Párhuzamos programok fejlesztése PVM könyvtárral (2. rész) Szegény ember szuperszámítógépe

Folytassuk ott ahol legutóbb abbahagytuk és építsük meg életünk első PVM klaszterét. Ha ezzel megvagyunk gyorsan számoljunk is rajta valami érdekeset...

© Kiskapu Kft. Minden jog fenntartva

Egy *PVM* klaszter építéséhez a legegyszerűbb recept a következő:

- Vegyünk néhány hálózatba kötött linuxos gépet
- Telepítsünk rájuk *OpenSSH*-t és *PVM*-et
- Hozzunk létre rajtuk egy adott azonosítóval rendelkező felhasználót
- A létrehozott felhasználó számára engedélyezzük a jelszómentes *SSH*-t.

Előkészítés

A *PVM* telepítése a szokásos módon történhet, erről az előző cikkben már volt szó, ismétlésképpen a lépések egy *Debian* alapú rendszerben a következők:

```
apt-get install pvm
apt-get install pvm-dev

mkdir ~/pvm3
mkdir ~/pvm3/bin
mkdir ~/pvm3/bin/LINUX
```

Ha minden jól ment, a `pvm` parancs kiadásával ekkor el kell indulnia a *pvm* démonnak.

```
pvm>
```

A `conf` parancs kiadásával megtekinthető a „klaszter” jelenlegi konfigurációja:

```
pvm> conf
conf
1 host, 1 data format
    HOST    DTID   ARCH   SPEED   DSIG
    Lorien 40000 LINUX    1000
                          ➥0x00408841
pvm>
```

A kapott listában a klaszterben szereplő gépek nevei láthatók. Új gép hozzáadása az add paranccsal történik, ahhoz azonban, hogy ezt könnyen és gyorsan megtehessük szükség van néhány kisebb beállításra a gépeken.

Jelszó nélküli SSH

Új gép hozzáadásánál a *PVM* démon megpróbál *SSH*-n keresztül csatlakozni a célgéphez és azon egy újabb *PVM* démont indítani. Ha a célgépet a felhasználó nem tudja jelszó nélkül elérni, akkor fárasztó gépelésre van szükség, ami ugyan biztonságos belépést biztosít, a munkát azonban nagyban megnehezítheti.
Az *OpenSSH* lehetőséget nyújt kulcs alapú azonosításra is. Ekkor adott gép adott felhasználója publikus és privát *RSA* vagy *DSA* kulcsokat generál, majd a publikus kulcsot elérhetővé teszi egy másik gép (ezentúl célgép) számára. Ha a felhasználó ezután *SSH*-n keresztül bejelentkezési kísérletet tesz a célgépre, akkor a publikus kulcs átküldésekor a célgép *SSH* szervere kódolt üzenetet vált a felhasználó gépével, felismeri a bejelentkező felhasználót és sikeres üzenetváltással azonosítja. A kulcshoz jelszó rendelhető, ez azonban nem kötelező, sőt *PVM* klaszter építésekor kifejezetten hasznos lehet ha nincsen jelszó.

A kulcs alapján történő azonosításhoz lássunk egy példát két gép között. Legyen az egyik gép ***cluster01.xyz.hu*** a másik pedig ***cluster02.xyz.hu***.
A kulcsgeneráláshoz ki kell választani azt a gépet ahonnan jelszó nélkül szeretnénk belépni a másikra, és a gép terminálján ki kell adni a következő parancsot:

```
ssh-keygen -t dsa -f .ssh/
➥id_dsa
```

Az `ssh-keygen` végzi a kulcsgenerálást, paraméterben a *DSA* rendszerű kulcs generálását kértük a *.ssh* könyvtárba. A parancs végrehajtása közben jelszót kér, ilyenkor jelszó nélküli belépéshez egyszerűen tovább kell lépni az *Enter* gomb lenyomásával. Az eredmény két fájl, az egyik a publikus (megosztandó) kulcsot a másik pedig a privát kulcsot tartalmazza.
Ha a kulcsgenerálást a ***cluster01.xyz.hu*** gépen végeztük akkor következő parancsokkal a kulcsot megoszthatjuk a másik géppel (***cluster02.xyz.hu***):

```
scp .ssh/id_dsa.pub
➥cluster02.xyz.hu:.ssh
ssh cluster02.xyz.hu
(itt még kér jelszót)
cat .ssh/id_dsa.pub >>
➥authorized_keys2
chmod 640 authorized_keys2
rm .ssh/id_dsa.pub
```

Ezután a ***cluster01.xyz.hu*** gépről belépve az *SSH* már nem kér jelszót. A *PVM*

© Kiskapu Kft. Minden jog fenntartva

1. Lista A master forráskódja

```
/* master.c */
#include <stdio.h>
#include <stdlib.h>
#include <pvm3.h>

/* az inditando slave folyamatok szama
(a pontossag) */
#define NUM_SLAVES 32

int main ()
{
    int mytid, slave_tids[NUM_SLAVES + 2];
    int i;
    int num_slaves;
    double x;
    double partial_sum;
    double result;


    /* sajat tid lekerdezese, belepes a PVM-be */
    mytid = pvm_mytid();

    /* slave folyamatok inditasa */
    num_slaves = pvm_spawn("slave1", (char **)0,
➥PvmTaskDefault, "", NUM_SLAVES, &slave_tids[1]);

    if (num_slaves != NUM_SLAVES) {
        fprintf(stderr, "HIBA: A futtatas nem
➥lehetseges. Keves bevonhato processzor.\n");
        pvm_exit();
        exit(-1);
    }

    /* a csatorna elso es utolso eleme a master */
    slave_tids[0] = slave_tids[NUM_SLAVES + 1] =
➥mytid;

    for (i = 1; i < NUM_SLAVES+1; i++) {
        /* az uzenetkuldes inicializalasa */
        pvm_initsend(PvmDataDefault);
        /* az elozo processzor tid-je */
        pvm_pkint(&slave_tids[i - 1], 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);

        pvm_initsend(PvmDataDefault);
        /* a kovetkezo processzor tid-je */
        pvm_pkint(&slave_tids[i + 1], 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);

        pvm_initsend(PvmDataDefault);
        /* index a szamitashoz */
        pvm_pkint(&i, 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);
    }

    x = 5;
    partial_sum = 0;

    /* x bedobasa a csatornaba */
    pvm_initsend(PvmDataDefault);
    pvm_pkdouble(&x, 1, 1);
    pvm_send(slave_tids[1], 0);

    /* a kezdo reszosszeg bedobasa a csatornaba*/
    pvm_initsend(PvmDataDefault);
    pvm_pkdouble(&partial_sum, 1, 1);
    pvm_send(slave_tids[1], 0);

    /* x kiolvasasa a csatornarol */
    pvm_recv(slave_tids[NUM_SLAVES], -1);
    pvm_upkdouble(&x, 1, 1);

    /* az eredmeny kiolvasasa a csatornarol */
    pvm_recv(slave_tids[NUM_SLAVES], -1);
    pvm_upkdouble(&result, 1, 1);

    /* kiiras */
    printf("exp(%f) = %f \n", x, result);

    /* slave taszkok "leallitasa" */
    for (i = 1; i < NUM_SLAVES+1; i++) {
        pvm_kill(slave_tids[i]);
    }

    /* kilepes */
    pvm_exit ();
    return 0;
}
```

klaszter felépítése most már gyerekjáték. Válasszunk ki egy központi gépet, ahol a programokat elindítjuk majd. Ha ezen a gépen publikus kulcsot generálunk és azt megosztjuk a leendő klaszter többi gépével akkor a *PVM* démon gond nélkül fel tudja építeni a kapcsolatot jelszó kérése nélkül. A központi gépen ezután hozzunk létre egy szöveges fájlt (például *hosts.txt*), melynek tartalma a klaszter többi gépeinek neve, valahogy így:

```
cluster02.xyz.hu
cluster03.xyz.hu
...
cluster10.xyz.hu
```

Most indítsuk el a *PVM* démont a *host* fájllal és a prompt megjelenése után adjuk ki a `conf` parancsot:

```
pvm hosts.txt
pvm>conf
10 hosts, 1 data format
        HOST    DTID    ARCH SPEED DSIG
cluster01  40000  LINUX 1000
                              ➥0x00408841
cluster02  80000  LINUX 1000
                              ➥0x00408841
. . .
cluster10  280000 LINUX 1000
                              ➥0x00408841
pvm>
```

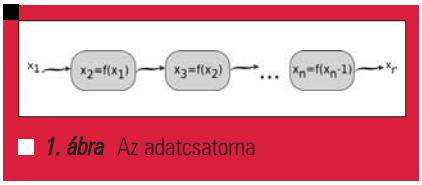
A klaszter ezzel bevetésre kész, kihasználni azonban már nem ilyen

© Kiskapu Kft. Minden jog fenntartva

könnyű. Most megnézünk egy egyszerű programot, ami egy általánosan elfogadott párhuzamos tervezési mintára épül és könnyen átírható komolyabb feladatok megoldásához.

Az adatcsatorna

Az adatcsatorna az egyik legkönnyebben megérthető és alkalmazható párhuzamos tervezési minta. Olyan számításokra használható, ahol az eredmény egymáshoz nagyon hasonló függvények kompozíciójával kapjuk. A párhuzamosítást a kompozíciós lépések szétdarabolásával lehet elérni. Az *1. ábra* egy egyszerű adatcsatornát ábrázol. Az ábrán látható dobozok a csatorna elemei és egyben a kiszámítás egyes lépéseit szemléltetik. Jó látható, hogy az x_1 bemenethez a csatorna végén az x_n kimenet tartozik. Az x_n kiszámításához az *f* függvényt kellett alkalmazni (n-1)-szer. Általában ez annyival bonyolultabb, hogy nem egyetlen függvény építi fel a csatornát, hanem pontosan (n-1) darab. Ha az adatcsatorna egyes elemei folyamatok és mindegyik egy külön processzoron fut, akkor a csatorna első elemébe bedobva egy kezdőértéket, nem kell kivárni a végeredmény kiszámítását, elegendő az első elem



1. ábra Az adatcsatorna

felszabadulására várni, ugyanis amint az továbbadta részeredményét ismét képes bemenetet fogadni. Így egy m lépésben kiszámítható függvény esetében n elemre szekvenciális esetben (n*m) lépést kellene végrehajtani, míg párhuzamos esetben mindössze (n+m)-et. A gyorsulás tehát igen nagy.

2. Lista – A slave kódja

```
/* slave.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pvm3.h>

int main ()
{
    int mytid;
    int parent_tid;
    int index;
    double partial_sum;
    int last;
    int next;
    double x;
    double factab[170];
    int i;

    /* a sajat tid lekerdezese, belepes
➥a PVM-be */
    mytid = pvm_mytid();
    /* szulo tidjenek lekerdezese */
    parent_tid = pvm_parent();

    /* faktorialis prekalkulacio */
    factab[0] = 1;

    for (i = 1; i < 171; i++) {
        factab[i] = factab[i - 1] * i;
    }

    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* a csatorna elozo eleme */
    pvm_upkint(&last, 1, 1);


    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* a csatorna kovetkezo eleme */
    pvm_upkint(&next, 1, 1);

    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* index */
    pvm_upkint(&index, 1, 1);


    /* fogadas -> szamolas -> kuldes */
    while (1) {
        /* a csatorna elozo elemetol */
        /* x */
        pvm_recv(last, -1);
        pvm_upkdouble(&x, 1, 1);

        /* a reszosszeg */
        pvm_recv(last, -1);
        pvm_upkdouble(&partial_sum, 1, 1);

        /* szamolas */
        partial_sum += pow(x, index-1) /
➥factab[index-1];

        /* x tovabbkuldese */
        pvm_initsend(PvmDataDefault);
        pvm_pkdouble(&x, 1, 1);
        pvm_send(next, 0);

        /* a reszosszeg tovabbkuldese */
        pvm_initsend(PvmDataDefault);
        pvm_pkdouble(&partial_sum, 1, 1);
        pvm_send(next, 0);
    }

    pvm_exit();
    exit(0);
}
```

Példaként nézzünk meg egy adatcsatorna megvalósítást *PVM*-ben. A csatorna az exp függvényt fogja kiszámítani adott pontosságig. Az exp függvény az e (Euler-féle szám: 2,718281...) x-edik hatványa. Kiszámítására létezik egy végtelen összeg alak: $x^0/0! + x^1/1! + \ldots + x^n/n! + \ldots$ amiből az ***exp(x)*** adott pontosságú értéke egy részletösszeg. Ennyi matek után az olvasó akár meg is ijedhet, a feladat megoldása azonban nagyon egyszerű. Most is két programunk van, egy master és egy slave.
A fenti program felépíti a csatornát és bedobja az elején az 5-ös számot, majd (miután elkészült) kiveszi az eredményt a csatorna végéről.
Az egyszerűség kedvéért most erre az egyetlen számra nézzük meg a működést, azonban komolyabb munkára is fogható a program, ha bemenetét például egy fájlból veszi és az nem egyetlen szám.
A csatorna felépítését egyszerű láncolással lehet megoldani, amihez az egyes elemeknek el kell küldeni az előző és a következő elem azonosítóját. Az azonosítón kívül az egyes csatornaelemek megkapják még a kiszámításhoz az indexüket is. Ebből az indexből határozza majd meg az adott elem, hogy a részletösszeg hányadik tagját alkotja.
Hogy mindez világosabbá váljék, lássuk a csatornát felépítő slave kódját (*2. Lista*).
A slave program miután megkapta a sorban előtte és mögötte álló elemet azonosítóját és saját indexét, vár az eddig kiszámolt eredményre és a bemenetre. E kettőre lesz szüksége ahhoz, hogy a részletösszeg következő tagját kiszámolja és továbbküldje.
A programok használatba vételéhez le kell fordítani őket, itt nincs semmi új, emlékeztetőül a fordítás menete:

```
gcc master.c -o master1 -lpvm3
gcc slave.c -o slave1 -lpvm3
➥-lm
```

Az elkészült állományokat (***master1*** és ***slave1***) másoljuk a *~/pvm3/bin/LINUX* könyvtárba, majd a ***slave1*** programot osszuk meg a klaszter többi gépével. A megosztás *SSH*-n keresztül *SCP*-vel történhet és könnyen automatizálható, hiszen a többi gép már jelszó nélkül is elérhető.
Például:

```
scp slave1 cluster02.xyz.hu:
➥pvm3/bin/LINUX
```

Ha a megosztást az összes gépre elvégeztük, akkor a ***master1*** program elindítható. Fontos megjegyezni, hogy ha az elindított program a képernyőre is ír, akkor nem szabad *PVM*-en belülről `spawn` paranccsal indítani, mert ekkor a kimenetet nem látjuk. Az indításhoz lépjünk ki a *PVM*-ből, mely a háttérben tovább fut, és adjuk ki a master1 parancsot. Ha minden jól működik, az eredmény nem más mint ***exp(5)*** egy egészen pontos közelítése.

Bánki Horváth András
(bha@elte.hu)
Végzős programtervező matematikus hallgató vagyok az ELTE-n. Minden érdekel ami az informatikával kapcsolatos. 1997 óta vagyok aktív Linux felhasználó. Ha nem dolgozom, legszívesebben a barátnőmmel és a barátaimmal vagyok.

© Kiskapu Kft. Minden jog fenntartva