1. Static and dynamic profiling

Analiza performantelor (profiling) reprezinta analiza comportamentului unui program pentru a determina partile din cadrul acestuia ce pot fi optimizate. Analiza dinamica se realizeaza in timpul rularii programului si poate fi implementata cu ajutorul interuperilor hardware, contori hardware, hookurilor din cadrul sistemleor de operare... Analiza statica este realizata fara a rula programul si se bazeaza pe analiza codului sursa, codului obiect sau a unei versiuni intermediare a acestuia.

Pentru arhitectura cell avem diferite utilitare pentru profiling:

- CPC (Cell Performance Counters)
- oProfile
- PDT (Performance Debugging Tool)
- PDTR (PDT Reader, or post performance)
- FDPR-Pro (Feedback Directed Program Restructuring)
- VPA (Visual Performance Analyzer)

1.1 CPC (Cell Perfromance Counters)

CPC este folosit pentru initializarea si citirea contorilor hardware din cadrul procesorului Cell. Acesti contori ii permit programatorului sa afle numarul de evenimente hardware produse de la momentul pornirii monitorizarii. Fiecare procesor Cell are 4 contori de cate 32 de biti, fiecare contor putand fi configurat ca o pereche de cate doi contori de 16 biti. Evenimentele hardware sunt produse de toate unitatile logice ale procesorului Cell (PPE, SPE, controllerul de memorie, controllerul de I/O, interface bus). Pentru monitorizarea acestor evenimente este folosita unitatea Cell Performance Monitoring Unit (PMU), aceasta permite colectarea datelor de monitorizare la intervale de timp programabile. Aceste date pot fi folosite pentru detectarea modificarilor performantelor sistemului pe perioade lungi de timp. CPC este disponibil in cadrul SDK-ului si datele de iesire oferite de acesta sunt disponibile in diferite formate: text, html, xml. Exista doua moduri de operare:

- workload - contorizare se efectueaza doar pentru un anumit proces
- system-wide - contorizare pentru toate procesele, se poate restrictiona la un anumit subset de CPUs

1.2 oProfile

---

oProfile este o aplicatie de profiling pentru sistemele Linux pe masini Cell, capabila de a analiza codul in momentul rularii cu un mic overhead suplimentar. Este inclusa in CBE SDK si are in componenta un kernel driver, un daemon ce colecteaza datele si o multitudine de statistici pentru o analiza a codului cat mai buna. Tot codul este analizat: handlerele de intreruperi hardware si software, kernelul, modulele kernel, librarirle si aplicatiile din sistem.

1.3 PDT (Performance Debugging Tool)

PDT ofera posibilitatea inregistrarii evenimentelor importante din timpul executiei programului pastrand ordinea aparitiei acestora. Principalul obiectiv al PDT-ului este de a inregistra evenimentele importante in timp real, cat si datele relevante ale SPE-uri si PPE. Pentru a colecta aceste date va creste numarul de mesaje transmise de la SPE-uri catre PPE, deoarece datele se colecteaza in memoria PPE-ului. Astfel, in cazul monitorizarii unui numar mare de SPE-uri, s-ar putea inregistra o creste mare a activitatii PPE-ului, putand influenta performantele aplicatiei monitorizate. Monitorizarea are loc la nivelul aplicatiei.

1.4 PDTR (PDT Trace Reader)

PDT Trace Reader este o aplicatie in linie de comanda care citeste si afiseaza datele furnizate de PDT si genereaza diferite rapoarte pe baza acestor date. Este inclus in SDK.

1.5 FDPR-Pro (Feedback Directed Program Restructuring)

FDPR-PRo este un utilitar folosit pentru reducerea timpului de executie si a utilizarii memoriei de catre aplicatiile user-level. Optimizeaza imaginea executabila colectand informatii despre comportamentul unui program in momentul rularii cu un volum mediu de date de prelucrare, creand o noua versiune aprogramului optimizata pentru acele date de prelucrare. Este inclus in SDK

1.6 VPA (Visual Performance Analyzer)

Calea catre VPA este /opt/tools/VPA/vpa-rcp. VPA este un tool inclus in Eclipse si este format din:

- Profiler Analyzer - include un set de unelte atat grafice cat si in mod text pentru a permite utlizatorului sa idnetifice problemele de performanta la nivel de proces, thread, modul, offset, instructiune sau linie de cod
- Code Analyzer - afiseaza informatii detaliate despre functii si instructiuni masina din fisiere executabile sau dll
- Pipeline Analyzer - afiseaza informatii despre executia in pipeline
- Counter Analyzer - este un utilitar ce analizeaza contorii de performanta hardware de pe diferite sisteme IBM
- Trace Analyzer - afiseaza informatii despre comunicaratia DMA, activatiti de lock si unlock, mesaje mailbox...
- Contor Flow Analyzer

2. Hands-on

Pentru a putea rula exemplele va trebui sa descarcati VPA de la adresa: http://www.alphaworks.ibm.com/tech/vpa/download si sa-l dezarhivati. Rulati programul vpa. Deoarece masinile din laborator nu au inca instalate utilitarele de profiling, va puteti conecta la urmatoarea masina cell

198.81.193.243 folosind ca login uid: cellstu1 ... cellstu25 si pwd: inn0vate. Fiecare student va trebui sa foloseasca un uid diferit de ceilalti, cellstuX (unde X este de la 1 la 25).

Pentru exemplul descris in acest laborator vom folosi aplicatia fft16m, o gasiti in /opt/cell/sdk/src/demos/FFT16M. Acest program calculeaza transformata fourier folosind operatii SIMD pe un array de 16.772.216 elemente.

2.1 Modificari necesare

Stergeti directorul FFT16M daca acesta exista deja si copiati codul in directorul home:

```
rm -R ~/FFT16M
cp -R /opt/cell/sdk/src/demos/FFT16M ~
```
[Get Code]

Intrati in directorul nou creat si modificati urmatoarele fisiere:

```
~/FFT16M/ppu/Makefile
~/FFT16M/spu/Makefile
```
[Get Code]

Mai jos sunt afisate fisierele Makefile cu modificarile necesare pentru rularea aplicatiilor de profiling.

~/FFT16M/ppu/Makefile

```
##########################################################################
## Target
##########################################################################
#

PROGRAM_ppu= fft

##########################################################################
## Objects
##########################################################################
#

IMPORTS = ../spu/fft_spu.a -lspe2 -lpthread -lm -lnuma

#INSTALL_DIR= $(EXP_SDKBIN)/demos
#INSTALL_FILES= $(PROGRAM_ppu)
LDFLAGS_gcc = -Wl,-q
CFLAGS_gcc = -g

##########################################################################
## buildutils/make.footer
##########################################################################
#

ifdef CELL_TOP
include $(CELL_TOP)/buildutils/make.footer
else
include ../../../../buildutils/make.footer
endif
```
[Get Code]

~/FFT16M/spu/Makefile

```
##########################################################################
## Target
```

```
############################################################################
#

PROGRAMS_spu:= fft_spu
LIBRARY_embed:= fft_spu.a

############################################################################
## Local Defines
############################################################################
#

CFLAGS_gcc:= -g --param max-unroll-times=1 # needed to keep size of
program down
LDFLAGS_gcc = -Wl,-q -g

############################################################################
## buildutils/make.footer
############################################################################
#

ifdef CELL_TOP
include $(CELL_TOP)/buildutils/make.footer
else
include ../../../../buildutils/make.footer
endif
```
[Get Code]

2.2 Colecatre data cu CPC

Dupa ce compilati noile surse, puteti sa incepeti sa folositi CPC pentru a contoriza numarul de evenimente aparute in cadrul aplicatiei. Pentru a vedea o lista cu evenimentele ce pot fi monitorizate rulati:

```
cpc --list-events
```
[Get Code]

Verificati ca cpc poate fi folosit pentru monitorizare:

```
cpc --cpus all --time 5s --events C
```
[Get Code]

Comanda de mai jos va numara instructiunile PPC comise si L1 cache load miss-uri si va afisa rezultatul intr-un format xml in fft_cpc.pmf.

```
cd ~/FFT16M
cpc --events C,2100,2119 --cpus all --xml fft_cpc.pmf ./ppu/fft 20 1
```
[Get Code]

Un alt exemplu de rulare, in care se numara evenimentele efectuate la fiecare ciclu de ceas intr-un interval de timp dat, este dat mai jos. Cu ajutorul argumentului --interval specificam un interval de timp de 100,000,000 cicli de ceas:

```
cpc -e 2100,2101,2106,2109 -e 2103,2104,2111,2119 -c all --sampling-buffer-size 15 --
interval 100000000 -X fft_cpc2.pmf -t 10 ./ppu/fft 20 1
```
[Get Code]

2.3 Vizualizare rezultate cu Counter Analyzer

Pentru a vizualiza rezultatele cu utilitarul Counter Analyzer din VPA copiati mai intai pe masina locala fisierul fft_cpc.pmf si efectuati urmatorii pasi:

- Deschideti VPA si selectati Tools > Counter Analyzer
- Selectati File > Open
- Deschideti fisierul fft_cpc.pmf

2.4 Utilizare FDPR-Pro

Utilitarul FDPR-Pro va ofera, pe langa functia de optimizare, posibilitatea de a investiga performantele aplicatiei mapand rezultatele la codul sursa in combinatie cu Code Analyzer. Initial trebuie sa utilizati FDPR-Pro pentru colectarea datelor:

- Stergeti rezultatele vechi si creati un director temporar pentru fdpr-pro

```
cd ~/FFT16M/ppu ; rm -f *.mprof *.nprof ; mkdir sputmp
```
[Get Code]

- Introduceti codul de profiling in executabilul fft:

```
fdprpro ./ppu/fft -cell -spedir sputmp -a instr
> spe_extraction -> ./ppu/fft ...
...
> processing_spe_file -> sputmp/fft_spu ...
...
> reading_exe ...
> adjusting_exe ...
...
> analyzing ...
> building_program_infrastructure ...
@Warning: Relocations based on section .data -- section may not be
reordered
> building_profiling_cfg ...
> spe_encapsulation -> sputmp/out ...
>> processing_spe -> sputmp/out/fft_spu ...
> instrumentation ...
>> throw_&_catch_fixer ...
>> adding_universal_stubs ...
>> running_markers_and_instrumenters ...
>> linker_stub_fixer ...
>> dynamic_entries_table_sections_bus_fixer ...
>> writing_profile_template -> fft.nprof ...
> symbol_fixer ...
> updating_executable ...
> writing_executable -> fft.instr ...
bye.
```
[Get Code]

- Rulati fisierul executabil creat:

```
./fft.instr 20 1
```
[Get Code]

- Ar trebui sa gasiti urmatoarele fisiere:

```
~/FFT16M/ppu/fft.nprof # PPU profile information
~/FFT16M/ppu/fft_spu.mprof # SPU profile information
```
[Get Code]

2.5 Vizualizati informatiile generate de FDPR-Pro cu Code Analyzer

Copiati mai intai fisierele fft, fft.c, fft.nprof, fft_spu.mprof pe masina locala. Pentru a putea vizualiza informatiile generate de FDPR-Pro efectuati urmatorii pasi:

- Deschideti VPA si selectati Tools > Code Analyzer
- Mergeti in File > Code Analyzer > Analyze Executable, si deschideti fisierul executabil fft orginal. Ar trebui sa vi se deschida doua taburi, unul pentru PPU, unul pentru SPU, ca in imaginea de mai jos:

- Introduceti informatiile generate despre PPU mergand in File > Code Analyzer > Add Profile Info si selectati fft.nprof.

- Dupa ce veti introduce informatiile despre PPU, instructiunile vor fi colorate in funcite de frecventa executiei lor (rosu pentru cele mai frecvente)

- Puteti asocia si codul sursa selectand simbolul dorit in Program Tree, click-dreapta > Open Source Code si deschizand fisierul sursa. Rezultatul ar trebui sa adauge codul sursa intr-un nou tab si sa arate frecventa executiei liniilor de cod.

- Pentru informatii despre disptach groups apasati pe butonul "Collect display information about dispatch groups"

- Vizualizati informatiile de pipeline pentru fiecare dispatch group selectand tabul Dispatch Info in coltul din dreapta jos si apasand butonul "Link with table"

- Code Analyzer ofera si posibilitatea de a inspecta informatiile SPU-Timing la nivel de pipeline. Selectati tabul SPU, selectati simboul dorit in Program Tree, click-dreapta si apasati pe "Select Timing"

2.6 Utilitarul PDT

PDT ofera posibilitatea inregistrarii evenimentelor importante din timpul executiei programului, vizualizarea acestora se face cu utilitarul Trace Analyzer. Va trebui sa recompilati aplicatia fft utilizand urmatoarele configurari:

- modificati spu/Makefile ca mai jos:

```
##############################################################################
## Target
##############################################################################
#

PROGRAMS_spu:= fft_spu
LIBRARY_embed:= fft_spu.a

##############################################################################
## Local Defines
##############################################################################
#

CFLAGS_gcc:= -g --param max-unroll-times=1 -Wall -Dmain=_pdt_main
-Dexit=_pdt_exit -DMFCIO_TRACE -DLIBSYNC_TRACE
LDFLAGS_gcc = -Wl,-q -g -L/usr/spu/lib/trace
INCLUDE = -I/usr/spu/include/trace
IMPORTS = -ltrace

##############################################################################
## buildutils/make.footer
##############################################################################
#

ifdef CELL_TOP
include $(CELL_TOP)/buildutils/make.footer
else
include ../../../../buildutils/make.footer
endif
```
[Get Code]

- recompilati aplicatia

```
cd ~/FFT16M
make
```
[Get Code]

- setati fisierul de configurare doar pentru stagnarile (stalls) relevante:

1. copiati fisierul default de configurare in directorul fft:

```
cp /usr/share/pdt/config/pdt_cbe_configuration.xmccl ~/FFT16M.
```
[Get Code]

1. Deschideti pentru editare fisierul copiat
2. Pe prima linie schimbati numele aplicatiei in fft (application_name="fft")
3. Cautati in fisier <*configuration name="SPE"*>. Sub acea linie veti gasi grupul MFCIO, setati tag-ul grupului pe active="false"
4. Stergeti grupul SPE_MFC si salvati.

- pregatiti mediul de lucru prin setarea urmatoarelor variabile

```
export LD_LIBRARY_PATH=/usr/lib/trace
export PDT_KERNEL_MODULE=/usr/lib/modules/pdt.ko
export PDT_CONFIG_FILE=~/FFT16M/pdt_cbe_configuration.xml
```
[Get Code]

- Rulati de cel putin trei ori aplicatia pentru a avea date adecvate.

```
cd ~/FFT16M/ppu ; ./fft 1 1 4 1 0
```
[Get Code]

Trebuie sa desetati variabila LD_LIBRARY_PATH pentru a putea rula binarul original mai tarziu.

Generati un raport cu PDTR

Pentru a genera un complet raport text din fisierele rezultate, puteti utiliza PDTR pentru a produce fisiere .pep in format ASCII:

```
$ /opt/cell/sdk/prototype/usr/bin/pdtr -trc <nume fisiere trace(fara extensie)>
```
[Get Code]

Importati datele PDT in Trace Analyzer

Copiati fisierul .pep rezultat pe masina locala. Utilitarul Trace Analyzer va permite vizualizarea etapelor de executie a unei aplicatii folsoind datele generate de PDT.

- In VPA selectati Tools > Trace Analyzer
- Selectati File > Open File si deschideti fisier .pex generat anterior.

2.7 Colectare date cu OProfile

Pentru a putea controla si configura oprofile se foloseste comanda:

```
opcontrol
```
[Get Code]

Puteti sa vizulizati lista de evenimente ce pot fi monitorizate cu ajutorul comenzii:

```
opcontrol --list-events
```
[Get Code]

Pentru a specifica un eveniment pentru monitorizare folositi:

```
opcontrol --event=NUME_EVENIMENT:INTERVAL
```
[Get Code]

Deoarece rularea oprofile necesita acces root, nu veti putea rula pe masinile cell disponibile. Un exemplu de rulare este prezentat mai jos. Informatiile generate de oProfile pot fi vizualizate in VPA in modul Profile Analyzer. Exemplu:

```
[brutman@qdc167 sandbox]$ sudo opcontrol --init
[brutman@qdc167 sandbox]$ sudo opcontrol --no-vmlinux
[brutman@qdc167 sandbox]$ sudo opcontrol --event=SPU_CYCLES:100000
[brutman@qdc167 sandbox]$ sudo opcontrol --start
Forcing required option --separate=lib with SPU_CYCLES
Forcing required option --separate=cpu with SPU_CYCLES
Using 2.6+ OProfile kernel interface.
Using log file /var/lib/oprofile/oprofiled.log
Daemon started.
Profiler running.
[brutman@qdc167 sandbox]$ sudo opcontrol --reset
Signalling daemon... done
[brutman@qdc167 sandbox]$ ./FFT16M/ppu/fft 1 0
```

```
(output from fft workload snipped)
[brutman@qdc167 sandbox]$ sudo opcontrol --stop
Stopping profiling.
[brutman@qdc167 sandbox]$ sudo opcontrol --dump
[brutman@qdc167 sandbox]$ sudo opcontrol --deinit
Stopping profiling.
Killing daemon.
Unloading oprofile module
[brutman@qdc167 sandbox]$
```
[Get Code]

Pentru a genera un raport bazat pe simboli folositi opreport --symbols.

```
[brutman@qdc167 sandbox]$ opreport --symbols
CPU: ppc64 Cell Broadband Engine, speed 3200 MHz (estimated)
Counted SPU_CYCLES events (SPU Processor Cycles) with a unit mask of 0x00 (Count
cycles [mandatory]) count 100000
Samples on CPU 0
Samples on CPU 1
Samples on CPU 2
Samples on CPU 3
Samples on CPU 4
Samples on CPU 5
Samples on CPU 6
Samples on CPU 7
samples % samples % samples % samples % samples % samples % samples % samples % image
name symbol name
398029 44.0202 398028 44.0201 391874 43.9982 390029 44.0115 390027 44.0113 390028
44.0114 390027 44.0113 390028 44.0114 fft_spu _exit
394862 43.6699 394729 43.6552 388813 43.6545 386899 43.6584 386750 43.6415 386880
43.6562 386718 43.6379 386781 43.6450 fft_spu main
71511 7.9088 71506 7.9082 70454 7.9103 70020 7.9012 70022 7.9014 70023 7.9015 70022
7.9014 70017 7.9008 fft_spu process8192_816
29430 3.2548 29594 3.2730 29233 3.2822 29013 3.2739 29210 3.2961 29069 3.2802 29199
3.2949 29192 3.2941 fft_spu process8192_0
6633 0.7336 6580 0.7277 6581 0.7389 6525 0.7363 6529 0.7367 6527 0.7365 6557 0.7399
6442 0.7269 fft_spu stage1
2881 0.3186 2902 0.3209 2859 0.3210 2868 0.3236 2816 0.3178 2830 0.3193 2832 0.3196
2890 0.3261 fft_spu _transpose_matrix4x4
289 0.0320 288 0.0319 285 0.0320 281 0.0317 284 0.0320 286 0.0323 282 0.0318 285
0.0322 fft_spu stage3
265 0.0293 270 0.0299 263 0.0295 261 0.0295 262 0.0296 262 0.0296 263 0.0297 261
0.0295 fft_spu stage2
122 0.0135 125 0.0138 122 0.0137 123 0.0139 121 0.0137 120 0.0135 118 0.0133 121
0.0137 fft_spu _sind2
119 0.0132 124 0.0137 122 0.0137 117 0.0132 117 0.0132 123 0.0139 122 0.0138 123
0.0139 fft_spu _cosd2
41 0.0045 40 0.0044 42 0.0047 48 0.0054 42 0.0047 44 0.0050 48 0.0054 39 0.0044
fft_spu __divdf3
7 7.7e-04 3 3.3e-04 6 6.7e-04 5 5.6e-04 7 7.9e-04 2 2.3e-04 5 5.6e-04 8 9.0e-04
fft_spu __pack_d
4 4.4e-04 4 4.4e-04 2 2.2e-04 5 5.6e-04 5 5.6e-04 2 2.3e-04 3 3.4e-04 3 3.4e-04
fft_spu __do_global_dtors_aux
4 4.4e-04 4 4.4e-04 4 4.5e-04 3 3.4e-04 5 5.6e-04 1 1.1e-04 1 1.1e-04 7 7.9e-04
fft_spu __unpack_d
[brutman@qdc167 sandbox]$
```
[Get Code]

Pentru a genera un raport bazat pe fisiere folositi opreport --long-filenames.

```
[brutman@qdc167 sandbox]$ opreport --long-filenames
CPU: ppc64 Cell Broadband Engine, speed 3200 MHz (estimated)
```

```
Counted SPU_CYCLES events (SPU Processor Cycles) with a unit mask of 0x00 (Count
cycles [mandatory]) count 100000
Samples on CPU 0
Samples on CPU 1
Samples on CPU 2
Samples on CPU 3
Samples on CPU 4
Samples on CPU 5
Samples on CPU 6
Samples on CPU 7
cpu:0| cpu:1| cpu:2| cpu:3| cpu:4| cpu:5| cpu:6| cpu:7|
samples| %| samples| %| samples| %| samples| %| samples| %| samples| %| samples| %|
samples| %|
-------------------------------------------------------------------------------------
-------------------------------------------------------
904197 100.000 904197 100.000 890660 100.000 886197 100.000 886197 100.000 886197
100.000 886197 100.000 886197 100.000
/home/brutman/sandbox/FFT16M/ppu/fft
[brutman@qdc167 sandbox]$
```
[Get Code]

3. Concluzii

In acest laborator ati putut vedea cum puteti folosi utilitarele de analiza a performantelor pentru a gasi o multitudine de informatii despre aplicatiile scrise. Utilizarea acestora va va permite optimizarea mai facila a aplicatiilor scrise, cat si gasirea diferitelor probleme din cadrul acestora. Deasemenea, deoarece puteti vizualiza locurile in care aplicatiia sta (stall) puteti identifica si problemele de sincronizare.

4. Linkuri utile

1. http://www.ibm.com/developerworks/edu/pa-dw-pa-cellide30-3.html
2. OProfile : http://oprofile.sourceforge.net/news/
3. Visual Performance Analyzer : http://www.alphaworks.ibm.com/tech/vpa/