Alkalmazói programozási felület SETI-jellegű elosztott programokhoz és végrehajtó rendszer a BOINC infrastruktúrára[1]

Podhorszki Norbert és Vida Gábor
MTA SZTAKI, Párhuzamos és Elosztott Rendszerek Laboratórium
H-1518 Budapest, Pf. 63
{pnorbert, vida}@sztaki.hu

1 Bevezetés

Az utóbbi években egyértelműen a legnagyobb méretű és kapacitású számítógép-rendszereket az Interneten keresztül összekapcsolódott személyi számítógépekből alakították ki. A nagy számú PC-k által alkotott, Interneten összekötött elosztott rendszer több teraflops teljesítményű erőforrást jelent olyan alkalmazások számára, amelyeknek megfelel ez a fajta felépítés. Az ilyen rendszerben a PC-k egymással nem kommunikálhatnak és így a végrehajtható programok is csak szekvenciálisak lehetnek (kivétel lehet esetleg a duál-processzoros gépekre optimalizált többszálú program). A nagy számítási kapacitást így csak független részfeladatokra szétdarabolható problémák megoldására lehet használni.

A legjobb példa a SETI@home [1][2], ahol az alkalmazás maga egy sok évig futó adatfeldolgozó alkalmazás. A SETI@home egy tudományos kísérlet, amely több millió Internetre kapcsolt számítógép kapacitását hasznosítja a Földönkívüli Intelligencia Kutatásában (SETI, Search for Extraterrestrial Intelligence). A SETI az egész kutatási terület átfogó neve, ennek egy technikai megközelítése a rádióhullámok vizsgálata, egy keskeny frekvenciatartományban sugárzott intelligens adás keresése. Az adatok feldolgozása lényegében egymástól független: két különböző égbolt tartomány jelei függetlenek és az egy helyről érkező különböző frekvenciájú jelek is a keresés szempontjából függetlennek tekinthetők. A hatalmas adathalmaz (35 GB/nap), amelyet a teleszkópok rögzítenek, tehát könnyen szétdarabolható kicsiny (350 KB) feladatokra, amelyeken az elemző algoritmus függetlenül futtatható. A projekt célja az égbolt egyharmadának vizsgálata, amely körülbelül 39 TB-nyi adatot jelent a több évig tartó teljes megfigyelés végén. A feladat tehát ideális párhuzamos végrehajtásra.

A SETI@home projekt 1999-ben kezdett működni, 2002 júliusig 3.89 millió felhasználó töltötte le a kliensprogramot, amely a háttérben vagy képernyővédőként futva dolgozza fel az adatokat. A kliens program 175 féle operációs rendszer, processzortípus és fordítóprogram kombinációra lefordítva tölthető le és installálható. 2001 július-tól számítva egy év alatt 221 millió munkacsomagot dolgoztak fel a kliensek, ami 27.36 teraflops átlagos (és ténylegesen kihasznált) teljesítményt jelentett [3][3]. A SETI@home a világ legnagyobb szuperszámítógépének tekinthető mind a mai napig. A SETI@home

[1] A cikkben leírt kutatómunkát és fejlesztést az IHM 4671/1/2003 számú, az OTKA T042459 számú és a GVOP-3.1.1.-2004-05-0145/3.0 számú projekt támogatta.

rendszer – szokásos – hátránya, hogy egyetlen alkalmazást valósít meg, azaz nem lehetséges többféle alkalmazást végrehajtani a kialakult óriási infrastruktúrán.

A nyílt forrású BOINC-et [4] a SETI@home fejlesztő csapata tervezi és implementálja azzal a céllal, hogy a SETI jellegű és a nagyközönség számára attraktív tudományos projektek nagy számítási igényű elosztott programjainak egy közös infrastruktúrát biztosítson. Ezzel lehetővé válik, hogy több millió felhasználó számítógépén ugyanaz a szoftver fusson, és minden egyes felhasználó eldönthesse, mely projektekben óhajt részt venni anélkül, hogy különféle szoftvereket kellene installálnia és letörölnie. A BOINC kliensszoftvert futtató PC-k többmilliós hálózata így a világ legerősebb szuperszámítógépévé válhat, amelyen már nem csak egyetlen alkalmazás futhat. A BOINC rendszer felállítása és használata felettébb kényelmetlen. Hozzáértő rendszergazdáknak kell telepítenie és konfigurálnia a szervereket. A programozáshoz pedig nem ad támogatást a BOINC, ugyanis a SETI fejlesztői elsősorban a maguk igényét akarták kielégíteni vele. Más programok fejlesztői nagy lendülettel és kitartással kénytelenek saját alkalmazásaikat a BOINC igényeihez és sajátosságaihoz szabni.

Az MTA SZTAKI-ban fejlesztett DC-API (Distributed Computing API) egy egyszerű elosztott programfejlesztési felületet definiál a SETI-hez hasonló ún. Farm-számítás jellegű alkalmazások fejlesztésére. Az API elrejti az aktuális végrehajtó rendszer jellegzetességeit és a programozó csak a feladat megoldására koncentrál, azaz részfeladatokat hoz létre és részeredményeket dolgoz fel. A végrehajtás megszervezése az API implementációjának dolga. Ezzel nagymértékben megkönnyíthető a szekvenciális programozáshoz szokott programozók dolga, valamint a kész program sem specifikus, hanem végrehajtható minden olyan elosztott rendszeren (Interneten összekötött PC-alapú elosztott rendszer, másféle Grid rendszerek, klaszterek stb.) amelyhez az API-t illesztettük.

A 2. fejezetben ismertetjük az MTA SZTAKI-ban kifejlesztett DC-API-t, és leírjuk, hogy egy alkalmazás hogyan használhatja azt. A 3. fejezetben röviden ismertetjük a BOINC infrastruktúra jellegzetességeit, és aztán hogy milyen módon hajtódik végre ténylegesen a DC-API segítségével megadott számítás a BOINC infrastruktúrán. A 4. fejezetben az API-nak egyéb Grid rendszerekhez történő illesztéseit ismertetjük.

2 DC-API farm-jellegű számítási feladatok feldarabolására

A legfőbb célja a „már megint egy” programozási interfésznek, hogy a párhuzamos programozáshoz nem értő fejlesztők is könnyedén tudjanak meglévő szekvenciális kódjukból egy többgépes rendszeren futó alkalmazást készíteni. A legegyszerűbb számítási modellt támogatja az interfész, azaz feltételezzük, hogy nincs szükség információcserére az egyes gépeken futó részprogramok között. Az ilyen jellegű feladatokat farm-számításnak nevezik. Egy egyszerű – de konkrét rendszerek felett álló – interfész lehetővé teszi azonban azt is, hogy az egyszer kifejlesztett programot módosítás nélkül lehessen más rendszeren futtatni. Párhuzamos programok fejlesztésére alkalmas nagy üzenetközvetítésen alapuló könyvtárak az MPI [5][6] és a PVM [7], közös-memóriás architektúrákra használható az OpenMP [8]. Ezek azonban például egyáltalán nem működnek a BOINC infrastruktúrán, hiszen annál jóval bonyolultabb szerkezetű

programok támogatására találták ki azokat, amelyeknek nem felel meg a független PC-k halmaza.

A farm-alkalmazás részfeladatokat hoz létre, amelyek mindegyike egy-egy szekvenciális program által, adott bemeneti adatokból számítható ki. Az alkalmazásnak az a feladata, hogy definiálja a részfeladatokat (program és input páros) és hogy az egyes részfeladatokra adott eredményeket (output) feldolgozza, összeillessze. Az alkalmazást (és a programozót) nem érdekli, milyen számítástechnikai rendszer hajtja végre a részfeladatokat, hogy hogyan juttassa el a részfeladatokat oda, és hogy milyen speciális melléktevékenységekkel lehet elérni, hogy működjön is.

Az alapforgatókönyv, ahogyan egy ilyen alkalmazás ténylegesen lefut:

1. Az alkalmazást elindítja egy felhasználó
2. Az alkalmazás részfeladatokat hoz létre. Ez egy szekvenciális program megadását (kliensprogram azonosítását), a program erőforrásigényeinek (CPU-, memória-, diszk-kapacitás), argumentum listának és bemeneti adatfájlok listájának megadását jelenti.
3. A részfeladatokat a DC-API-n keresztül kell definiálni és létrehozni.
4. A DC-API helyezi el (küldi el) oda, ahol a részfeladatok ütemezése és elosztása megtörténik (Ütemező).
5. Amikor egy szabad erőforrás adódik (az Ütemező talál ilyet, vagy egy ilyen jelentkezik be nála), akkor az Ütemező egy részfeladatot erre az erőforrásra elküld.
6. Az adott részfeladathoz tartozó kliensprogramot és adatfájlokat le kell tölteni a végrehajtó erőforrásra.
7. Az erőforráson a kliensprogram feldolgozza az adatokat, amelyeket már megtalál ott helyben, és eredményül kimeneti adatokat hoz létre. Ezért lehet ez a kliensprogram egy eredeti szekvenciális program, nem kell kommunikálnia más gépekkel.
8. Az alkalmazás – a részfeladatok létrehozása után – várakozik, amíg egy eredmény vissza nem érkezik. Amikor egy részfeladat elvégződött és az alkalmazás feldolgozza az eredményét, olyankor az alkalmazás dönthet újabb részfeladatok kiadásáról, vagy akár más részfeladatok leállításáról is.
9. Az alkalmazás akkor fejeződik be, ha tényleges feladatát elvégezte a más gépeken végzett rész-számítások alapján.

A fenti leírás ráhúzható egy szimpla PC-re, egy klaszterre, egy sok független PC-ből álló halmazra illetve a Grid rendszerekre. Mégis, nincs olyan programozási API, amely minden ilyen rendszeren működőképes. A következőkben a DC-API-t ismertetjük, amely minden architektúrára megvalósítható.

A DC-API feladatai:

- Alkalmazás inicializálása
  dc_init(projektnév, alkalmazásnév, konfigurációs fájl)

- Részfeladat létrehozása
  dc_createWU(kliensprogram, argumentum)
  dc_setInput(inputfájlok listája)
- Részfeladat elküldése végrehajtásra
  dc_submitWU(részfeladat)
- Részfeladat megszakítása
  dc_cancelWU(részfeladat)
- Eredményre várakozás (csak ellenőrzés és azonnali visszatérés, vagy blokkolt várakozás addig, amíg nincs eredmény)
  dc_checkForResult(timeout idő, eredményfeldolgozó eljárás)
- Részfeladat teljes törlése az alkalmazás memóriájából
  dc_destroyWU(részfeladat)

Mint látható, ténylegesen csak néhány egyszerű feladatot kell elvégeznie az alkalmazásnak, csakis a részfeladatok megadására koncentrálva, és nem szükséges a végrehajtó rendszerrel törődnie. Az eredményekre várva egy függvényt kell definiálni, amely egy részeredményt fel tud dolgozni. Ezt a függvényt a DC-API hívja meg, amikor egy részeredmény visszaérkezik az alkalmazás szintjére.

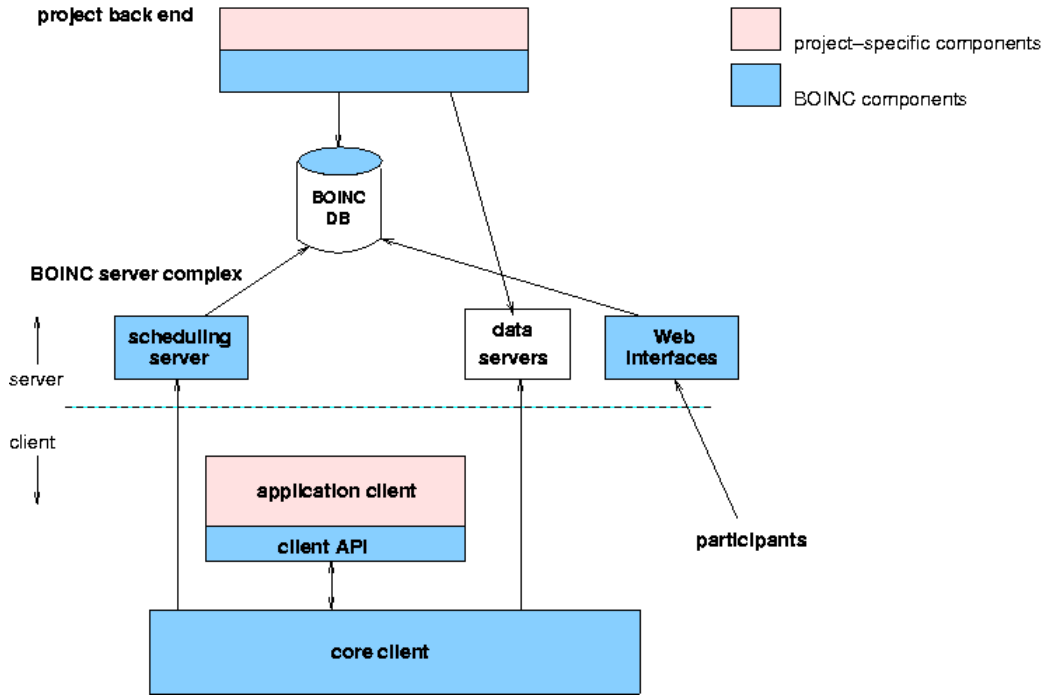
Az egyes architektúrák különbözőségeit a dc_init-ben megadható konfigurációs fájl tartalmazza, például az API-nak, a távoli rendszer eléréséhez szükséges parancsok elérési útvonala, az inputfájlok elérési útvonala, gépnevek, erőforrásmenedzserek nevei, stb. A konfigurációs fájl különböző lehet az egyes architektúrákra, de ez nem befolyásolja az alkalmazás működését, sem létrehozásának módját.

3 A DC-API működése a BOINC infrastruktúrán

3.1 A BOINC infrastruktúra röviden

A BOINC [4] felépítése a SETI@home eredeti struktúráját [3] követi, de szétválasztva az egyes belső funkciókat külön szerverekre és megengedve több szerver együttműködését egy funkcióra. Egy ún. *projekt*-et több alkalmazás kiszolgálására kell telepíteni egy BOINC szerveren. A 1. ábra szerint egy *Scheduler* szerver (ütemező) osztja ki a munkacsomagokat a kliens gépeknek, illetve jegyzi fel, ha eredmény érkezett. A *Data* szerveren tárolódnak a munkacsomagok input adatai és az eredményadatok. Mindkét fajta szerverből több is üzembe állítható, ha a projekt mérete ezt megkívánja (legalábbis a tervek szerint, mert a jelenlegi állapotban csak egy ütemező szerver üzemelhet). Ezek mellett egy különálló Webes felület (PHP nyelven készített honlap) biztosítja az emberi felhasználókkal a kapcsolatot, ahol a felhasználók regisztrálhatnak és áttekinthetik saját és mások kredit-eredményeit, illetve fórumokon társaloghatnak. Egy *projekt* az összes működéshez szükséges adatot egy relációs adatbázisban tárolja (felhasználók, munkacsomagok, eredmények, csapatok, kliens gépek, alkalmazói programok és verziói stb).

Az egyes PC-ken (klienseken) egy általános szoftver, a *Core Client* fut. Ez tartja a kapcsolatot a regisztrált projektekkel, kér munkacsomagokat a Scheduler szervertől, tölti le a programokat és adatokat a Data szerverről, illetve tölti fel oda az eredményeket és a Scheduler szervernél bejegyezteti az elvégzett munkát. Mindemellett ez a kliens biztosít egy általános célú grafikai kimenetet a projektek alkalmazásai számára, amelyek például a képernyővédő képeként ilyen módon jeleníthetik meg az aktuális munkát. Ne feledjük, hogy ezeknek a projekteknek attraktívaknak kell lenniük, hogy önkéntes emberek ezt a szoftvert futtassák a gépeiken!



1. ábra A BOINC infrastruktúra egy projekthez (több szerver is felállítható egy projekthez)

Egy tudományos projekt több alkalmazást is telepíthet a szerverein. Egy alkalmazásnak több platformon futó változata is szükséges, hogy mindenféle platformú kliens képes legyen futtatni azt. Ezen felül, hosszú távon, akár évekig futó projektekről van szó, és ezért az alkalmazás újabb és újabb verzióinak kezelése is megoldott. Az infrastruktúra intézi el az újabb verziók automatikus letöltetését a kliens gépekre, így nem kell a felhasználókat minden egyes alkalommal a webes honlapon felkérni, hogy töltsék le az új verziót.

Az erőforrásait önkéntesen felajánló felhasználó szabadon választhat a BOINC-en futó projektek között, illetve százalékos arányban meg is oszthatja gépidejét több projekt között. Minden projektnek egy hivatalos URL (a webes honlap címe) a belépési pontja, itt egy adott nevű szöveges fájl írja le a projekthez tartozó Scheduler szerverek listáját.

Erőforráskezelés

Egy igen fontos következménye van annak, hogy a SETI@home egyetlen alkalmazása és egyenméretű feladatcsomagjai helyett akármennyi alkalmazói programot és különféle erőforrás-igényű feladatot akar a BOINC támogatni. A kliensek különböző kapacitással (CPU sebesség, speciális utasításkészlet (SSE, 3DNow, stb.), memóriaméret, diszkkapacitás) rendelkeznek és nem biztosítható, hogy bármely alkalmazás bármely munkacsomagja elvégezhető legyen akármelyik gépen. Ezért a BOINC nyilvántartást vezet minden egyes kliens képességeiről, amelyet a Core Client-be épített automatikus tesztprogram állapít meg és közöl a szerverrel bejelentkezéskor. Hasonlóan, a munkacsomagok leírásában megjelennek az erőforrásigények is. A szerver ezután csak olyan kliensnek osztja ki a munkacsomagot, amely biztosan képes megbirkózni vele.

Kommunikáció

A BOINC a jól bevált HTTP protokollt használja a kliensek és a szerverek között. Az információt XML leírásban adják át egymásnak. C++-ban írt CGI programok futnak mind a Scheduler mind a Data szerveren.

Biztonsági kérdések

A BOINC fejlesztői abból a feltételezésből indultak ki, hogy mivel a kliens gépek idegenek kezében vannak, ezért azok jóhiszeműségéről nem tehető fel semmi. Ezért a projekteknek tudomásul kell venniük, hogy adataikat idegen gépekre bízzák rá, és a BOINC nem gondoskodik azok védelméről.

Bár a fenti érv ugyan nem teljesen támasztja alá, mégis ugyanezen okból a hálózati kommunikáció titkosításával sem foglalkoztak eddig a fejlesztők. Ez azt jelenti, hogy minden adat nyíltan halad át a publikus hálózaton. Véleményünk szerint várható, hogy ez meg fog változni a jövőben, már csak a felhasználói azonosító kulcs szöveges átvitele miatt is. A BOINC fejlesztői levelezőlistán már megjelent egy HTTPS kommunikációt biztosító modul igénye.

A BOINC teljesen nyitott filozófiája nem tesz különbséget a felhasználók között, ezért egy projekt nem válogathat közöttük. Ezért nincs szükség sem speciális authentikációra (a felhasználó azonosításán túl a kreditek miatt) sem authorizációra.

Szoftverkövetelmények

A BOINC szerverek Debian vagy RedHat Linux számítógépeken futnak és működésükhöz szükséges számos szabad szoftver: Apache webszerver, MySQL adatbáziskezelő, PHP, Perl és Python nyelvi interpreterek. A Core Client jelenleg a különféle Windows-okra, mindenfajta Linux-ra, ezen felül Mac OS X-re és Sun Solaris-ra érhető el. A BOINC teljes forráskódja nyilvános és így szükség esetén más platformokra is lefordítható.

3.2 A DC-API működése a BOINC infrastruktúrán

A DC-API-t legelőször a BOINC infrastruktúrán valósítottuk meg, mert elsősorban az intézményekben található számos PC kapacitását szeretnénk kihasználni, illetve mert az ilyesfajta erőforrás éppen a farm-számításokat végző alkalmazásoknak felel meg. Legelőször egy BOINC szervert kell telepíteni és abban egy projektet kell létrehozni,

amely a futtatandó kliensprogramokat tartalmazni fogja. A DC-API segítségével létrehozott alkalmazás egy részfeladata a BOINC egy munkacsomagja lesznek, amely egy adott kliensprogramból az ütemező szerveren és a bemeneti adatokból az adatszerveren áll. A BOINC lehetővé teszi, hogy többféle szekvenciális programot is elhelyezzünk egy projektben, így a farm-számítás többféle algoritmust is implementálhat, illetve használhat fel már meglévő programokat a céljainak megfelelően, vagy többféle farm-számítás is végrehajtható ugyanazon a rendszeren egy projekt alatt (egy szerver által kiszolgálva).

A munkacsomag létrehozásakor az alkalmazásnak azonosítania kell a részfeladatot kiszámító programot (ez egy névvel történik), illetve meg kell adnia a bemeneti adatokat. Ez utóbbiakat a DC-API másolja át az adatszerverre. A munkacsomagnak egyedi azonosítót kell kapni egy BOINC szerveren belül, ezt az API hozza létre, az alkalmazásnak nem kell törődnie vele. A munkacsomag elküldésekor az API hozza létre a BOINC-on belüli munkacsomagot a megadott információk alapján. Ezután a BOINC feladata, hogy a részfeladat kiszámításra kerüljön valamelyik alkalmas számítógépen és az eredmény visszakerüljön az adatszerverre.

Az eredmények megérkezését az API figyelemmel követi. Ha egy részfeladat eredménye megérkezik, az API visszamásolja az eredményfájlokat az alkalmazáshoz és meghívja az alkalmazásban a megadott eljárást, amely feldolgozza az eredményeket.

Láthatjuk, hogy az API használatával az alkalmazásnak valóban nem kell törődnie az elosztott rendszer tulajdonságaival, csak definiálnia kell a részfeladatokat és fel kell dolgoznia a visszaérkező eredményeket.

4 Más Grid architektúrák

A BOINC által támogatott, sok független PC-ből álló számítási erőforrás csak egy lehetséges architektúra, ráadásul ennek használhatósága erősen korlátozott az egyes gépek közötti kommunikáció hiánya miatt. Egy klaszterben a számítógépek egymással összekötöttek, és ezért olyan alkalmazások is készíthetők, amelyek egyes részeinek szüksége van arra, hogy más részekkel információt cseréljen. A Grid rendszerek is támogatják a párhuzamos programokat tekintve, hogy legfőképpen klasztereket és szuperszámítógépeket kötnek össze bennük. Ezért nekünk is célunk, hogy a DC-API ilyen architektúrákon is működjön, kiterjesztve ezzel az ezt használó programok felhasználhatósági körét.

Egy klaszter több PC erős hálózattal összekötött (és egy helyiségben tartott) egysége, amelynek elemei jellemzően azonosak, hardver és szoftver szempontból is. Általában egy ún. *job-manager* ütemezi a beérkező programokat a szabad gépekre, tehát egy felhasználó nem az egyes gépekhez fér hozzá, hanem a klaszterhez, mint egységhez. A farm-számítások természetesen könnyen adaptálhatók egy klaszterhez. Itt a munkacsomag, mint *job* ismeretes, amely a klaszter egy gépén végrehajtható programból és a bemenő adatfájlokból áll. A DC-API tehát egy klaszter job-manager által megkívánt formátumban definiálja a jobot és adja át neki végrehajtásra. A program lefutásáról és az eredmények elérhetőségéről ugyancsak a jobmanager tájékoztat (természetesen sokféle jobmanager sokféle nyelvezetén, de a lényegi elemek azonosak). A DC-API feladata itt is az eredmények figyelése, azok továbbítása az alkalmazás felé és annak értesítése.

A Grid rendszerek ma javarészt klaszterek és szuperszámítógépek összekötéséből áll, ahol azokat továbbra is egy helyi job-manager felügyeli, míg a Grid-ben egy globális feladatelosztó, a bróker fogadja a felhasználóktól a jobokat és választja ki a megfelelő erőforrásokat (klasztert vagy szuperszámítógépet). A bróker a jobot a helyi ütemezőnek adja át, és a végrehajtás onnatól kezdve azonos a klaszterekben megszokottnál. Ennek megfelelően semmi akadálya sincs annak, hogy a DC-API a különböző Grid implementációkon is használható legyen. A munkacsomagot itt is egy job reprezentálja, csak nem egy helyi ütemezőnek kell átadni, hanem a brókernek. Természetesen a legfőbb nehézség itt is, hogy minden rendszerrel más nyelven kell beszélni, ezért minden egyes rendszerhez különálló munka a DC-API-t illeszteni.

5 Összefoglalás

A DC-API egy egyszerűen használható interfész farm-számításokat végző alkalmazások létrehozására, amely aztán végrehajtható független PC-k hálózatán a BOINC infrastruktúra segítségével, vagy egy klaszteren, vagy akár egy Grid rendszeren minden módosítás nélkül. Az interfész használatának legnagyobb előnye, hogy nem kell hozzá semmiféle párhuzamos és elosztott programozási tapasztalat; szekvenciális programozók könnyen használhatják. Feltéve természetesen, hogy az adott feladatot meg lehet fogalmazni sok különálló és egymástól független részfeladat kiszámításaként.

Hivatkozások

[1] SETI@home: http://setiweb.ssl.berkeley.edu/

[2] E. Korpela, D. Werthimer, D.P. Anderson, J. Cobb, M. Lebofsky: SETI@home: Massively Distributed Computing for SETI. *IEEE "Computing in Science and Engineering", http://www.computer.org/cise/articles/seti.htm*

[3] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer: SETI@home: An Experiment in Public-Resource Computing, *Communications of the ACM*, Vol. 45 No. 11, November 2002, pp. 56-61.

[4] BOINC: http://boinc.berkeley.edu

[5] W. Gropp, E. Lusk, N. Doss and A. Skjellum: A high-performance, portable implementation of the MPI message passing interface standard. *Journal of Parallel Computing*, Vol. 22, No. 6, pp. 789-828, Sept. 1996

[6] MPI: Message Passing Interface Forum. http://www.mpi-forum.org

[7] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine – a User's Guide and Tutorial for Network Parallel Computing*. MIT Press, Cambridge, MA, 1994.

[8] OpenMP Application Programming Interface. http://www.openmp.org