David Morillo Massagué (**1666540**)

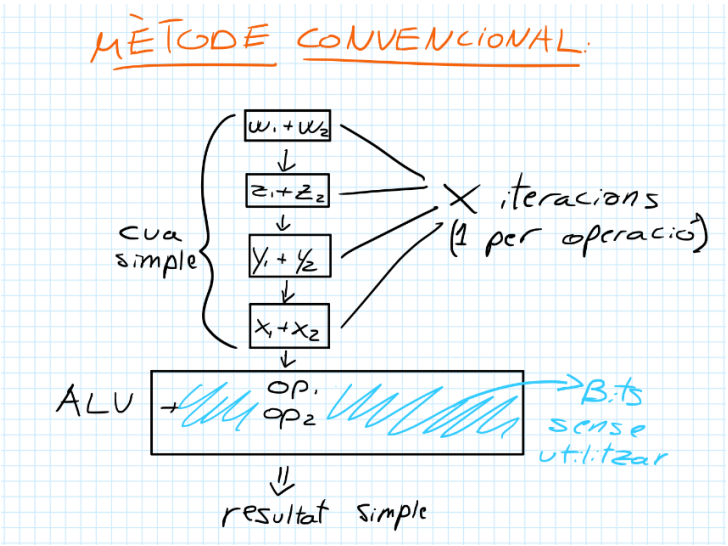
01/01/2024

Informe de Pràctiques Individual

En aquest informe s'intentarà explicar, resumidament, el que he après i considero més important de cada de les següents pràctiques, utilitzant les meves paraules i sense recolzar-se en expressions dels documents penjats al Campus Virtual.

P6. Vectorització i instruccions SIMD

En aquesta pràctica, el que es volia demostrar era la capacitat del processador de realitzar una sola operació matemàtica però, proporcionar "més d'un resultat" i, per tant, poder reduir el nombre de iteracions (per tant, reduir el temps de còmput) d'un programa.



ALU: Unitat aritmètica lògica

En aquest fragment dels meus apunts es mostra l'idea que tenia inicialment de com es feien les operacions que apareixien en el nostre codi: un càlcul (un ús de la ALU del processador) per cada operació elemental del codi.

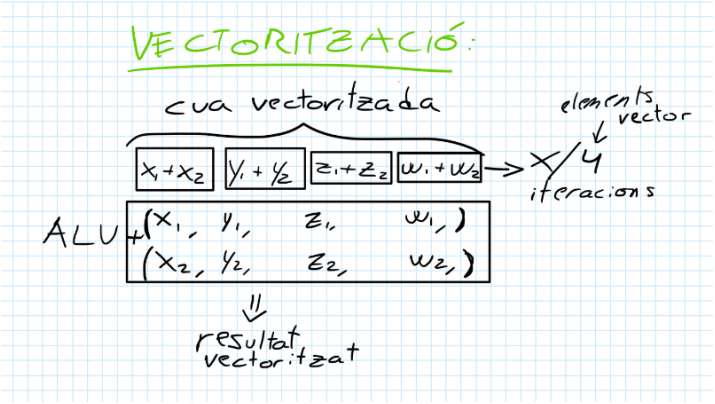
Com es veu al diagrama, cada operació (1 per cada iteració) "ocupa" la ALU, aquesta realitza el càlcul i retorna el resultat. Tenint N iteracions, això resulta en N operacions realitzades al llarg del programa. Veiem que hi ha 4 elements a la cua, i que haurem de fer 4 càlculs diferents. *Veient això, no se'm passava pel cap una millora possible pel programa.

Resulta que a cada ús de la ALU, hi havia bits sense utilitzar (marcats en blau) que era espai per una potencial suma de variables que també havien de fer la mateixa operació. La vectorització consisteix bàsicament en utilitzar el màxim espai possible en el que es fan les operacions per a operar amb altres variables simultàniament. Algunes de les limitacions que té aquest mètode poden ser:

- Les operacions han de ser del mateix tipus per a que les variables puguin existir en el mateix vector.

- Les variables han de ser del mateix tipus (float, int, etc.)
- Les variables en el vector no poden ser dependents les unes de les altres (hem de conèixer els valors de tots els elements del vector per a fer la operació)

Aquest és un esquema del concepte de vectorització que he après (el pas de les variables a la CPU):



Podem veure que, a diferencia de la cua en l'esquema anterior (longitud 4), ara tenim una cua de longitud 1 que inclou les 4 operacions des les quals volem saber el resultat. En la ALU, es veu com, el que realment està fent és obtenir un vector on el primer element és la suma de X_1 i X_2, el segon és la suma entre Y_1 i Y_2, etc.

Utilitzant vectorització també hem de tenir en compte que:

- Per a carregar els elements del vector, s'utilitzen instruccions SIMD (Single Instruction Multiple Data) com LOADV per a "empaquetar" aquests valors.
- Per a operar amb vectors, s'utilitzen instruccions SIMD com "**v**fadd" (normalment comencen per V).
- Cal "desempaquetar" el resultat que ens proporcioni la CPU, al contrari que el que hem fet al carregar-ho (utilitzant instruccions SIMD).
- El nombre d'elements del vector queda limitat pel la mida del vector establert per cada CPU. Elements = (bits max. vector)/(mida en bits de les variables)

El resultat de la implementació de la vectorització és una "speedup" substancial que sol rondar els valors de x4/x8 (depenent del tipus de variable i la mida del vector).

***Comentari personal**: Un cop après aquest concepte, he començat a notar exemples de vectorització en la via quotidiana. El meu preferit és l'exemple que les àvies porten implementant un munt d'anys: Per a fer **canelons**, sembla obvi posar-los al forn en grups de 4 o 8, seria absurd posar-ne un cada 20 minuts i no fer servir l'espai del forn sense aprofitar (Sempre que es vulgui cuinar tots els canelons a la mateixa temperatura i temps). Per a vectoritzar canelons també hem de posar-los a la mateixa safata i traient-los per a compartir-los.

P7B. Memòria Cache

Fins ara, havíem vist que per a operar amb una variable guardada en qualsevol altre lloc que no sigui un registre, accedíem a memòria (MM) fent servir un LOAD, i guardàvem dades amb l'STORE. Amb el "descobriment" de la memòria cache tot això canviava. Apareix una memòria que a la que podem accedir a les dades més ràpidament, però aquesta té molta menys capacitat.

En aquesta pràctica vam comprovar el funcionament d'un nivell de la piràmide de jerarquia de memòria molt important, i que cal tenir en compte per a comprendre per què un programa s'executa diferent fent ús d'aquesta.

El funcionament de la cache és un sistema de "preferències" en el sistema de jerarquia de memòria. Quan la CPU vol operar amb una variable mirarà si existeix en L1 de la cache. Si no la troba, la busca a L2, i el mateix amb L3. Si la variable no es troba en cap de les memòries cache, la CPU l'ha de buscar a la memòria principal (RAM). Això és perquè la memòria cache és més ràpida que la memòria principal (però també més petita).

Amb aquest **coneixement**, la **informació sobre el processador** i un **programa** donat; en aquesta pràctica vam poder predir les fallades a cache que es produirien, i vam experimentar amb diferents programes i característiques de CPU variades.

***Comentari personal**: Al començar aquest tema, el primer que se'm va venir a la ment va ser la cache del telèfon mòbil i les desenes **d'aplicacions que "netegen" la cache**). Sempre m'he preguntat si aquestes aplicacions realment "netegen" la cache del teu mòbil, com ho fan, i si serveix d'alguna cosa.

Després d'aprendre la teoria i de les pràctiques puc extreure dos conclusions d'aquesta idea:

- **Què i com ho fan?** Aquestes aplicacions realment son capaces de fer la funció que anuncien. Simplement esborren ell contingut de cada bloc de memòria del cache, i si vas a les opcions del mòbil pots comprovar que realment compleixen aquesta funció.
- **Son efectives?** Amb el coneixement que tinc avui en dia, concloc que aquestes aplicacions resulten en un **pitjor rendiment del telèfon** els minuts després d'esborrar la cache per aquesta raó: Al esborrar tota la cache pot semblar que tot està més organitzat, però realment és un problema ja que quan un programa demani carregar unes dades, trigarà més temps en obtenir-les ja que no ho trobarà a cache i haurà d'accedir a la memòria principal (Fallada en fred).

En resum, penso que aquestes aplicacions perjudiquen en el rendiment del telèfon; de cara a l'usuari com més espai hi hagi ocupat a la cache, pensarà

que funciona pitjor, quan és el cas contrari (es triga el mateix en intercanviar un bloc a cache que una fallada en fred). **És l'equivalent a tirar tot el menjar de la nevera per a que es vegi més bonica, amb l'inconvenient d'haver d'anar al supermercat cada cop que trobis a faltar alguna cosa.**

Reflexió personal:

Penso que per mi aquesta assignatura ha sigut una d'aquelles difícils, però de les més útils i interessants de totes, sense dubte. Sé que faré servir conceptes apresos en aquest curs en els meus programes (i si m'ho fa una IA, sabré per què ho ha fet). El fet de saber els conceptes que ha après, considero que aporta a tot estudiant un grau de control molt útil per als seus projectes, a part d'estalviar temps i recursos. Espero que, després d'aquest curs, els meus companys i jo, li perdem la por al llenguatge ASM, o almenys a C (és a dir, declarar variables i compilar). També he reforçat coneixement de l'arquitectura de CPU, l'ús del llenguatge C, o simplement a l'anàlisi i comprensió de programes ja escrits.

Definitivament també he notat la extrapolació d'aquesta assignatura a la vida quotidiana amb, per exemple, el *fast-thinking*, amb el qual he fet broma amb els companys, però m'ha ajudat a comprendre comportaments que tinc a l'hora de resoldre problemes (inclosos fora de la universitat).