 256)
0,77	246	1167	1,465 us	6,947 us	5,488	0,242
2,55	4982	118206	29,655 us	0,704 ms	674,461	29,744
0,60	143	105513	0,852 us	0,629 ms	628,151	27,701
19,46	213	32300	1,268 us	0,193 ms	191,737	8,456
-0,09	3932	88002	23,405 us	0,524 ms	500,686	22,080
1,89	4844	108107	28,834 us	0,644 ms	615,279	27,134
0,33	3811	192429	22,685 us	1,146 ms	1123,404	49,542
2,73	5254	134501	31,274 us	0,801 ms	769,848	33,950
-0,91	3855	38287	22,947 us	0,228 ms	205,143	9,047
3,68	6679	180665	39,756 us	1,076 ms	1036,399	45,705
-0,22	3992	100638	23,762 us	0,600 ms	576,331	25,416
2,70	5256	134468	31,286 us	0,801 ms	769,836	33,950
0,09	163	1155	0,971 us	6,875 us	5,908	0,261
1,32	173	598	1,030 us	3,560 us	2,534	0,112
0,61	203	104503	1,209 us	0,623 ms	621,796	27,421
1,53	3731	147532	22,209 us	0,879 ms	856,878	37,788
0,42	3989	247332	23,745 us	1,473 ms	1449,348	63,916
0,68	139	6580	0,828 us	39,167 us	38,342	1,691
33,97	148	3712	0,881 us	22,096 us	21,218	0,936
0,69	117	52469	0,697 us	0,313 ms	312,306	13,773
-1,01	3501	35090	20,840 us	0,209 ms	188,241	8,301
4,54	1693	74403	10,078 us	0,443 ms	432,961	19,094
4,48	220	3220	1,310 us	19,167 us	17,862	0,788
0,67	127	52436	0,756 us	0,313 ms	312,247	13,770
0,80	468	220696	2,786 us	1,314 ms	1311,225	57,825
0,55	4362	560248	25,965 us	3,335 ms	3309,136	145,933
0,69	343	103931	2,042 us	0,619 ms	616,966	27,208
0,78	279	106965	1,661 us	0,637 ms	635,345	28,019
3,07	5134	102435	30,560 us	0,610 ms	579,559	25,559
-0,09	3998	89141	23,798 us	0,531 ms	507,295	22,372
2,51	4964	117944	29,548 us	0,703 ms	673,567	29,704
0,65	199	517241	1,185 us	3,079 ms	3077,820	135,732
1,01	3606	351772	21,465 us	2,094 ms	2072,619	91,402

1,49	3577	146641	21,292 us	0,873 ms	851,791	37,564
35,99	151	3419	0,899 us	20,352 us	19,457	0,858
2,37	180	106512	1,072 us	0,634 ms	632,932	27,912
0,00	125	663	0,745 us	3,947 us	3,205	0,141
3,95	1430	82320	8,512 us	0,490 ms	481,521	21,235
30,73	182	17427	1,084 us	0,104 ms	102,920	4,539
0,27	211	49527	1,256 us	0,295 ms	293,749	12,954
3,00	1007	70223	5,995 us	0,418 ms	412,028	18,170
0,63	368	157452	2,191 us	0,938 ms	935,818	41,270
0,60	433	160455	2,578 us	0,956 ms	953,432	42,046
0,59	613	228646	3,649 us	1,361 ms	1357,365	59,860
3,93	1674	98383	9,965 us	0,586 ms	576,074	25,405
0,05	214	46042	1,274 us	0,275 ms	273,731	12,072
-0,25	331	2780	1,971 us	16,548 us	14,585	0,643
0,17	242	54760	1,441 us	0,326 ms	324,565	14,313
4,76	3099	144089	18,447 us	0,858 ms	839,625	37,027
0,42	263	61274	1,566 us	0,365 ms	363,440	16,028
3,91	1674	98750	9,965 us	0,588 ms	578,074	25,493
0,05	222	58537	1,322 us	0,349 ms	347,683	15,333
0,22	3513	140860	20,911 us	0,839 ms	818,171	36,081
0,63	479	211458	2,852 us	1,259 ms	1256,159	55,397
2,19	127	624	0,756 us	3,715 us	2,962	0,131
0,65	259	208400	1,542 us	1,241 ms	1239,464	54,660
0,61	275	105530	1,637 us	0,629 ms	627,369	27,667
0,63	304	208837	1,810 us	1,244 ms	1242,197	54,781
0,63	277	208716	1,649 us	1,243 ms	1241,357	54,744
0,00	80	238	0,477 us	1,417 us	0,942	0,042
41,96	138	2642	0,822 us	15,727 us	14,908	0,657
0,00	184	565	1,096 us	3,364 us	2,272	0,100
0,65	88	103488	0,524 us	0,616 ms	615,478	27,143
0,16	238	51339	1,417 us	0,306 ms	304,589	13,432
5,14	1621	65108	9,649 us	0,388 ms	378,389	16,687
0,21	264	50758	1,572 us	0,303 ms	301,434	13,293
3,93	1399	82301	8,328 us	0,490 ms	481,705	21,243
1,23	168	315900	1,000 us	1,881 ms	1880,004	82,908
0,66	168	207966	1,000 us	1,238 ms	1237,004	54,552
2,27	159	110775	0,947 us	0,660 ms	659,057	29,064
0,67	173	104832	1,030 us	0,624 ms	622,974	27,473
3,88	4611	98009	27,447 us	0,584 ms	556,660	24,549
0,64	1430	779413	8,512 us	4,640 ms	4631,521	204,250
0,64	1374	571894	8,179 us	3,405 ms	3396,853	149,801
0,64	4775	2083523	28,423 us	12,402 ms	12373,688	545,680
0,00	282	34019	1,679 us	0,203 ms	201,328	8,879
0,00	462	85864	2,750 us	0,512 ms	509,261	22,458
0,63	289	631806	1,721 us	3,761 ms	3759,286	165,785

0,63	289	631806	1,721 us	3,761 ms	3759,286	165,785
0,58	5139	815748	30,590 us	4,856 ms	4825,529	212,806
0,58	5152	815562	30,667 us	4,855 ms	4824,453	212,758
0,62	8475	2119308	50,447 us	12,615 ms	12564,750	554,105
0,65	199	104316	1,185 us	0,621 ms	619,820	27,334
0,00	128	342	0,762 us	2,036 us	1,277	0,056
1,40	182	60750	1,084 us	0,362 ms	360,920	15,917
0,72	543	2008326	3,233 us	11,955 ms	11951,780	527,073
0,63	505	215556	3,006 us	1,284 ms	1281,006	56,492
-0,88	157	17888	0,935 us	0,107 ms	106,069	4,678

Dans le tableau suivant, « SU » indique l'accélération (*speedup*) entre la version « Hard FP subnormal » et celle gérant les flottants en mode « soft fp » (utilisation des registres classiques et non des registres dédiés aux flottants) sans prendre en compte la dénormalisation.

	Soft FP no subnormal					
SU %	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	**Cumul. T block 256 (µs)**	**Load (block 256)**
0,77	246	1167	1,465 us	6,947 us	5,488	0,242
2,55	4982	118206	29,655 us	0,704 ms	674,461	29,744
0,60	143	105513	0,852 us	0,629 ms	628,151	27,701
17,46	213	29832	1,268 us	0,178 ms	176,737	7,794
-0,09	3932	88002	23,405 us	0,524 ms	500,686	22,080
1,89	4844	108107	28,834 us	0,644 ms	615,279	27,134
0,33	3811	192429	22,685 us	1,146 ms	1123,404	49,542
2,73	5254	134501	31,274 us	0,801 ms	769,848	33,950
-0,91	3855	38287	22,947 us	0,228 ms	205,143	9,047
3,68	6679	180665	39,756 us	1,076 ms	1036,399	45,705
-0,22	3992	100638	23,762 us	0,600 ms	576,331	25,416
2,70	5256	134468	31,286 us	0,801 ms	769,836	33,950
0,09	163	1155	0,971 us	6,875 us	5,908	0,261
1,32	173	598	1,030 us	3,560 us	2,534	0,112
0,61	203	104503	1,209 us	0,623 ms	621,796	27,421
1,29	3731	146906	22,209 us	0,875 ms	852,878	37,612
0,42	3989	247332	23,745 us	1,473 ms	1449,348	63,916
0,68	139	6580	0,828 us	39,167 us	38,342	1,691
33,38	148	3086	0,881 us	18,370 us	17,492	0,771
0,69	117	52469	0,697 us	0,313 ms	312,306	13,773
-1,01	3501	35090	20,840 us	0,209 ms	188,241	8,301
4,54	1693	74403	10,078 us	0,443 ms	432,961	19,094
4,48	220	3220	1,310 us	19,167 us	17,862	0,788
0,67	127	52436	0,756 us	0,313 ms	312,247	13,770
0,80	468	220696	2,786 us	1,314 ms	1311,225	57,825
0,55	4362	560248	25,965 us	3,335 ms	3309,136	145,933
0,69	343	103931	2,042 us	0,619 ms	616,966	27,208

0,78	279	106965	1,661 us	0,637 ms	635,345	28,019
3,07	5134	102435	30,560 us	0,610 ms	579,559	25,559
-0,09	3998	89141	23,798 us	0,531 ms	507,295	22,372
2,51	4964	117944	29,548 us	0,703 ms	673,567	29,704
0,65	199	517241	1,185 us	3,079 ms	3077,820	135,732
0,91	3606	351146	21,465 us	2,091 ms	2069,619	91,270
1,25	3577	146015	21,292 us	0,870 ms	848,791	37,432
35,81	151	2793	0,899 us	16,625 us	15,730	0,694
2,06	180	105886	1,072 us	0,631 ms	629,932	27,780
0,00	125	663	0,745 us	3,947 us	3,205	0,141
3,95	1430	82320	8,512 us	0,490 ms	481,521	21,235
29,44	182	14959	1,084 us	89,042 us	87,962	3,879
0,27	211	49527	1,256 us	0,295 ms	293,749	12,954
3,00	1007	70223	5,995 us	0,418 ms	412,028	18,170
0,63	368	157452	2,191 us	0,938 ms	935,818	41,270
0,60	433	160455	2,578 us	0,956 ms	953,432	42,046
0,59	613	228646	3,649 us	1,361 ms	1357,365	59,860
3,93	1674	98383	9,965 us	0,586 ms	576,074	25,405
0,05	214	46042	1,274 us	0,275 ms	273,731	12,072
-0,25	331	2780	1,971 us	16,548 us	14,585	0,643
0,17	242	54760	1,441 us	0,326 ms	324,565	14,313
4,76	3099	144089	18,447 us	0,858 ms	839,625	37,027
0,42	263	61274	1,566 us	0,365 ms	363,440	16,028
3,91	1674	98750	9,965 us	0,588 ms	578,074	25,493
0,05	222	58537	1,322 us	0,349 ms	347,683	15,333
0,22	3513	140860	20,911 us	0,839 ms	818,171	36,081
0,63	479	211458	2,852 us	1,259 ms	1256,159	55,397
2,19	127	624	0,756 us	3,715 us	2,962	0,131
0,65	259	208400	1,542 us	1,241 ms	1239,464	54,660
0,61	275	105530	1,637 us	0,629 ms	627,369	27,667
0,63	304	208837	1,810 us	1,244 ms	1242,197	54,781
0,63	277	208716	1,649 us	1,243 ms	1241,357	54,744
0,00	80	238	0,477 us	1,417 us	0,942	0,042
43,40	138	2016	0,822 us	12,000 us	11,181	0,493
0,00	184	565	1,096 us	3,364 us	2,272	0,100
0,65	88	103488	0,524 us	0,616 ms	615,478	27,143
0,16	238	51339	1,417 us	0,306 ms	304,589	13,432
5,14	1621	65108	9,649 us	0,388 ms	378,389	16,687
0,21	264	50758	1,572 us	0,303 ms	301,434	13,293
3,93	1399	82301	8,328 us	0,490 ms	481,705	21,243
1,12	168	315274	1,000 us	1,877 ms	1876,004	82,732
0,66	168	207966	1,000 us	1,238 ms	1237,004	54,552
1,97	159	110149	0,947 us	0,656 ms	655,057	28,888
0,67	173	104832	1,030 us	0,624 ms	622,974	27,473
3,56	4611	97383	27,447 us	0,580 ms	552,660	24,372

0,64	1430	779413	8,512 us	4,640 ms	4631,521	204,250
0,64	1374	571894	8,179 us	3,405 ms	3396,853	149,801
0,64	4775	2083523	28,423 us	12,402 ms	12373,688	545,680
0,00	282	34019	1,679 us	0,203 ms	201,328	8,879
0,00	462	85864	2,750 us	0,512 ms	509,261	22,458
0,63	289	631806	1,721 us	3,761 ms	3759,286	165,785
0,63	289	631806	1,721 us	3,761 ms	3759,286	165,785
0,58	5139	815748	30,590 us	4,856 ms	4825,529	212,806
0,58	5152	815562	30,667 us	4,855 ms	4824,453	212,758
0,62	8475	2119308	50,447 us	12,615 ms	12564,750	554,105
0,65	199	104316	1,185 us	0,621 ms	619,820	27,334
0,00	128	342	0,762 us	2,036 us	1,277	0,056
1,40	182	60750	1,084 us	0,362 ms	360,920	15,917
0,70	543	2007700	3,233 us	11,951 ms	11947,780	526,897
0,63	505	215556	3,006 us	1,284 ms	1281,006	56,492
-0,88	157	17888	0,935 us	0,107 ms	106,069	4,678

c. No FP

Dans le tableau suivant, « Δ » indique le ratio (*speedup*) entre la version «Soft FP subnormal » et celle gérant les flottants sans unité de traitement flottant, mais en prenant en compte la dénormalisation.

	No FPU subnormal					
Δ	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	**Cumul. T block 256 (µs)**	**Load (block 256)**
4,9	307	5754	1,828 us	34,250 us	32,429	1,430
2,8	5024	327765	29,905 us	1,951 ms	1921,212	84,725
10,0	146	1054290	0,870 us	6,276 ms	6275,133	276,733
3,4	238	108693	1,417 us	0,647 ms	645,589	28,470
4,6	4033	401676	24,006 us	2,391 ms	2367,088	104,389
3,7	5386	400754	32,060 us	2,386 ms	2354,065	103,814
9,0	4140	1723711	24,643 us	10,261 ms	10236,453	451,428
2,7	5241	362735	31,197 us	2,160 ms	2128,925	93,886
4,3	4191	166210	24,947 us	0,990 ms	965,150	42,563
2,2	6727	397382	40,042 us	2,366 ms	2326,114	102,582
4,2	4146	426403	24,679 us	2,539 ms	2514,417	110,886
2,7	5245	362383	31,221 us	2,158 ms	2126,901	93,796
3,9	154	4494	0,917 us	26,750 us	25,837	1,139
3,0	196	1773	1,167 us	10,554 us	9,392	0,414
10,0	232	1047696	1,381 us	6,237 ms	6235,624	274,991
8,3	4046	1227788	24,084 us	7,309 ms	7285,010	321,269
9,1	4406	2260039	26,227 us	13,453 ms	13426,875	592,125
4,2	177	27406	1,054 us	0,164 ms	162,950	7,186
3,4	146	12559	0,870 us	74,756 us	73,889	3,259
10,1	118	528481	0,703 us	3,146 ms	3145,300	138,708

4,3	3838	150079	22,846 us	0,894 ms	871,243	38,422
1,9	1718	137658	10,227 us	0,820 ms	809,813	35,713
1,2	248	3773	1,477 us	22,459 us	20,988	0,926
10,1	118	528011	0,703 us	3,143 ms	3142,300	138,575
9,7	548	2149008	3,262 us	12,792 ms	12788,751	563,984
9,6	4771	5401234	28,399 us	32,151 ms	32122,712	1416,612
10,0	368	1043557	2,191 us	6,212 ms	6209,818	273,853
9,8	309	1052922	1,840 us	6,268 ms	6266,167	276,338
2,6	5317	262384	31,649 us	1,562 ms	1530,475	67,494
4,2	4147	378289	24,685 us	2,252 ms	2227,411	98,229
2,8	5004	327096	29,786 us	1,947 ms	1917,330	84,554
10,1	228	5213716	1,358 us	31,035 ms	31033,647	1368,584
9,4	3881	3300368	23,102 us	19,646 ms	19622,988	865,374
8,3	3873	1224040	23,054 us	7,286 ms	7263,036	320,300
2,7	148	9378	0,881 us	55,822 us	54,944	2,423
9,9	186	1049905	1,108 us	6,250 ms	6248,896	275,576
4,8	143	3206	0,852 us	19,084 us	18,235	0,804
2,1	1240	176454	7,381 us	1,051 ms	1043,648	46,025
3,3	191	57299	1,137 us	0,342 ms	340,867	15,032
5,0	285	248503	1,697 us	1,480 ms	1478,310	65,193
3,5	1368	247411	8,143 us	1,473 ms	1464,889	64,602
10,0	419	1575045	2,495 us	9,376 ms	9373,515	413,372
9,9	535	1593992	3,185 us	9,489 ms	9485,827	418,325
9,5	943	2178106	5,614 us	12,965 ms	12959,408	571,510
2,2	1463	211633	8,709 us	1,260 ms	1251,325	55,183
4,2	182	195434	1,084 us	1,164 ms	1162,920	51,285
5,5	420	15374	2,500 us	91,512 us	89,022	3,926
5,1	271	277781	1,614 us	1,654 ms	1652,392	72,871
1,7	2943	245343	17,518 us	1,461 ms	1443,550	63,661
4,4	304	271738	1,810 us	1,618 ms	1616,197	71,274
2,1	1463	211668	8,709 us	1,260 ms	1251,325	55,183
3,5	193	206915	1,149 us	1,232 ms	1230,855	54,281
8,6	3838	1205995	22,846 us	7,179 ms	7156,243	315,590
10,0	598	2108603	3,560 us	12,552 ms	12548,454	553,387
2,6	115	1616	0,685 us	9,620 us	8,938	0,394
10,0	310	2094243	1,846 us	12,466 ms	12464,161	549,670
10,0	325	1055298	1,935 us	6,282 ms	6280,073	276,951
10,0	353	2096045	2,102 us	12,477 ms	12474,906	550,143
10,0	321	2095531	1,911 us	12,474 ms	12472,096	550,019
2,5	95	604	0,566 us	3,596 us	3,032	0,134
2,4	144	6242	0,858 us	37,155 us	36,300	1,601
6,3	256	3574	1,524 us	21,274 us	19,756	0,871
10,1	100	1041701	0,596 us	6,201 ms	6200,406	273,438
5,1	271	260783	1,614 us	1,553 ms	1551,392	68,416
1,7	1564	109088	9,310 us	0,650 ms	640,726	28,256

4,4	325	224478	1,935 us	1,337 ms	1335,073	58,877
2,1	1207	176504	7,185 us	1,051 ms	1043,843	46,033
10,0	199	3149609	1,185 us	18,748 ms	18746,820	826,735
10,0	188	2086590	1,120 us	12,421 ms	12419,884	547,717
9,7	174	1073558	1,036 us	6,391 ms	6389,968	281,798
10,0	183	1046037	1,090 us	6,227 ms	6225,914	274,563
3,0	4752	289359	28,286 us	1,723 ms	1694,824	74,742
10,1	1742	7836298	10,370 us	46,645 ms	46634,671	2056,589
10,0	1638	5747199	9,750 us	34,210 ms	34200,288	1508,233
10,0	5682	20899632	33,822 us	0,125 s	91,310	4,027
3,8	333	128131	1,983 us	0,763 ms	761,025	33,561
3,1	576	269973	3,429 us	1,607 ms	1603,584	70,718
10,0	353	6326756	2,102 us	37,660 ms	37657,906	1660,714
10,0	353	6326756	2,102 us	37,660 ms	37657,906	1660,714
9,8	5714	7988715	34,012 us	47,552 ms	47518,121	2095,549
9,8	5721	7987705	34,054 us	47,546 ms	47512,079	2095,283
9,9	9548	21051506	56,834 us	0,126 s	69,388	3,060
10,0	248	1046459	1,477 us	6,229 ms	6227,529	274,634
3,2	142	1108	0,846 us	6,596 us	5,753	0,254
9,2	199	558310	1,185 us	3,324 ms	3322,820	146,536
10,0	712	20049742	4,239 us	0,120 s	115,778	5,106
9,8	762	2111787	4,536 us	12,571 ms	12566,482	554,182
4,4	182	78055	1,084 us	0,465 ms	463,920	20,459

Dans le tableau suivant, « Δ » indique le ratio (*speedup*) entre la version «Soft FP subnormal » et celle gérant les flottants sans unité de traitement flottante, sans prendre en compte la dénormalisation.

	No FPU no subnormal					
Δ	**Cycles**	**Cumul. cycles**	**Time**	**Cumul. Time**	**Cumul. T block 256 (µs)**	**Load (block 256)**
4,9	307	5754	1,828 us	34,250 us	32,429	1,430
2,8	5024	327765	29,905 us	1,951 ms	1921,212	84,725
10,0	146	1054290	0,870 us	6,276 ms	6275,133	276,733
3,3	238	107485	1,417 us	0,640 ms	638,589	28,162
4,6	4033	401676	24,006 us	2,391 ms	2367,088	104,389
3,7	5386	400754	32,060 us	2,386 ms	2354,065	103,814
9,0	4140	1723711	24,643 us	10,261 ms	10236,453	451,428
2,7	5241	362735	31,197 us	2,160 ms	2128,925	93,886
4,3	4191	166210	24,947 us	0,990 ms	965,150	42,563
2,2	6727	397382	40,042 us	2,366 ms	2326,114	102,582
4,2	4146	426403	24,679 us	2,539 ms	2514,417	110,886
2,7	5245	362383	31,221 us	2,158 ms	2126,901	93,796
3,9	154	4494	0,917 us	26,750 us	25,837	1,139

3,0	196	1773	1,167 us	10,554 us	9,392	0,414
10,0	232	1047696	1,381 us	6,237 ms	6235,624	274,991
8,3	4046	1227502	24,084 us	7,307 ms	7283,010	321,181
9,1	4406	2260039	26,227 us	13,453 ms	13426,875	592,125
4,2	177	27406	1,054 us	0,164 ms	162,950	7,186
3,3	146	12273	0,870 us	73,054 us	72,187	3,183
10,1	118	528481	0,703 us	3,146 ms	3145,300	138,708
4,3	3838	150079	22,846 us	0,894 ms	871,243	38,422
1,9	1718	137658	10,227 us	0,820 ms	809,813	35,713
1,2	248	3773	1,477 us	22,459 us	20,988	0,926
10,1	118	528011	0,703 us	3,143 ms	3142,300	138,575
9,7	548	2149008	3,262 us	12,792 ms	12788,751	563,984
9,6	4771	5401234	28,399 us	32,151 ms	32122,712	1416,612
10,0	368	1043557	2,191 us	6,212 ms	6209,818	273,853
9,8	309	1052922	1,840 us	6,268 ms	6266,167	276,338
2,6	5317	262384	31,649 us	1,562 ms	1530,475	67,494
4,2	4147	378289	24,685 us	2,252 ms	2227,411	98,229
2,8	5004	327096	29,786 us	1,947 ms	1917,330	84,554
10,1	228	5213716	1,358 us	31,035 ms	31033,647	1368,584
9,4	3881	3300082	23,102 us	19,644 ms	19620,988	865,286
8,3	3873	1223754	23,054 us	7,285 ms	7262,036	320,256
2,7	148	9092	0,881 us	54,120 us	53,242	2,348
9,9	186	1049619	1,108 us	6,248 ms	6246,896	275,488
4,8	143	3206	0,852 us	19,084 us	18,235	0,804
2,1	1240	176454	7,381 us	1,051 ms	1043,648	46,025
3,2	191	56091	1,137 us	0,334 ms	332,867	14,679
5,0	285	248503	1,697 us	1,480 ms	1478,310	65,193
3,5	1368	247411	8,143 us	1,473 ms	1464,889	64,602
10,0	419	1575045	2,495 us	9,376 ms	9373,515	413,372
9,9	535	1593992	3,185 us	9,489 ms	9485,827	418,325
9,5	943	2178106	5,614 us	12,965 ms	12959,408	571,510
2,2	1463	211633	8,709 us	1,260 ms	1251,325	55,183
4,2	182	195434	1,084 us	1,164 ms	1162,920	51,285
5,5	420	15374	2,500 us	91,512 us	89,022	3,926
5,1	271	277781	1,614 us	1,654 ms	1652,392	72,871
1,7	2943	245343	17,518 us	1,461 ms	1443,550	63,661
4,4	304	271738	1,810 us	1,618 ms	1616,197	71,274
2,1	1463	211668	8,709 us	1,260 ms	1251,325	55,183
3,5	193	206915	1,149 us	1,232 ms	1230,855	54,281
8,6	3838	1205995	22,846 us	7,179 ms	7156,243	315,590
10,0	598	2108603	3,560 us	12,552 ms	12548,454	553,387
2,6	115	1616	0,685 us	9,620 us	8,938	0,394
10,0	310	2094243	1,846 us	12,466 ms	12464,161	549,670
10,0	325	1055298	1,935 us	6,282 ms	6280,073	276,951
10,0	353	2096045	2,102 us	12,477 ms	12474,906	550,143

10,0	321	2095531	1,911 us	12,474 ms	12472,096	550,019
2,5	95	604	0,566 us	3,596 us	3,032	0,134
2,3	144	5956	0,858 us	35,453 us	34,598	1,526
6,3	256	3574	1,524 us	21,274 us	19,756	0,871
10,1	100	1041701	0,596 us	6,201 ms	6200,406	273,438
5,1	271	260783	1,614 us	1,553 ms	1551,392	68,416
1,7	1564	109088	9,310 us	0,650 ms	640,726	28,256
4,4	325	224478	1,935 us	1,337 ms	1335,073	58,877
2,1	1207	176504	7,185 us	1,051 ms	1043,843	46,033
10,0	199	3149323	1,185 us	18,746 ms	18744,820	826,647
10,0	188	2086590	1,120 us	12,421 ms	12419,884	547,717
9,7	174	1073272	1,036 us	6,389 ms	6387,968	281,709
10,0	183	1046037	1,090 us	6,227 ms	6225,914	274,563
2,9	4752	289073	28,286 us	1,721 ms	1692,824	74,654
10,1	1742	7836298	10,370 us	46,645 ms	46634,671	2056,589
10,0	1638	5747199	9,750 us	34,210 ms	34200,288	1508,233
10,0	5682	20899632	33,822 us	0,125 s	91,310	4,027
3,8	333	128131	1,983 us	0,763 ms	761,025	33,561
3,1	576	269973	3,429 us	1,607 ms	1603,584	70,718
10,0	353	6326756	2,102 us	37,660 ms	37657,906	1660,714
10,0	353	6326756	2,102 us	37,660 ms	37657,906	1660,714
9,8	5714	7988715	34,012 us	47,552 ms	47518,121	2095,549
9,8	5721	7987705	34,054 us	47,546 ms	47512,079	2095,283
9,9	9548	21051506	56,834 us	0,126 s	69,388	3,060
10,0	248	1046459	1,477 us	6,229 ms	6227,529	274,634
3,2	142	1108	0,846 us	6,596 us	5,753	0,254
9,2	199	558310	1,185 us	3,324 ms	3322,820	146,536
10,0	712	20049456	4,239 us	0,120 s	115,778	5,106
9,8	762	2111787	4,536 us	12,571 ms	12566,482	554,182
4,4	182	78055	1,084 us	0,465 ms	463,920	20,459