TP 2
Mesure du temps et de la charge CPU

Objectif

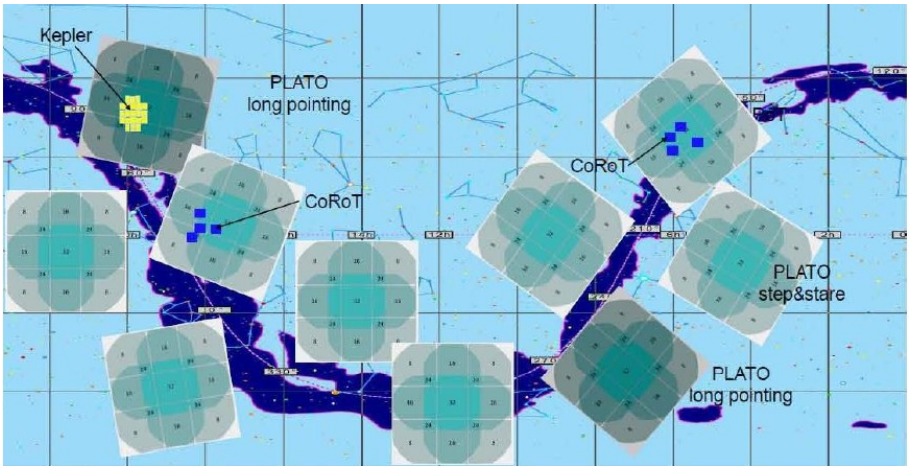
Mesurer précisément le temps d'exécution d'un algorithme est indispensable à l'élaboration d'un budget CPU, ainsi qu'à l'évaluation des optimisations possibles.

Grâce au debugueur GDB, vu dans le TP précédent, nous mesurerons les ressources CPU nécessaires à trois algorithmes et évaluerons leur utilisation dans le cadre du projet de télescope spatial PLATO. Basée d'abord sur les outils du simulateur, cette évaluation prendra en compte les optimisations du compilateur. Elle sera ensuite refaite avec les ressources propres de la cible.

>> Vous produirez un rapport de TP contenant vos réalisations et observations.

Le projet PLATO

L'observatoire spatial PLATO, en cours de développement, scrutera dès 2026 de large zone du ciel. Ses mesures photométriques continues sur plusieurs mois permettront d'identifier et caractériser les systèmes planétaires, notamment en détectant les transits d'exoplanètes et en mesurant les oscillations des étoiles.



Dans un système embarqué diverses ressources peuvent être limitantes, comme le poids, la puissance ou la mémoire. Pour PLATO une contrainte importante est la bande passante. Sur les 189 Tbits de données brutes quotidiennes, seules 435 Gbits peuvent être transmit, soit 1/434ème.

Dans cette situation, maximiser les traitements à bord permet d'augmenter les retours scientifiques, sinon limités par la bande passante disponible. Ce TP porte sur l'évaluation de certains de ces algorithmes proposés par les scientifiques.

Les algorithmes de traitement

Pour être étudiée, chaque étoile est extraite de l'image dans une fenêtre de 6x6 pixels. Lui est associé un masque de même dimension correspondant à la déformation attendue sur le capteur.

Voici un exemple de fenêtre 6x6 et de masque :

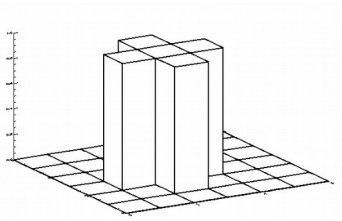
```
float window[] = {
              2.4,   52.38,       2.36,  2.34,  82.32,       92.3,
              12.4,  1710.38,     3716.36,     3558.34,     7.32,  1.3,
              7.4,   1852.38,     4516.36,     6558.34,     1689.32,     1.3,
              2.4,   52.38,       1289.36,     1289.34,     1646.32,     92.3,
              2.4,   52.38,       9.36,  1610.34,     1486.32,     92.3,
              2.4,   52.38,       2.36,  2.34,  1486.32,     92.3,
};

float mask[] = {
              0, 0, 0.12, 0.14, 0, 0,
              0, 0, 1, 0.9, 0, 0,
              0.09, 0.19, 0.96, 0.75, 0.47, 0.19,
              0, 0.13, 0.15, 0.39, 0.88, 0,
              0, 0, 0.15, 0.39, 0.88, 0,
              0, 0, 0.07, 0.19, 0, 0
};
```

Calcul de centroïde

Afin d'étudier le mouvement d'une étoile dans le plan focal, le logiciel de bord devra implémenter une fonction prenant en paramètres les tableaux de 36 float (fenêtre et masque) et retournant le barycentre dans une structure contenant deux nombres flottant (x,y).

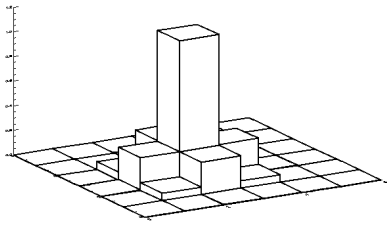
Photométrie d'ouverture



Pour étudier l'évolution de la luminosité d'une étoile, doit être étudié un algorithme de photométrie d'ouverture. Celui-ci somme les valeurs des pixels dont le masque n'est pas à zéro.

Photométrie par masque pondéré

Moins sensible à la confusion entre étoiles et au jitter, la photométrie par masque pondéré a cependant l’inconvénient de perdre des photons.



Ces 3 algorithmes ont été implémenté et leurs fichiers sources disponibles conjointement au sujet du TP.

Mesurez les performances via les outils du simulateur

Créez un projet appelant successivement chacune des 3 fonctions présentées précédemment.

Utilisez dans un script GDB le point d'arrêt suivant pour mesurer les temps d'exécution.

```
hbreak break_simu
commands
	silent
	printf "\n\n**** break = %s ****\n", break_name
	if (break_id == 0)
		mon perf reset
	end
	if (break_id == 1)
		mon perf
		mon perf reset
	end
	cont
end
```

>> Sur quel code va fonctionner ce point d'arrêt ?

>> Quelles sont les sorties des appels à ce script ? Quel est le temps d'exécution de chaque algorithme ?

Calculez le budget CPU

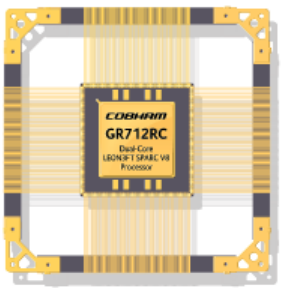
Sur le projet PLATO, pour chacun des 26 télescopes, 111500 étoiles doivent être analysées toutes les 25 secondes. Le processeur chargé de cette tâche pour 2 télescopes est un GR712RC, dont voici les spécifications :

Features

- Dual-core SPARC V8 integer unit, each with 7-stage pipeline, 8 register windows, 4x4 KiB multi-way instruction cache, 4x4 KiB multi-way data cache, branch prediction, hardware multiplier and divider, power-down mode, hardware watch-points, single-vector trapping, SPARC reference memory management unit, etc.
- Two high-performance double precision IEEE-754 floating point units
- EDAC protected (8-bit BCH and 16-bit Reed-Solomon) interface to multiple 8/32-bits
- PROM/SRAM/SDRAM memory banks
- Advanced on-chip debug support unit
- 192 KiB EDAC protected on-chip memory
- Multiple SpaceWire links with RMAP target
- Redundant 1553 BC/RT/MT interfaces
- Redundant CAN 2.0 interfaces
- 10/100 Ethernet MAC with RMII interface
- SPI, I2C, ASCS16 (STR), SLINK interfaces
- CCSDS/ECSS Telemetry and Telecommand
- UARTs, Timers & Watchdog, GPIO ports,
- Interrupt controllers, Status registers, JTAG, etc.
- Configurable I/O switch matrix

Description

The GR712RC is an implementation of the dual-core LEON3FT SPARC V8 processor using RadSafe technology. The fault tolerant design of the processor in combination with the radiation tolerant technology provides total immunity to radiation effects.



Specification

- CQFP240 package
- Total Ionizing Dose (TID) up to 300 krad(Si)
- Proven Single-Event Latch-Up (SEL) immunity
- Proven Single-Event Upset (SEU) tolerance
- 1.8V & 3.3V supply
- 15 mW/MHz processor core power consumption
- 100 MHz system frequency
- 200 Mbps SpaceWire links
- 10 Mbps CCSDS Telecommand link
- 50 Mbps CCSDS Telemetry link

>> D'après vos résultats précédents et les spécifications ci-dessus, quelle serait la charge CPU si GR712RC devait calculer les centroïdes et les photométries d'ouverture ? les centroïdes et les photométries par masques pondérés ?

Mesurez l'influence du FPU

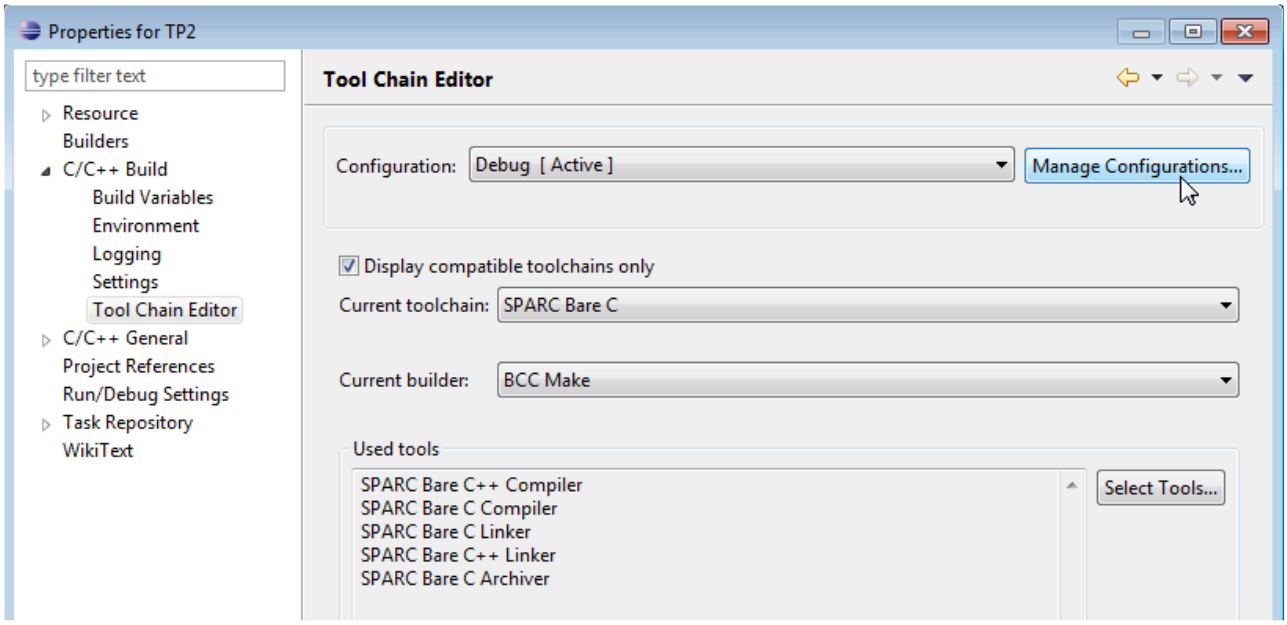
Tous les processeurs n'ayant pas de Floating Point Unit, l'option `-nofpu` de `tsim-leon3`, permet de simuler cette absence. Il est alors nécessaire de compiler le projet avec l'option `-msoft-float` pour émuler en logiciel le FPU.

>> Comparer les mesures obtenues avec et sans FPU hardware.

Mesurez l'influence des optimisations

Le compilateur GCC permet plusieurs niveaux d'optimisation de l'exécutable : O0, O1, O2 ou O3.
https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Comme les autres options de compilations, celle-ci peut être sauvegardée dans une « configuration ». Pour en créer une nouvelle, utilisez le `Manager de configuration` et créez une configuration `Debug-O1` à partir de la configuration `Debug`.



Modifiez ensuite l'option d'optimisation pour cette configuration en modifiant les propriétés du projet :
`Properties > C/C++ Build > Settings > Tool Settings > Cross GCC compiler > Optimization > Optimization Level`

Pour pousser plus loin l'étude de faisabilité, en conservant la meilleur option (FPU hardware / logiciel), mesurez les performances des algorithmes avec les niveaux d'optimisation : -O0, -O1, -O2 et -O3.

>> Dans un tableau, présentez les temps d'exécution des différents algorithmes, selon leur niveau d'optimisation ?

Calculez le budget CPU de phase A

Sur PLATO, avant de pouvoir faire le calcul du centroïde et de la photométrie, des pré-traitements doivent être réalisés. Voici les résultats de leur prototypage en O2 + GRFPU, pour une étoile :

	Computed Reference Sample (cycles)
Window extraction and conversion to double	1315
Offset and Background substraction	1191
Smearing substraction	1251

>> En prenant en compte l'ensemble des opérations à réaliser pour chaque étoile et le nombre d'étoiles traités par processeur, le taux d'occupation CPU calculé est-il acceptable au niveau d'une étude de faisabilité (phase A) ?

Mesure de performance via les ressources de la cible

Les mesures ont jusqu'à présent été réalisées grâce aux outils du simulateur. Pour monitorer le temps d'exécution des principaux algorithmes aussi bien sur simulateur qu'en conditions réelles, ces mesures doivent être faites avec les ressources propres à la cible.

Pour utiliser la fonction `bcc_timer_get_us()` équivalente à la fonction `clock()` (slide 21 du cours 3), l'inclusion suivante est nécessaire :

```
#include <bcc/bcc.h>
```

Il est égale nécessaire d'avoir préalablement défini le ticks système via :

```
int bcc_timer_tick_init_period(uint32_t usec_per_tick);
```

ainsi que d'avoir démarré les timers du processeur LEON et configurer les handlers d'interruption :

```
bcc_timer_tick_init();
```

Modifier la méthode appelée entre chaque algorithme pour qu'elle prenne en paramètre le résultat de `get_elapsed_time()`.

>> Dans un tableau, présentez les temps d'exécution des différents algorithmes, selon leur niveau d'optimisation ?

>> Expliquez les différences par rapport aux mesures précédentes.