A rendszermag finomhangolása

Assembly nyelvű programrészletek akár negyvenszázalékos sebességnövekedést is eredményezhetnek az Alpha-processzorokon.

A nagy teljesítményű kiszolgálók vezető fejlesztője az API Networks. Vevőinkkel együtt kifejezetten érzékenyek vagyunk a rendszer teljesítményére. Legtöbb vásárlónk Alpha-alapú rendszert futtat, ezért módfelett odafigyelünk a kulcsfontosságú nyílt forráskódú összetevőkre és csomagokra, hogy a gép által nyújtott teljesítmény ne maradjon kihasználatlan.

A kereskedelmi termékek zárt forrású világában a programok kiadása után felfedett hibák és a további fejlesztések mindig csak a következő változatban jelennek meg. A nyílt forrású programok világában a két fejlesztés közti idő órákban vagy napokban mérhető. Ráadásul a finomhangolásra szolgáló megoldásokat a minél fejlettebb közösségi tudásbázis létrehozása érdekében hamar elterjesztik. Az előző szempontok figyelembevételével vizsgáltuk meg a Linux-rendszermag assembly nyelvű programrészleteit. Kutatási témánk a legújabb Alpha 21264 (más néven ev6) processzor mikrofelépítésének lehető legjobb kihasználása volt.

Felfedeztük, hogy ha a 21264 tulajdonságait a Linux-rendszermag egyes felületfüggő részeinek átírásával kihasználjuk, jelentős sebességnövekedést érhetünk el, amely bizonyos feladatoknál akár negyven százalékig is elmehet. Írásunkban bemutatjuk, hogyan és miért működnek ezek a továbbfejlesztések, így tapasztalatainkat mások is felhasználhatják Alpha-alkalmazásaik felgyorsítására. Ezek a finomhangolások például egy Alpha 21264-en futó levelezőkiszolgálót nagyobb terhelés elviselésére teszik képessé, a teljesítmény pedig átlépheti azokat a határokat, amelyeknél a gép válaszideje általában megnövekedne. A végfelhasználók a rendszert gyorsabbnak fogják érezni, és a rendszergazdák szintén elégedettek lesznek, mert nem kell új gépet venni, mégha a forgalom növekszik is.

A Linux-rendszermagban a teljesítményre nagy befolyással bíró programrészleteket gondosan (és kényelmes módon) külön felületenként kezelik, és többnyire assembly nyelven valósították meg. Számos fejlesztő munkálkodik a rendszermag teljesítményének algoritmikus továbbfejlesztésén, emellett jó páran foglalkoznak azzal is, hogy megfelelő érdeklődés, tudás és tapasztalat birtokában az assembly programrészleteket finomhangolják, ezáltal a processzorból a lehető legnagyobb teljesítményt hozzák ki. A legtöbb fejlesztő az x86-os rendszerekhez tartozó kódot fejleszti, a megfelelő Alpha-kódhoz azonban előttünk jó ideje senki sem nyúlt.

A szájhagyományból és a mérésekből arra következtetésre jutottunk, hogy a web- és más hálózati kiszolgálók futtatásakor a rendszer aránytalanul sok időt tölt a Linux-, illetve az Alpha-rendszermagban. Mivel a teljesítményre nagy befolyással bíró programrészletek száma a Linux-rendszermagban viszonylag kicsi (20–30 assembly nyelvű programrész), átfésülésük érdemesnek tűnt annak kiderítésére, hogy a kódokat vajon a 21264 figyelembe vételével írták-e meg. A programrészek gyors átolvasása után azonnal kiderült, hogy gondosan, kézzel megtervezett kódról van szó, amelyet az Alpha-processzorok előző nemzedékéhez, a 21164-eshez (más néven ev5) készítettek.

A különbség jelentéktelennek tűnhet, azonban a 21264-es lényegesen eltér a 21164-től a mikrofelépítés (lapkamegvalósítás) szintjén, ennek köszönhetően azonos órajelen megközelítőleg kétszeres a teljesítménye. Mielőtt a teljesítménynövekedés részletezésébe bocsátkoznánk, célszerű áttekinteni a 21164-es és a 21264-es közötti legfontosabb különbségeket (*lásd 1. táblázat*).

Miután felismertük, hogy a gondos újraírás jelentős teljesítménynövekedést eredményezhet, hozzákezdtünk a Linux-rendszermag Alpha assembly nyelvű kódrészleteinek finomhangolásához. (Akadtak kísérletek annak a feladatnak dinamikus és önműködő megoldására, hogy a régebbi Alpha-processzorokra fordított binárisokat az újabb Alpha processzorokon hatékonyan lehessen futtatni, de ez a téma jelentősen túlmutatna írásunk keretein.)

Közelebbről nézve világossá vált, hogy körülbelül 20 programrészt a linux/arch/alpha/lib, 4–6 programrészt pedig a linux/include/asm.alpha könyvtárban szükséges átírni ahhoz, hogy teljesen kihasználjuk a 21264-es tulajdonságait.

1. táblázat **Az Alpha 21164-es és 21264-es processzorok összehasonlítása**

21164 (ev5, ev56)	21264 (ev6, ev67)
Kétutas	Négyutas
Egyszerű elágazásvizsgálat	Fejlett elágazásvizsgálat
Közvetlen leképezésű átíró gyorstár	Kétszeresen asszociatív gyorstár
Egyszerű utasításütemezési szabályok	Bonyolult utasításütemezési szabályok
	A sávszélesség nagyjából kétszerese a 21164-esének
	Új utasítások: WH64, CTLZ, CTTZ

A forrás fejlesztése

Belső nézőpontból szemlélve a 21264-es lényegesen összetettebb és hatékonyabb processzor, mint a régebbi 21164-es.

A különbségekből adódóan a teljesítmény növekedésének alapját a következő (később részletezett) átalakítások képezik:

- a kód átütemezése, hogy a processzor ne álljon meg utasításbetöltés közben;
- az elágazások elkerülése és az elágazások célcímeinek helyes megválasztása;
- ahol csak lehet, az ismétlési csapdák (replay traps) elkerülése;
- a 21264-es utasításkésleltetési és ütemezési szabályainak alkalmazása;
- a 21164-esben és a 21264-esben meglévő, eddig használaton kívüli utasítások alkalmazása;
- az 21264-esben újonnan megjelent utasítások használata és;
- lehetőség szerint bizonyos utasítások mellőzése a 21264-en.

Az utasításbetöltés leállásának (fetch stalling) megakadályozása

Ha az utasításbetöltés leállását el szeretnénk kerülni, biztosítanunk kell, hogy az utasítások a betöltési tömbben ne próbáljanak egy vagy több végrehajtóegységnél a rendelkezésükre álló kapacitásnál többet igénybe venni. A 21264-esnél az is fontos, hogy az elágazások célcímei a betöltési tömb határára legyenek igazítva. Mivel négy utasítást tölt be egyszerre, a 0mod4 utasításokat igazítani kell a csomópontokhoz, ami egyenértékű a 0mod16 igazításával.

Az elágazásokból származó lassulás elkerülése

A 21264-esen különösen fontos, hogy elkerüljük az elágazásokból származó lassulást. A bonyolult, tanítható elágazás-feltérképező beépített logika csak akkor működik hatékonyan, ha csupán egyetlen vezérlésátadó utasítás található a betöltési tömbben (a „négyes csomagban”). A 21164-eshez igazított rendszermagban több vezérlésátadó utasítás számos helyen előfordult egy négyes csomagon belül. Ráadásul az elágazások célcímei 8mod16 címekre voltak igazítva, amely gyakran azt eredményezte, hogy a cél címkéje a négyes csomag közepére került. Ezek az utasítássorozatok elég jól futnak a 21164-esen, ám a 21264-esen viszonylag lassúak.

Az ismétlési csapdák elkerülése

Ismétlési csapda akkor lép fel, amikor a processzor a soros feldolgozáshoz kénytelen visszaállítani a memória állapotát, kikényszerítve bizonyos memóriaterületek elérését, vagy amikor különböző méretű hozzáférési kérések érkeznek ugyanarra a memóriaterületre. A módosított programrészletekben ilyen gond nem akadt, ebből a szempontból nem kellett módosítani.

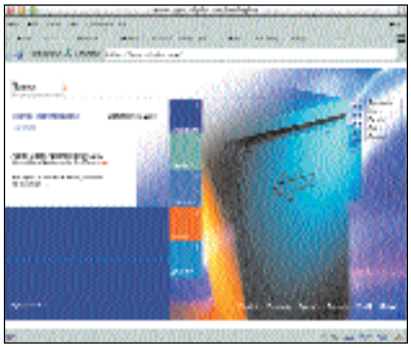
A 21264-es utasításainak késleltetése

A 21264-es utasítások ütemezésének szabályai túl bonyolultak ahhoz, hogy itt közöljük őket, akit azonban érdekelnek a részletek, a „21264 Compiler Writer’s Guide” című írásban utánanézhet.

Igénybe vehető, de használaton kívüli utasítások

A bájt- és szóméretű betöltések, valamint kiírások a 21164A (ev56) processzorban jelentek meg először, de az eredeti assembly programrészletekben ezeket nem használták. Előző tapasztalataink (az alkalmazások statikus binárisainak fordítása kapcsán) arra utaltak, hogy a teljesítmény – ha kihasználjuk ezeket az utasításokat – általában 10–20 százalékkal nő. Különösen igaz ez az stw (store word – „szó tárolása”) és az stb (store byte – „bájt tárolása”) utasításokra, mert ezek olyan módon küszöbölik ki a memóriaforgalmat, hogy nem lép fel ismétlési csapda. A rendszermag finomhangolásánál is hasznosnak bizonyultak ezek az utasítások, rendszerint a nagy területmásolások végére kerültek, míg az adatmozgatás nagy részét a nyolc bájtot egyszerre mozgató betöltő és kiíró utasítások végezték.

Alpha-felületen sokféle előbetöltő utasítás létezik. Az előbetöltő utasítások arra készítetik a memóriaalrendszert, hogy egy memóriatömböt későbbi feldolgozás céljából az adatgyorstárban helyezzen el. Ezek általában nem jelennek meg a lefordított kódban, mivel néhány fordítóprogram elég ismerettel rendelkezik ahhoz, hogy létrehozásuk szükségtelenné váljon. Például a Compaq fordítóprogramja létrehozza az előbetöltő utasításokat. Amikor nagy mennyiségű adatot kell mozgatni, az assembly nyelven programozó használhatja az előbetöltést (és ajánlatos is használnia). A gcc-ben az __asm__() lehetővé teszi, hogy a programozó a program kulcsfontosságú pontjaiba illessze be a megfelelő előbetöltő utasításokat, amennyiben a program megírása tiszta assemblyben nem kívánatos. Mivel ezek az utasítások csökkenthetik vagy megakadályozhatják az adatgyorstár leállását, használatukkal jelentős sebességnövekedés érhető el.



Újonnan elérhető utasítások

A 21264-es az első olyan Alpha-megvalósítás, amely támogatja a következő három, a teljesítménynövelés szempontjából hasznos utasítást: CTLZ, CTTZ és WH64.

A CTLZ és a CTTZ utasítások megszámolják egy 64 bites regiszter kezdő, illetve a végen elhelyezkedő nulláinak számát, de a karakterlánc-műveletek végrehajtása során is hasznosak. Amikor a program mintakeresést magában foglaló karakterlánc-műveletet hajt végre (az strlen a NULL-ra keres rá), gyakran előfordul, hogy a mintakeresés bájtnagyságú indexe egy nyolcbájtos regiszterben van. A CTTZ nélkül körülbelül tíz utasítást kellene végrehajtani, többek közt több CMOVxx-t (feltételes áthelyezés), hogy ezt az indexet meghatározzuk. Az eredmény egyrészt a kód méretének csökkenése (mindig hasznos), másrészt kevesebb órajelciklus szükséges a karakterlánc-műveletek végrehajtásához. Akad még néhány alacsony szintű fájlrendszerművelet, ahol ezek az utasítások ugyancsak jól használhatók a bitmezők lyukainak megkereséséhez.

A WH64 (write hint for 64-bytes – „felkészülés 64 bájt írására”) a memória-alrendszert értesíti, hogy a közeljövőben egy megadott 64 bájtos terület kerül kiírásra. A processzor ezt az értesülést átadhatja a memória-alrendszernek, amely érvényteleníthet a célterület tartalmát, és megtakaríthat néhány memóriaciklust a memóriaállapot folytonosságának fenntartásával. Mivel a folyamatok váltása nagy mennyiségű adat mozgatásával jár a memória egyik részéből a másikba,

2. táblázat **Terheléspróba eredményei a 2.2.18-as rendszermaggal**

2.2.18 fordítása	make -j 1	make -j 2	make -j 4	make -j 8
Foltozatlan: rendszermag	27,7	35,0	36,3	36,8
Foltozott: rendszermag	23,9	30,8	32,3	32,7
Gyorsulás	15,9%	13,6%	12,4%	12,5%

3. táblázat **Terheléspróba eredményei gcc-2.95.3-mal**

gcc-2.95.3 fordítása	make -j 1	make -j 2	make -j 4	make -j 8
Foltozatlan: gcc	50,2	65,3	67,4	70,2
Foltozott: gcc	43,7	56,8	59,1	61,3
Gyorsulás	14,9%	15,0%	14,0%	14,5%

4. táblázat **Terheléspróba eredményei a 2.4.2-es rendszermaggal**

2.4.2 fordítása	make -j 1	make -j 2	make -j 4	make -j 8
Foltok eltávolítva: rendszermag	22,5	23,1	23,3	23,6
Alap (foltozott): rendszermag	20,2	20,7	20,3	20,3
Gyorsulás	9,8%	11,6%	14,8%	16,3%

5. táblázat **Terheléspróba eredményei a gcc-vel és a 2.4.2 rendszermaggal**

gcc fordítása	make -j 1	make -j 2	make -j 4	make -j 8
Foltok eltávolítva: gcc	44,9	48,3	50,2	51,1
Alap (foltozott):	39,8	40,8	42,4	43,6
gcc				
Gyorsulás	12,8%	18,4%	18,4%	17,2%

6. táblázat **Terheléspróba eredményei az UP1000 rendszeren 2.4.0-test6-os rendszermaggal**

2.4.0-test6 fordítása	make -j 1	make -j 2	make -j 4	make -j 8
Foltozatlan: rendszermag	25,0	25,5	26,7	34,4
Foltozott: rendszermag	18,0	18,2	18,7	26,1
Gyorsulás	38,8%	40,1%	42,8%	31,8%

7. táblázat **Terheléspróba eredményei az UP1000 rendszeren gcc-vel és 2.4.0-s rendszermaggal**

gcc fordítása	make -j 1	make -j 2	make -j4	make -j 8
Foltozatlan: gcc	74,0	57,8	62,5	71,5
Foltozott: gcc	63,7	47,2	52,4	61,2
Gyorsulás	16,2%	22,5%	19,3%	16,8%

minden sebességnövekedés jól jön a rendszermag és a felhasználói memória között. A másik terület, amely nagymértékben a memória-memória forgalmának sebességétől függ, az operációs rendszerbeli programok betöltési ideje. A program bitjeit mind le kell képezni, és az összes kinullázott memóriába (a .BSS végrehajtható állományokban) nullákat kell írni.

Elkerülendő utasítások

A 21264-es a CMOVxx feltételesáthelyezés-utasítást úgy valósítja meg, hogy azt a processzor belsejében két külön részre bontja. Ennek eredményeként a CMOVxx utasítás késleltetése legalább két órajelciklus, de akár öt is lehet, attól függően, hogy az adott betöltési tömbben hány CMOV található. Bizonyos esetekben sebességnövekedés érhető el, ha a CMOV utasításokat jól kiszámítható feltételes elágazásokkal helyettesítjük. Jó ökölszabálynak tűnik, hogy amennyire csak lehetséges, csökkentsük a CMOV-utasítások számát.

Az adatgyűjtés és a kipróbálás módszere

Az adatokat egy API Network CS20 kiszolgálóról gyűjtöttük. A gép kiépítése: két 833 MHz-es processzor 4 MB DDR gyorstárral, 1 GB SDRAM és Ultra-160 SCSI lemez. Két terhelési próba futott le: ötször a 2.2.18 rendszermag fordítása és ötször a gcc-2.95.3 fordítása. Az átlagos rendszeridőt (`/usr/bin/time -p`) vettük figyelembe a `make` különböző párhuzamosításai mellett (lásd a *2.* és a *3. táblázatot*). Hasonló kísérletet végeztünk a 2.4.2 rendszermaggal (az összes sebességnövelő folt használatával). Az eredményeket a foltozatlan 2.4.2 rendszermaghoz hasonlítottuk, amelyből a legtöbb (de nem az összes) teljesítményfokozó változtatást eltávolítottuk.

Ezt a kísérletet eredetileg egy API Network UP1000 alaplappal rendelkező rendszeren hajtottuk végre, amelyben 700 MHz-es processzor volt 4 MB gyorstárral, 128 MB SDRAM és IDE lemez. Ismét öt rendszermag- és öt gcc-fordítást futtattunk, majd feljegyeztük a futási idők átlagát. A 2.4.0-test6-os rendszermagot foltokkal és anélkül is használtuk.

A szerény kiépítettségű rendszeren (az UP1000-n) a teljesítménynövekedés látványos, abban az értelemben, hogy csökkent a rendszermagban töltött idő, és bizonyos tevékenységeknél (a rendszermag fordítása) a sebességnövekedés a negyven százalékot is elérheti. A nagyvonalúbban kiépített CS20-as rendszeren a mért terheléseknél következetesen 14-15 százalékos sebességnövekedést észleltünk.

Az UP1000 és a CS20 közötti különbségeket a memóriának tulajdonítottuk: az UP1000 800 MB/s átviteli sebességgel és 64-bites sínnel, míg a CS20 2,65 GB/s átviteli sebességgel és 256-bites sínnel rendelkezik.

Összegzés

Ilyen vagy olyan formában az összes újraírt programrész része a 2.4.2 rendszermagnak (bizonyosakat ismét újraírtak). A 2.2.17-eshez is készítettünk foltot, és vállalatunk weboldalán a ➲ http://www.api-networks.com/products/downloads/developer_support/ címen a „Performance” pont alatt elérhetővé tettük. További erőfeszítések eredményeként a fejlesztések bekerültek a `glibc`-be, és ott segítik a felhasználó módban futó programok sebességnövelését.

Rick Gorton az API NetWorks szakembercsapatának tagja. A Linuxszal először az eredeti 32-bites Alpha-változat (BLADE) formájában találkozott. 1992 óta fejleszt Alphára bináris fordítókat, teljesítménynövelőket és egyéb binárisokat kezelő eszközöket. A rick.gorton@api-networks.com címen érhető el.