PLANIFICAREA LUCRĂRILOR DE LABORATOR LA DISCIPLINA "SISTEME ÎNCORPORATE – C"

"TEHNOLOGII PENTRU SISTEME DEDICATE – TI"

"SISTEME DEDICATE – ISM"

(an IV, sem. 2)

- **http://webspace.ulbsibiu.ro/adrian.florea/html/simulatoare/Planif_Embedded_Systems.pdf** -

L1. INTRODUCERE.

a) Prezentare generală a conţinutului laboratorului structurat pe simularea şi optimizarea arhitecturilor RISC de tip VLIW folosind simulatoarele *execution-driven* **aferente** unei maşini cu execuţie multiplă.
b) Instalare SO Linux pe maşina nativă sau instalarea unei maşini virtuale pe care să ruleze Ubuntu 16.04. Familiarizarea cu comenzile Linux.
c) Analiza rezolvării unei probleme prin compilarea, depanarea şi execuţia programului C/C++. Aplicarea unor opţiuni de compilare şi link – editare.

L2. UTILIZAREA SIMULATORULUI VLIW-DLX: ilustrarea principiilor fundamentale ale procesării VLIW (*very long instruction word*). Rolul software-lui în detecţia şi eliminarea hazardurilor RAW. Relaţia dintre instrucţiunea multiplă şi instrucţiunile RISC primitive şi independente, care vor fi alocate unităţilor de execuţie în conformitate strictă cu poziţia lor în instrucţiunea multiplă (număr/latenţe). Optimizări software in procesoarele VLIW (Loop Unrolling, Software Pipelining). Avantaje / Dezavantaje faţă de procesoarele superscalare. Aplicabilitate – sisteme dedicate (procesoare de semnal, procesoare multimedia). Aplicaţie rezolvată – Se dă un vector de elemente în virgulă mobilă care trebuie translatat cu o constantă (de asemenea flotantă). Se execută programul xpk.s pe procesorul scalar pipeline (DLX) – cu și fără forwarding și apoi vliwxpk.vdlx (program neoptimizat) pe procesorul VLIW-DLX. Se aplică optimizarea de tip Loop Unrolling (LU) cu 2 (2loop_xpk.vdlx) respectiv 4 desfășurări. Se execută pe VLIW_DLX programul care inmulteste doua matrici patratice apoi se aplica LU cu 2 desfășurări + implementare pe DLX a aceluiași program. Comparația performanțelor obținute.

L3. a) Descărcarea sistemului VEX (VLIW Example) de la adresa **http://www.hpl.hp.com/downloads/vex/vex-3.43.i586.tgz**. Instalarea şi testarea pe un program simplu care determină maximul / minimul / sortează dintr-un şir de numere. Descriere arhitectură și compilator pe baza cunoștințelor acumulate la curs. Parcurgerea manualului de utilizare a instrumentului VEX (http://home.deib.polimi.it/ashouri/courses/VEX_manual.pdf; VEX_slides.pdf). Prezentarea rezultatelor sub forma unui tutorial. Sesizarea diferenţelor dintre compilatorul pentru arhitectura nativă (Intel) versus cel dedicat pentru arhitectura simulată VEX (**./cc**) si **gcc** (cc). Testarea opțiunilor de compilare / optimizare, a flag-urilor de profiling **-O1**, **-c**, **-s**, **-width**, **-fmmdump**, **-mas_t** pe diverse programe etc. (cu informații privind accesele la cache, salturi). Analizarea statisticilor privitoare la configurația cache-urilor și a ratelor de hit obținute (fișierele **ta.log.###**).

Realizare tutorial la fiecare lucrare de laborator cu verificare pe concret a comenzilor / opțiunilor de compilare date. Fiecare echipă descrie 10 comenzi de compilare diferite, le rulează fără erori și face capturi de ecran în urma execuțiilor.

b) Să se scrie un program care adună diferența a doi vectori (Fischer's book pg. 48 cap. 2). i) Să se genereze codul în limbaj de asamblare specific arhitecturii VEX. ii) Să se verifice

latența (-fmmdump) execuției instrucțiunilor de tip load și compare. iii) Să se impună ca latența configurației VLIW să devină (load – 3 cycles și compare – 2 cycles).

L4. a) Cerința de la punctul b) lucrarea L3 se va testa și pe programul de decodare **mpeg2 (https://www.complang.tuwien.ac.at/cd/vliw/ass2.tgz și https://www.complang.tuwien.ac.at/cd/vliw/Makefile).**

Using the VEX toolchain (http://www.hpl.hp.com/downloads/vex), perform the following experiments with the MPEG2 decoder available on the link above. The makefile assumes that the vex toolchain is installed in "$(HOME)/vex". You can use the phony target "sim" to run the decoder and compare the results to reference data. Setul de instrumente VEX oferă suport pentru personalizarea arhitecturii VLIW din perspectiva resurselor de execuție și a arhitecturii memoriei folosite. **Să se compileze și execute ulterior aplicația mpeg2 decoder pentru modelele arhitecturale RISC, vex2, vex4 și vex8.** Să se verifice codul în limbaj de asamblare rezultat. Să se analizeze și ilustreze grafic timpul de execuție, IPC, Rata de hit, ciclii de stagnare și numărul de operații NOP în funcție de modelele arhitecturale.

Se execută succesiv etapele (comenzile):

i) Doar pentru realizarea cerinţei de la punctul **c)** trebuie să tastaţi şi această comandă:
export VEXCFG = /home/aflorea/Downloads/vex-3.4.3/share/apps/h264dec/test/vex_nou.cfg

ii) În fişierul Makefile (varianta nu din arhivă ci din cel de-al doilea link) se adaugă / modifică calea spre directorul unde se află setul de instrumente VEX (de exemplu ar putea fi: **/home/aflorea/Downloads/vex-3.4.3**)

iii)Se compilează cu comanda **make all**

iv) Se rulează cu **make gen**

v) Se rulează cu **make sim**

b) Descriere fișier de configurare a cache-urilor. Identificarea unei configurații. Modificare parametrii. Să se analizeze timpul de execuție, IPC, Rata de hit, ciclii de stagnare și operații NOP în funcție de configurația cache-ului. Aplicaţii bazate pe modificarea parametrilor ierarhiei de memorie. Se modifică fişierul *vex.cfg*.

L5. a) Să se implementeze algoritmul de compresie Huffman. Sa se aplice codului asamblare rezultat in laboratorul precedent. Sa se determine factorul de reducere a puterii consumate prin compresie (slide 44 Curs_3_Embedded_Systems_AF.pdf) presupunand ca fisierul rezultat va avea un numar de instructiuni redus prin rata de compresie Huffman. De asemenea, ratio of on-chip/off-chip memory power dissipation este data de numarul acceselor cu Hit raportat la numarul acceselor cu miss din cache-ul de instructiuni rezultat in urma executiei. Se considera instructiunile pe 8, 16 si 32 de biti.

b) Sa se repete cerinta de la 5a) folosind de aceasta data algoritmul de compresie nativ Linux (gzip).

c) Sa se testeze cele 4 metode de codificare a instructiunilor VLIW (Uncompressed Encoding, Fixed-overhead Encoding, Distributed Encoding si Template-based Encoding - **Fischer's book pg. 115-117 cap. 3.5 VLIW Encoding**) pe un program sursa asamblare (poate fi unul dintre cele folosite in laboratoarele precedente - aplicatia vliw_xpk.vdlx sau programul asamblare din laboratorul 2 punctul b). Sa se determine impactul dpdv al puterii consumate. Pentru codificari a se vedea pg. 8-10 de la adresa http://www.complang.tuwien.ac.at/cd/vliw/vliw_lecture2.pdf

d) In mod analog rezolvati problema 4 din assignment1.pdf (http://www.complang.tuwien.ac.at/cd/vliw/assignment1.pdf).

L6. a) Scrierea unei funcţii apelată în programul principal de calcul a minimului dintr-un vector. Aplicarea acesteia la sortarea prin selectie. Compilarea şi generarea codului maşina cu ***-mas_G***. Se rulează ***gprof*** **nume_exec**.

Se analizează numărul de apeluri al funcţiilor. Se rulează **./rgg nume_exec** şi se generează graful apelurilor. Se determina in ce functii se executa cea mai mare parte din timp.

b) Se repetă cerinţa de la **a)** pe programul de decodare **mpeg2** (vezi lucrarea L4).

L7. a) Folosind compilatorul VEX compilați și executați un program de test care include cel puțin o funcție. Colectați informațiile **statice** privitoare la IPC-ul obținut în fiecare funcție (similar exemplului cu optimizarea Loop Unrolling de la curs). Se va folosi utilitarul **pcntl**. Se va rula executabilul generat și se va determina IPC-ul **dinamic**. Comparați rezultatele obținute și explicate diferența dintre ele.

Subetape în realizarea cerinței:

- Se compilează (**-S -O0**) codul sursă (**.c**) pentru obținerea codului în asamblare (**.s**).
- Se rulează utilitarul **pcntl** conform manualului VEX (http://home.deib.polimi.it/ashouri/courses/VEX_manual.pdf).
- Se analizează codul asamblare și se compară cu rezultatul obținut după aplicarea **pcntl**.
- Se compilează (**-o**) codul sursă (**.c**) pentru obținerea codului executabil.
- Se rulează codul executabil și se compară IPC-urile obținute în varianta statică cu cel obținut în varianta dinamică
- Se repetă pașii anteriori pentru codul optimizat (**-O3**)

b) Se aplica metodele de optimizare prezentate la curs in aplicatii (***Loop Interchange, Loop-Based Strength Reduction, Induction Variable Elimination, Merging Array, Loop Fusion, Loop Tiling / Blocking***). Se va realiza un meniu pentru selectia la alegere a metodelor de optimizare. Se vor compila sursele cu optiunile **-O0** si **-O4** si se vor compara rezultatele: timp executie, IPC, Rata de hit in cache-ul de date.

L8÷L10. a) Să se descarce suita de benchmark-uri MiBench (http://wwweb.eecs.umich.edu/mibench/source.html) specifică sistemelor incorporate. Sa se compileze suita pentru arhitectura VEX introducând in mod obligatoriu flag-urile "**-mas_G**" si "**-fmm=vex4.mm**". Acest flag va fi variat apoi pentru diferite configurații ale mașinii. Atenție, trebuie schimbat compilatorul (implicit este gcc) si trebuie eliminata opțiunea de compilare "-sttatic". De asemenea, trebuie incluse biblioteca matematica. Descrierea programelor de test se găsește la adresa http://wwweb.eecs.umich.edu/mibench/Publications/MiBench.pdf .

b) Se repetă graficul din Figura 1 din articolul „*MiBench: A free, commercially representative embedded benchmark suite”* (cu procentajul fiecărui tip de operaţii: fp, int, load, store, salturi condiţionate şi necondiţionate) pentru benchmark-urile cu număr de instrucţiuni < 10 milioane de instrucţiuni, pentru fiecare clasă de programe de test: Auto/Industrial, Consumer, Office, Network, Security, Telecomm.

c) Să se scrie un fișier script care compilează si lansează succesiv in execuție benchmark-urile suitei MiBench pentru toate configurațiile din tabelul 2 prezentat in articolul "Najafi, M. Hassan, and Mostafa E. Salehi. "*Exploring the design space for area-efficient embedded vliw packet processing engine*", in Electrical Engineering (ICEE), 2013 21st Iranian Conference on, pp. 1-6. IEEE, 2013."

d) Se va scrie un script prin intermediul căruia se va citi fișierul de rezultate ta.log.001 de unde se va selecta IPC-ul. Pe baza fiecărei configurații rulate se calculează aria de integrare compusa din doua componente. Pentru prima (aria ocupata de nucleul funcțional) se folosește formula descrisa in cap. IV.B din articolul amintit anterior. Pentru cea de-a doua, se identifica din fișierul de rezultate informațiile referitoare la configurația cache-ului și apoi se apelează

utilitarul CACTI (http://www.hpl.hp.com/research/cacti/) care identifica timpul de acces memorie, aria de integrare si energia disipata. Se va realiza un fișier Excel care ilustrează pe axa OX performanta iar pe OY aria de integrare.

e) După identificarea configurațiilor cu performanta cea mai buna se vor compila benchmark-urile cu compilatorul VEX pentru toate cele 4 optimizări posibile: "**-O1**" - "**-O4**". Se execută din nou și se reface graficul de la punctul c).

f) Se repetă analiza folosind de aceasta data pentru calculul ariei de integrare formulele propuse in articolul "A VLIW architecture for GSM benchmark & Code Optimization using Vex Compiler", Sumit Ahuja, Gurashish Singh Brar, (http://www.geocities.ws/rushtosumit/report_vliw.pdf).

L11. Testarea aplicațiilor demonstrative de la adresa https://sites.google.com/site/alarivliw11/demo.tgz?attredirects=0 (***demo1***, ***demo2***). Analiza grafic vizuală a rezultatelor – prezentare grafuri de control şi de date. Atentie trebuie modificată calea spre **cc** şi **xvg**. Folosirea unor programe mai mici (subrutine).

L12. Evaluare practică privind materia studiată la laborator.

Teme proiect

1) **Scrierea unui program care identifică timpul de viață al unor variabile (scalare, vectoriale). În cazul vectorilor variabilă de tip vector se identifica prin adresa de start a sa.** Se considera un program simplu. Se identifica fiecare linie a programului (se creează un tablou de linii de șiruri de caractere) care are o instrucțiune de atribuire / sau este startul unei bucle sau este un return. Se identifica toate variabilele existente in stânga egalului. Pentru simplitate consideram identificatorii de variabile ca încep cu _ (ex. _a, _b).

2) Dezvoltarea unei aplicatii care implementeaza un GA pentru maparea variabilelor in registrii CPU: Register Allocation: a discrete optimization problem (mapping a large number of variables (fie ***n***) to small number of registers of the processor (fie ***k = 32***)). (Probleme_propuse_HC.doc - pr. 14)
Solutie de start implementarea algoritmului pe un graf neorientat dat prin matrice de adiacenta, folosind tehnica de programare backtracking.

3) Explorarea spațiului de proiectare aferent procesoarelor VLIW din punct de vedere a performanței de procesare, a eficienței energetice și a ariei de integrare. Testarea pe benchmark-urile MiBench.