Correction
-
TP 2
Mesure du temps et de la charge CPU

Objectif

Dans ce TP l'objectif était d'évaluer la charge CPU d'un type de calculateur du projet PLATO, d'après les prototypes des algorithmes.

Ces prototypes permettent de tester l'impact de différentes configurations, tel que différents niveaux d'optimisation ou l'utilisation ou non d'une unité de calcul en virgule flottante (Floating Point Unit – FPU).

Enfin, pour mesurer la charge CPU des algorithmes, a été étudié les différences liées à l'utilisation des outils du simulateurs ou aux ressources du système embarqué.

Mesurez les performances via les outils du simulateur

Le code suivant étant fournis pour définir le point d'arrêt, la question était de savoir comment l'utiliser.

```
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
```

Enregistré dans un fichier `script/gdb-break.txt`, le point d'arrêt a été intégré au script suivant.

```
set remotetimeout 10000

set logging file result/gdb_result-O0.txt
set logging overwrite on
set logging on
set height 0
set print pretty on
set print array on

tar extended-remote localhost:1234

load

source script/gdb-break.txt

start
cont

quit
```

La fonction break_simu() a également été ajouté au projet pour que le point d'arrêt corresponde à une fonction C. Elle prend en argument des variable dont les noms sont utilisé dans le point d'arrêt : une chaînes de caractères `break_name` et un entier `break_id`.

```
uint32_t break_simu(int break_id, const char* break_name){
	uint32_t static test_counter = 0;
	return test_counter++;
}
```

Enfin, l'appel à cette fonction s'intercale entre les appels aux 3 fonctions prototypant les algorithmes.

```
break_simu(0, "START");
barycentre(window, mask);
break_simu(1, "barycentre");
computeFluxBinaire(window, mask);
break_simu(1, "flux binaire");
computeFluxPondere(window, mask);
break_simu(1, "flux pond");
```

Le résultat obtenu est le suivant :

```
**** break = START ****


**** break = barycentre ****
Merged performance statistics for all started CPUs:
 Cycles       :      5526
 Instructions :      2511
 Overall CPI  :      2.20
 CPU performance (50.0 MHz)  :  22.72 MOPS (20.62 MIPS, 2.10 MFLOPS)
 Cache hit rate              :   96.4 % (inst: 98.0, data: 89.9)
 Simulated time              :   0.11 ms
 Processor utilisation       : 100.00 %

Performance of the simulator:
 Real-time performance       :  24.44 %
 Simulator performance       :   5.55 MIPS
 Used time (sys + user)      :   0.00 s



**** break = flux binaire ****
Merged performance statistics for all started CPUs:
 Cycles       :      2821
 Instructions :      1530
 Overall CPI  :      1.84
 CPU performance (50.0 MHz)  :  27.12 MOPS (25.77 MIPS, 1.35 MFLOPS)
 Cache hit rate              :   98.5 % (inst: 98.3, data: 99.2)
 Simulated time              :   0.06 ms
 Processor utilisation       : 100.00 %

Performance of the simulator:
 Real-time performance       :  23.38 %
 Simulator performance       :   6.34 MIPS
 Used time (sys + user)      :   0.00 s



**** break = flux pond ****
Merged performance statistics for all started CPUs:
 Cycles       :      3192
 Instructions :      1620
 Overall CPI  :      1.97
 CPU performance (50.0 MHz)  :  25.38 MOPS (23.65 MIPS, 1.72 MFLOPS)
 Cache hit rate              :   97.9 % (inst: 98.5, data: 95.5)
 Simulated time              :   0.06 ms
 Processor utilisation       : 100.00 %

Performance of the simulator:
 Real-time performance       :  24.36 %
 Simulator performance       :   6.18 MIPS
 Used time (sys + user)      :   0.00 s
```

On trouve donc le nombre de cycles suivant pour chacun des 3 algorithmes, lorsque le programme est compilé sans optimisation (O0).

	barycentre	flux binaire	flux pondéré
O0	5526	2821	3192

Calculez le budget CPU

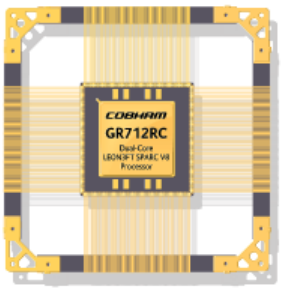
Sur le projet PLATO, pour chacun des 26 télescopes, 111500 étoiles doivent être analysées toutes les 25 secondes. Le processeur chargé de cette tâche pour 2 télescopes est un GR712RC, dont voici les spécifications :

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

Pour calculer le budget CPU, les spécifications précédentes nous apporte comme informations que :

- c'est un processeur dual-core
- qui possède deux FPUs
- et fonctionne à 100 MHz.

On peut donc calculer le budget CPU d'un cœur à 100 MHz, analysant 111500 étoiles toutes les 25 secondes, en s'appuyant sur un FPU.

$$charge\,CPU = temps\,d'exécution\,pour\,une\,étoile \times \frac{nombre\,d'étoiles\,à\,traiter}{temps\,de\,pause}$$

$$charge\,CPU = (nombre\,cycles \times durée\,d'un\,cycle\,(sec)) \times \frac{nombre\,d'étoiles\,à\,traiter}{temps\,de\,pause}$$

$$charge\,CPU = (nombre\,cycles \times \frac{1}{fréquence\,processeur}) \times \frac{nombre\,d'étoiles\,à\,traiter}{temps\,de\,pause}$$

$$charge\,CPU\,(barycentre + flux\,binaire) = ((5526+2821) \times \frac{1}{100\,000\,000}) \times \frac{111\,500}{25} \approx 37\,\%$$

$$charge\,CPU\,(barycentre + flux\,pondéré) = ((5526+3192) \times \frac{1}{100\,000\,000}) \times \frac{111\,500}{25} \approx 39\,\%$$

Mesurez l'influence du FPU

Le processeur GR712RC ayant des FPUs, étudions leur utilité en cherchant à faire sans.

Pour cela, on compile et fait l'édition de lien (linkage) du projet avec l'option `-msoft-float` :

```
sparc-gaisler-elf-gcc -O0 -msoft-float -g3 -Wall -c -fmessage-length=0 -o src/TP2.o ../src/TP2.c
sparc-gaisler-elf-gcc -O0 -msoft-float -g3 -Wall -c -fmessage-length=0 -o src/algo.o ../src/algo.c
sparc-gaisler-elf-gcc -msoft-float -o TP2 src/TP2.o src/algo.o
```

Comparons les résultats obtenus :

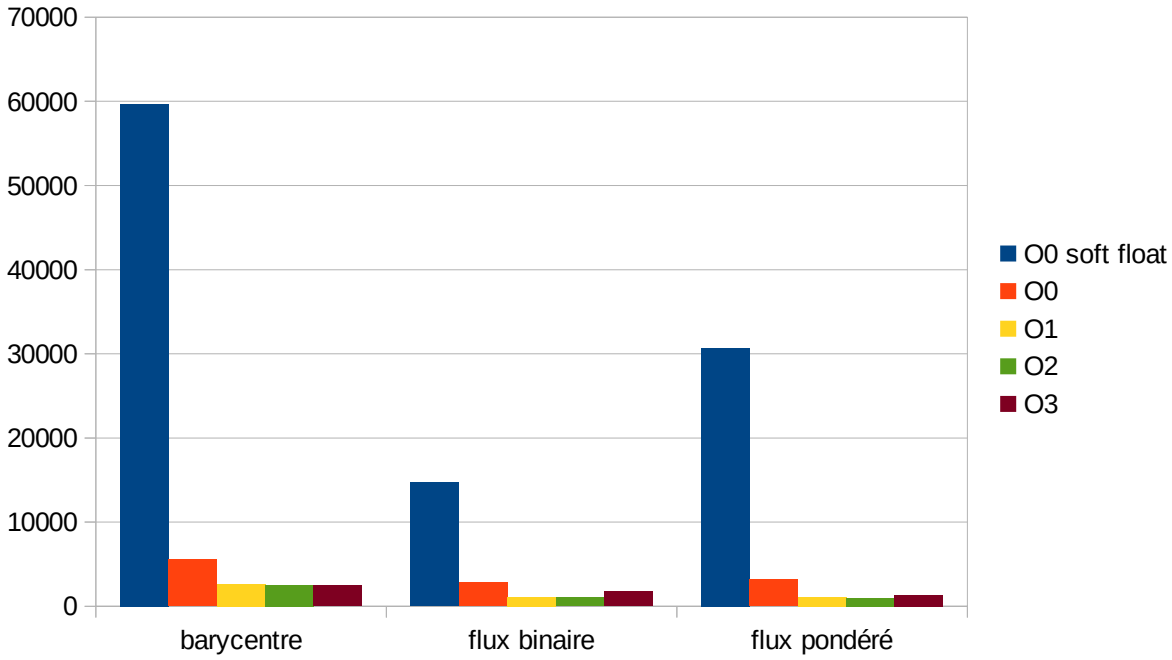
	barycentre	flux binaire	flux pondéré
O0 soft float	68008	16946	30685
O0	5526	2821	3192

Il est flagrant d'observer que la non-utilisation du FPU divise les performances presque par 10.

Mesurez l'influence des optimisations

Une fois admis la nécessité d'utiliser le FPU, le test initial a été décliné avec différents niveaux d'optimisation. On obtient les résultats suivant :

	barycentre	flux binaire	flux pondéré
O0 soft float	59715	14707	30700
O0	5526	2821	3192
O1	2661	1086	1056
O2	2555	1046	995
O3	2460	1747	1351



Si le gain entre O0 et O1 est important, il est aussi observé en O2.

Au contraire, O3 se dégrade les performances de 2 des 3 fonctions.

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. Avec une optimisation O2, voici le nombre de cycles pour l'ensemble des analyses à réaliser pour une étoile, ainsi que la charge CPU correspondante :

	Computed Reference Sample (cycles)	
Window extraction and conversion to double	1315	1315
Offset and Background substraction	1191	1191
Smearing substraction	1251	1251
barycentre	2555	2555
flux binaire	1046	
flux pondéré		995
Nombre de cycles total	**7358**	**7307**
charge CPU	32,82 %	32,59 %

Un taux d'occupation de CPU jusqu'à 40 % étant acceptable en phase A, quelque soit la méthode de calcul de flux, la charge est acceptable.

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 cela on utilise la fonction `bcc_timer_get_us()`, équivalente à la fonction `clock()`, qui donne le temps système (en micro secondes) écoulé depuis l'appel à `bcc_timer_tick_init()`

Pour évaluer l'impact de la période du tick système, on compare la performance des algorithmes, compilés en O2.

	bcc_timer_get_us() / O2		
	barycentre	flux binai	flux pondér
sans bcc_timer_get_us()	2555	1046	995
tick 10us	4054	1879	1604
tick 25us	3476	1338	1266
tick 50us	3167	1356	1092

Tout d'abord, on remarque que plus la période est courte, plus le nombre de cycles est élevé pour le même algorithme : les performances se dégradent.

	bcc_timer_get_us() / O2		
	barycentre	flux binaire	flux pondéré
sans bcc_timer_get_us()	None	None	None
tick 10us	80	40	30
tick 25us	75	25	25
tick 50us	50	0	50

En regardant les temps mesurés, on observe que les résultats « perdent en finesse » à mesure que la période augmente. Cela vient de `bcc_timer_get_us()` qui donne en µs le nombre de ticks systèmes écoulés. Le résultat ne peut donc qu'être un multiple de la période du tick. Pour être significatif, le tick système doit donc être bien inférieur au phénomène que l'on cherche à mesurer.

	bcc_timer_get_us() / O2					
	barycentre		flux binaire		flux pondéré	
	cycles	cycles(µs)	cycles	cycles(µs)	cycles	cycles(µs)
sans bcc_timer_get_us()	2555	25,55	1046	10,46	995	9,95

Si l'on converti en µs le nombre de cycles obtenu dans la première partie du TP, on observe que les mesures tournent autour de quelques dizaines de µs. Pour la suite du TP on fixe donc le tick système à 10µs.

	bcc_timer_get_us() / tick 10µs								
	barycentre			flux binaire			flux pondéré		
	cycles	cycles(µs)	measured_	cycles	cycles(µs)	measured_	cycles	cycles(µs)	measured_
O0	8620	86,2	180	4526	45,26	90	5072	50,72	100
O1	4318	43,18	80	1743	17,43	40	1877	18,77	40
O2	4016	40,16	80	1880	18,8	30	1604	16,04	40

O3 est absent du tableau car la sortie GDB ne fournissait pas le resultat de `get_elapsed_time()` : je ne suis pas parvenu à conserver la variable à l'optimisation. Ayant préalablement montré l'impact négatif de ce niveau d'optimisation, nous ne perdons pas d'information utilise à s'en passer.

Les temps « measured » montrent toujours l'intérêt d'utiliser l'optimisation lors de la compilation. Les résultats ne sont cependant pas suffisamment précis pour monter une différence entre O1 et O2. Les temps « measured » et ceux obtenus via le calcul à partir des cycles diffèrent d'un facteur 2, car le processeur simulé est à 50MHz, alors que les temps sont calculés pour un processeur à 1MHz.

	bcc_timer_get_us() / tick 10µs					
	barycentre		flux binaire		flux pondéré	
	cycles sans ticks	cycles avec ticks	cycles sans ticks	cycles avec ticks	cycles sans ticks	cycles avec ticks
O0	5526	8620	2821	4526	3192	5072
O1	2661	4318	1086	1743	1056	1877
O2	2555	4016	1046	1880	995	1604

Enfin, la comparaison du nombre de cycles obtenu dans la première partie du TP , sans configuration du tick système, avec ceux obtenues en fin de TP, montre d'importants écarts (≈ 2x), quelque soit l'optimisation. Cet écart vient de la charge CPU lié à la gestion du tick système de 10µs. C'est pour limiter cet effet que le tick système est par défaut à 10ms.

Si l'on souhaite monitorer nos traitements sans que ce le tick système ne consomme un part importante des ressources CPU, nous pourront remonter la période du tick et monitorer les traitement pour une séries d'étoiles et non individuellement pour chaque étoile.