Tudományos folyamatok gyorsítása

Nézzük meg, hogyan tehetjük hatékonnyá Matlab projektjeinket egyes részeinek C-kóddá történő fordításával!

A NASA Moffett Field-i Ames Kutatóközpontjában, a Szilíciumvölgy szívében, egy olyan csapatban voltam kutató, amelyik Linuxot használt egy igen érdekes és előremutató fejlesztésben. Az Ames-i Neuro Engineering Laboratóriumban az agy–számítógép-csatolófelületen dolgoztam, mégpedig egy olyan rendszeren, amelynek a segítségével az EEG (elektroencephalogram–agyhullám) jeleket elektronikus rendszerek és roboteszközök vezérlésére lehet felhasználni. Az volt a feladatom, hogy az elsődleges labor kutatóitól átvegyem a prototípuskódokat, majd ezekből hatékony megvalósításokat fejlesszek ki, amelyeket valós idejű adatfeldolgozással, emberi alanyokon ki lehet próbálni. Gyakran csak az algoritmusok durva vázlatát vagy egy kódtöredéket kaptam meg, és ki kellett derítenem, hogy fel lehet-e őket használni az általunk gyűjtött agyhullámadatokon.

A Matlab és a GNU Octave nevű szabad program nagy segítséget jelentett nekem ebben a munkában: lehetővé tették, hogy az adatfeldolgozásra és adatmegjelenítésre vonatkozó hatékony módszereket dolgozzak ki, amit C-ben (ne adj Isten, Fortranban) csak igazán kínkeservesen tudtam volna megoldani. A megvalósítás egyszerűsége pedig komoly szempont olyankor, amikor nagy tömegű kísérleti kóddal dolgozunk, amiket aztán vagy véglegesen felhasználunk, vagy elvetünk. Amikor az eljárás megfelelőnek tűnt, és eljött az idő, hogy komolyan elgondolkozzunk valós idejű adatfeldolgozó rendszerünkbe történő illeszthetőségén, azonnal kiderült, hogy a Matlab könnyű programozhatóságának bizony megvan az ára. Ez az ár pedig a sebesség. Egyetlen másodpercnek megfelelő adatmennyiség feldolgozása például percekig vagy akár órákig is eltarthat. Nyilvánvaló, hogy ezt nemigen használhatjuk valós idejű rendszerekben. Minden kód, ami értékesnek bizonyul, meglévő kódbázisunkhoz illeszkedve előbb vagy utóbb C-, illetve C++-kódként végzi. E két ok miatt a Matlab elég sok kódját C nyelven újra kellett írnunk.

Aki már dolgozott Matlabbel, most biztosan rögtön arra gondol, hogy „de hiszen a Matlab magától is képes C-be menteni”, vagy „mi a baj a Matlab új JIT fordítójával?”. Az új JIT fordító helyenként valóban felgyorsíthatja a kódot (ha belenézünk a leírásba, látni fogjuk, hogy jó néhány olyan helyzet létezik, amikor meg sem próbál egyszerűsíteni), azonban soha nem versenyezhet egy jól megírt, lefordított C-kód hatékonyságával. A Matlab C-exportáló felületéről pedig csak annyit, hogy a Matlab által kiírt kód pontosan ugyanolyan lassú, mint a Matlab belső környezetében futó értelmező (interpreter) kód, és csatolófelület-munkálatok nélkül igen nehéz más projektekbe illeszteni. Továbbá egyik lehetőség sem segíti a GNU Octave felhasználóit vagy azokat, akik nem tudják tartani az ütemet a költséges Matlab-frissítésekkel. Általában véve a legjobb megoldás az, ha az eredetileg Matlab alatt létrehozott algoritmusainkat kézzel alakítjuk át gyors, termelési szintű kóddá.

Ebben a cikkben először néhány tippet láthatunk, hogyan írhatunk némileg hatékonyabb Matlab-kódot. Majd bemutatjuk, hogy miképpen lehet a MEX-függvények segítségével C-kódot illeszteni a Matlab programba, hogy felgyorsítsuk a végrehajtást, miközben továbbra is Matlab-környezetben maradunk. Innen már csak egy viszonylag apró lépés a teljes projekt átvitele C vagy C++ alá. Az itt található adatok különböző helyeken a hálózaton is megtalálhatóak; ezt a cikket inkább rövid HOGYAN-nak, még inkább személyes jegyzetnek szántuk – olyannak, ami bemutatja, hogyan hozhatunk át egy kísérleti Matlab-kódot a valódi világba.

Cikkünkben példaként a fej felületén mért feszültség gyors változásainak szétválasztására tervezett kódot fogjuk használni. A kód a többelemű eseményfüggő potenciálbecslés (multicomponent event-related potential estimation, röviden mcERP) nevű algoritmuson alapszik. Akkor kezdtem el először Matlab-programok C-kóddá alakításával foglalkozni, mialatt ezen az algoritmuson dolgoztam. Amikor ugyanis valamilyen beállítási értékkel és bemenő adatokkal kipróbáltam az algoritmust, általában egész éjszakára ott kellett volna hagynom. Semmilyen Matlabba épített optimalizálási módszer sem volt képes jelentősen csökkenteni a végrehajtási időt.

A teljes C-átalakítást követően általában tíz másodperces nagyságrendű időmennyiségre volt szükség egy nagyobb adatmennyiség feldolgozásához. Ez a különösen nagy időmegtakarítás az algoritmus igen mélyen egymásba ágyazódó jelle-

1. lista Beágyazott ciklus a mcERP algoritmusban

```
for i = 1:M
   for j = N:-1:1
      norm = 0;
      for r = 1:R
         for t = 1:T
            X = x(i,r,t);
            for n = 1:N
      if (n ~= j)
          X = X - coupling(i,n) *
                      alpha(n,r) *
                      s (n,pad+t-tau(n,r));
      end
            end
         Y = alpha(j,r)*s(j,pad+t-
             ➥tau(j,r));
         c(i,j) = c(i,j) + X*Y;
         norm = norm + Y^2;
       end
    end
    c(i,j) = c(i,j)/norm;
  end
end
```

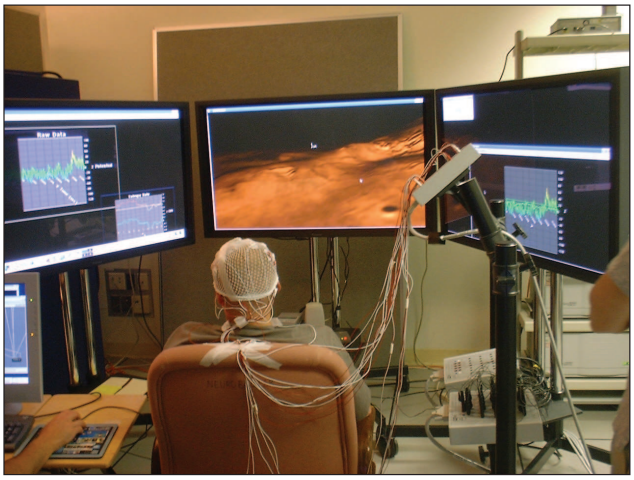
gével magyarázható (lásd az *1. listát*). Nem állíthatjuk, hogy minden algoritmus ennyire felgyorsul. Ráadásul még ez a teljesítmény sem elég jó a valós idejű műveletekhez, de már elég gyors ahhoz, hogy adatcsökkentő módszereket, kódpár-huzamosítási megoldásokat és egyéb trükköket kezdhessünk el keresni, hogy a kívánt sebességhez közeli értéket érjünk el.

Kódoegyszerűsítés Matlabben

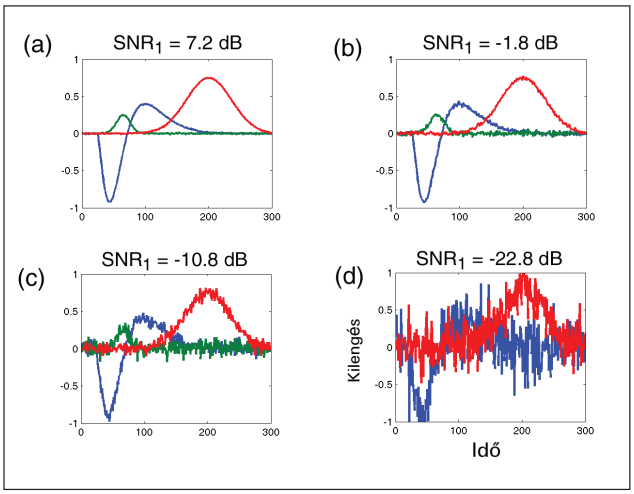
A Matlab teljesítménye a ciklusok futtatáskor a leggyengébb. Akár azt is mondhatnánk, hogy a Matlab kifejezetten utálja a ciklusokat; sokkal hatékonyabban tud végrehajtani néhány ciklusszerű műveletet, ha a kódot előtte vektorizáljuk, és a függvényeket mátrixba rendezett adathalmazon hajtjuk végre, mintha végig kellene lépkednie az adatokon. Sajnos ez csak bizonyos műveletek esetében működik. Amikor nagyobb dimenziószámú mátrixokkal dolgozunk, gyakran kapunk nehezen olvasható és érthető kódot. A ciklusképzés viszont éppen az a terület, amiben a C igencsak jeleskedik – mutatóműveletekkel végiglépkedni egy mátrixon (amennyiben nagy adathalmazokkal dolgozunk) hihetetlenül hatékony és gyakran a legérthetőbb módszer. A Matlab-kódok C-optimalizálásakor a legtöbb energiát a beágyazott ciklusszerkezetek egyszerűsítésébe kell fektetnünk.
Egyéb módszerek, amelyek segítségével megnövelhetjük a Matlab hatékonyságát:

1. Minden tömböt, még a kisebb méretű tömböket is, értékadás előtt a `zeros()` függvény segítségével előre foglaljunk le, és ne hagyjuk, hogy a Matlab az értékadás során egy már létező tömbhöz adatokat rendeljen.
2. Ahogy azt a Matlab leírásában is olvashatjuk, parancsfájlok (scripts) helyett minden kódunkat függvényekben érdemes tárolni. Ezáltal két–háromszoros sebességnövekedést érhetünk el.
3. Szervezzük az adatainkat úgy, hogy a mátrixműveletek oszlopcentrikus módon hajtódjanak végre. A Matlab Fortran-stílusban tárolja a tömböket, azaz a mátrixok oszlopai folytonos területet foglalnak el a memóriában. Ez C alatt pont fordítva működik: itt a memóriában a mátrixok sorai helyezkednek el folytonosan. Amennyiben valamilyen adathalmazon szeretnénk egy függvényt végrehajtani, az adatokat oszlopban, és ne egy mátrix sorában tároljuk. Lehet, hogy ez teljesen értelmetlen és furcsa számunkra, de úgy tűnik, akad valami haszna.
4. Próbáljuk meg elkerülni a belső típusátalakításokat, amelyek időről időre megtörténnek. Ez is egy olyan példa, amire nincsen igazi bizonyítékom, de a Matlab általában nem kényszerít bennünket arra, hogy megadjuk a változók adattípusát, és néha igen könnyű olyan ciklust létrehozni, amelybe implicit típusátalakítás kerül. Sokkal jobb megoldás, ha változóinkat az ismételt végrehajtás előtt általános adattípussá alakítjuk. Tulajdonképpen ez ugyanaz, mint amit C vagy C++ nyelven tennénk, csak éppen nehezebb a helyes utat látni, mivel Matlab alatt a változók közvetlenül szinte soha nem kapnak típust.

Vessünk egy pillantást az mcERP algoritmus kódjának egy szeletkéjére (1. lista). Itt bemutatjuk a kódba ágyazott számos ciklus egyikét. Az mcERP algoritmus Bayes-féle összetett bejárásos (iteration) hullámformabecslésen alapul. Számos, az itt látható részlethez hasonló ciklus található a kódban, amiket többször le kell futtatni, hogy megfelelhessenek az adatban fellelhető hullámformáknak.



1. kép
Ezen a fotón kísérleti összeállításunkat láthatjuk, ahol a valós visszajelzésű agy–számítógép-csatolófelület kifejlesztését célzó kísérleteket végezzük. A három nagy megjelenítőn keresztül tökéletesen irányítani tudjuk, hogy az alany mit láthasson látóterületének legnagyobb részén. Az összes számfeldolgozó és megjelenítő program házon belül készült, és Linux alatt fut



1. ábra
Az ábrán az mcERP algoritmus példakimenetét mutatjuk be, ahol fejtető-elektródákból kiolvasott valós idejű potenciálból levezetett alap-hullámalakok láthatók egy szimulált kísérleti folyamatban. Minden egyes hullámforma számos, egyre pontosabbá váló Bayes-féle hullámforma-közelítési ciklus eredménye, ahol minden egyes ciklus egy ütem alatt rengeteg számítást végez. Ezeket az eredményeket Matlab alatt hosszú órák alatt kapjuk meg, ugyanakkor a számítás csak másodperceket, esetleg perceket vesz igénybe, ha az algoritmus egyes részeit C nyelven írjuk újra

Láthatjuk, hogy az ilyen típusú szerkezetek nem futnak túl gyorsan az értelmezőben, hiszen az a ciklusok futtatásakor nem teljesít valami jól. Ugyanakkor a belső `if` utasítás miatt a kódot belső függvényhívás hozzáadása nélkül nem tudjuk vektorizálni – így ez a módszer nem sokat segít. Ezért aztán C-átalakításunkhoz ez a kód lesz az elsődleges tesztalany. Természetesen ettől még nem baj, ha az algoritmusképzés során a Matlabot is felhasználjuk, hiszen igen könnyen készíthetünk vele az *1. képhez* hasonló látványos képeket. A MEX-

2. lista MEX-függvény

```
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "mex.h"
#include "pops.h"
void latcalc(double *x, double *erp,
             ➥double *alpha,
             double *tau, double *coupling,
             double *range,
             ➥double *inresultlat,
             ➥double *inresulterp,
             int M, int R,
             int T, int NsoFar);

void mexFunction(int nlhs, mxArray * plhs[],
                 int nrhs, const mxArray *
                 ➥prhs[])
{

    int M, R, T, NsoFar;
    const int *dim_array;

    if ((nrhs != 6) || (nlhs != 2))
        mexErrMsgTxt("wrong # of args to
                     ➥c_mLAT");

    dim_array = mxGetDimensions(prhs[0]);

    M = dim_array[0];
    R = dim_array[1];
    T = dim_array[2];

    NsoFar = mxGetM(prhs[1]);

    plhs[0] =
        mxCreateDoubleMatrix(NsoFar, R,
        ➥mxREAL);
    plhs[1] =
        mxCreateDoubleMatrix(NsoFar, T,
        ➥mxREAL);


/*A tényleges munkát végző függvény
meghívása*/
    latcalc(mxGetPr(prhs[0]),
            ➥mxGetPr(prhs[1]),
             mxGetPr(prhs[2]),
             mxGetPr(prhs[3]),
             mxGetPr(prhs[4]),
             mxGetPr(prhs[5]),
             mxGetPr(plhs[0]),
             mxGetPr(plhs[1]),
             M, R, T, NsoFar);


}
```

függvénynek nevezett megoldást fogjuk használni. Ezzel a módszerrel a belső mag gyors futásra lesz képes, ugyanakkor megtartjuk a teljes algoritmus finomítását és vizsgálatát végző Matlab-részekhez kapcsolódó csatolófelületet.

C-környezet összeállítása Matlab alatt

A MathWorks honlapján (lásd a ***Kapcsolódó címeket***) igen jelentős mennyiségű leírást találunk, amit a cikk elolvasása után nem árt, ha átnézegetünk, amennyiben komolyan MEX-függvényekkel szeretnénk foglalkozni. Ez a cikk az egyszerűsítésre összpontosít, illetve néhány lényeges pontot emel ki, amelyek segítenek életet lehelni a dolgokba.
Ha Linux alatt szeretnénk MEX-függvényeket fejleszteni, térjünk vissza minden jó forrásához, a parancssorhoz, és gépeljük be a `mex start` utasítást. Ha ez nem működne, a Matlab telepítési könyvtárában keressük meg a MEX-parancsfájlt. Elképzelhető, hogy elérési utunkban rendszergazdánk csak a futtatható Matlab-állományra készített hivatkozást.
A MEX futtatása során kiválaszthatjuk a fordítónkat. Hogy kihasználhassuk a későbbiekben ismertetett fordítófüggő egyszerűsítési lehetőségeket, jobb, ha a Matlab beépített LCC rendszere helyett inkább a GCC-t használjuk. A MEX ezután létrehozza a ***~/.matlab/R12/mexopts.sh*** állományt, amit akkor használ fel, amikor a MEX-eszközzel külső kódot fordítunk a Matlabhez. Hasznos és tanulságos belenézni a ***mexopts.sh*** állományba felület/fordító párosunk megfelelő szakasza alatt. x86 Linux/GCC használata esetén nézzük át a fő elágazás `glnx86` szakaszát. Minden változtatás, amit ezen a szakaszon kívül végzünk el, semmilyen hatással sem lesz a kódfordítás menetére. Helyezzünk ide azokat a kapcsolókat, amelyekkel C-függvényeinket fordítani szeretnénk. Amennyiben szeretnénk hatékonnyá tenni, a következőket használhatjuk:

```
COPTIMFLAGS='-O3 -funroll-loops -finline-
functions'
```

(ami elég kíméletlen – úgyhogy legyünk óvatosak), illetve bármilyen más tetszés szerinti kapcsolót is használhatunk. Ha a fordításánál ezeket a kapcsolókat szeretnénk használni, a MEX-et a `-O` kapcsolóval kell lefuttatnunk. Akárcsak a `make`-fájlok esetében, szúrjuk be ide azokat a fejléckönyvtárakat, amelyeket a `CFLAGS` végéhez szeretnénk hozzácsapni:

```
-I/a/fejléc/mappa/útvonala
```

A befűzendő programkönyvtárakat így jelöljük:

```
-l[lib-név]
```

és:

```
-L/a/programkönyvtár/mappa/útvonala
```

Ezek az `LDFLAGS` végére kerülnek.
Ha ezzel megvagyunk, készítsünk egy könyvtárat, ahol majd a MEX-szel fordítandó C-fájljainkat tartjuk. Nem javaslom, hogy C alapú és Matlab alapú kódjainkat azonos könyvtárban tároljuk. Ezt a könyvtárat, vagy az általunk készített harmadik eredmény- (build) könyvtárat adjuk a Matlab-környezet elérési útjához. Most már készen állunk a kód létrehozására.

Hogyan írjunk C-kódot Matlab alatt?

Mindenekelőtt gondoljuk át, milyen céljaink vannak az egyszerűsítéssel, illetve hogy programunk mely részei nyerhetnek a legtöbbet azáltal, ha C nyelven újraírjuk őket. Mivel a C elsősorban a ciklusok értelmezése terén lényegesen jobb a Matlabnál, logikusnak tűnik, ha végignézzük a kódunkat, és kikeressük, hol alkalmaztunk jelentős mennyiségű adattal dolgozó ciklusokat. Nemigen éri meg újrakódolni valamit, ami mindössze háromszor hajtódik végre, de ha például egy 500×500×500 egységes térben lépkedünk végig minden egyes voxelen, a C-kód rengeteg megtakarítást jelenthet. Külön figyelmet érdemelnek a beágyazott ciklusokban található egyszerű műveletek, ilyeneket az 1. lista kódrészletében is láthattunk. Ami összetett műveleteket végez egy beágyazott ciklusban – azaz amit láthatóan csak nagyon nehezen tudnánk magunktól elkészíteni, vagy nem találunk hozzá egy harmadik fél által készített programkönyvtárat –, az valószínűleg nem lesz jó kiindulási pont az egyszerűsítéshez. A Matlab-függvényeket C-kódból is meg tudjuk hívni, de ezzel nyilvánvaló módon nem sokat javítottunk a végrehajtási időn.

A C MEX-állományok készítését általában azzal kezdjük, hogy az algoritmusunk valamelyik kódblokkját függvényesítjük, esetleg egy már korábban elkészült függvényt jelölnünk ki egyszerűsítésre. Ha ez megvan, eljött az ideje, hogy elkészítsük a függvény C-változatát. A módszer a következő: hozzuk létre az általános Matlab-csatolófelület-függvényt, majd a tényleges munkát végző lényegi függvényt. A *2. listában* egy ilyen MEX-függvényre láthatunk példát. Az itt meghívott lényegi függvény az 1. listában látható Matlab-fájlhoz tartozik, és a Linux Journal FTP-lapján az ➲ ftp.ssc.com/pub/lj/issue110/6722.tgz címen.

A `mexFunction()` a MEX-fájlprogramozás világában olyasmi, mint C-ben a `main()`. Amikor Matlab alól meghívjuk a függvényünket, ez a függvény fog elindulni. Függvényünk tényleges nevét a lefordított *.c*-állományok neve határozza meg, pontosabban általában a MEX-nek fordításkor elsőként átadott *.c*-állomány neve. Linux x86 felületen a MEX-fájlok kiterjesztése ***.mexglx***. Ha a Matlabet Linux x86-felületen futtatjuk, a Matlab ugyanúgy és ugyanazon az elérési úton keresi a ***.mexglx***-állományokat, mintha a hagyományos *.m*-fájlokat keresné, így az ***.mexglx***- és *.m*-állományok felcserélhetők. Ügyes módszer a Matlab és az egyszerűsített kód közötti váltásra, ha megváltoztatjuk a Matlab keresési útvonalát. Én például a ***c_mLAT.c*** állományt ***c_mLAT.mexglx*** fájllá fordítottam, majd a lefordított kódot a Matlab-környezetből egyszerűen a `c_mLAT()` függvénnyel használhatom. Remek kis rendszer.

A dolgok akkor kezdenek egy kicsit bonyolultabbá válni, amikor adatokat szeretnénk át- és visszaadni a Matlab és a C között. A `mexFunction` tulajdonságai (argumentum) közt két duplamutatót vehetünk észre. A `*plhs[]` a függvény visszatérési értékeire mutat (a Matlab-függvényeknek több visszatérési értékük is lehet), a `*prhs[]` pedig a bemenő értékeket tartalmazza. A bemenő adatok és a visszatérési értékek számát az `nlhs` és az `nrhs` értékekben szintén átadhatjuk. A `mexFuntion()` függvényen belül az `mxCreateDoubleMatrix()` függvénnyel kell lefoglalnunk a visszaadott mátrixok memóriahelyét, amennyiben helyesen akarjuk őket a Matlab-környezetnek visszaadni. Az mx-függvények a Matlab-környezetben készítenek memóriaterületet és a Matlab memóriakezelője kezeli őket, így az mx-függvények által foglalt területek felszabadításával nem kell foglalkoznunk. A mex szócskával kezdődő függvények a Matlab-környezetben futnak le; a `mexCallMATLAB()` függvénnyel például tetszőleges Matlab-függvényt hívhatunk meg a programunkból. A `mexFunction()` függvényünk belsejéből a kimenet lefoglalása után meghívhatjuk a lényegi munkát végző függvényünket, formára szabhatjuk a bemenetet, és például ellenőrizhetjük az értékeket.

A Matlab-leírás sajnos nem foglalkozik egy, a C-programozók számára igen kiábrándító nehézséggel, nevezetesen azzal, hogy a Matlab az adatait oszlopcentrikus formában tárolja. Ez különösen bosszantó lehet, hiszen amikor végiglépkedünk a többdimenziós bemeneti mátrixokon, nyilván a könnyen érthető mutatóaritmetikát szeretnénk használni. Viszont meglehetősen kiábrándító lehet, és hibalehetőségeket rejthet magában, ha nekünk kell kitalálnunk, hogy ciklusonként milyen messzire kell ugranunk és így tovább. Amennyire én látom, három megoldás létezik:

1. Számoljuk ki magunk, hogy ciklusonként mekkora lépéseket kell megtennünk, és tartsuk észben. Ez megoldhatónak tűnik, és talán a legjobb megoldás, de három- vagy több dimenziós mátrixok esetében kétségtelenül komoly fejfájást okoz.
2. Formázzuk újra a kódot, mielőtt belépnénk a MEX-függvénybe, hogy a C nyelvnek megfelelően legyen szervezve. Igaz, ez Matlab alatt igen költséges lehet, és valószínűleg nem árt, ha van egy helyettesítő példányunk az eredetiből.
3. Tegyük azt, amit én is tettem, és készítsünk makrókat vagy makrószerű elemeket, amelyekkel a Matlab-tömböket elérhetjük. Ez lassabb ugyan, mint végiglépkedni a tömbökön, és talán nem is tűnik túl elegáns megoldásnak. Tapasztalataim szerint azonban általában igen gyors és a kódja jól olvasható. Én például egy ***pops.h*** nevű állományt készítettem, ami a következő függvényt tartalmazza:

```
extern inline double
num3d (double *start,
       int rows, int cols,
       int x, int y, int z)
{
     return(*(start
              + rows * cols * z
              + rows * y
              + x));
}
```

KAPCSOLÓDÓ CÍMEK

The MathWorks, a Matlab készítője
➲ http://www.mathworks.com
A Matlab dokumentációs forrása
➲ http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml Válasszuk az External Interfaces/API pontot, majd innen a MEX-leírást.
A Matlabhoz hasonló GNU Octave szabad program honlapja ➲ http://www.octave.org
A NASA Ames-i Számítási Tudományos Részleg, a NASA Neuromérnöki Laboratórium bázisa
➲ http://ic.arc.nasa.gov

A fenti részlet 3D Matlab-tömbformátumban adja vissza az értékeket, amennyiben megadtuk a tömb címét, a sorok és az oszlopok számát, valamint a lekért adat xyz helyzetét. Kicsit talán esetlen, de azért használható. A kódoptimalizálás során ez a kódrészlet a programban ugyanúgy beszúródik, mint az előfeldolgozó makrók. Makrókat is alkalmazhatunk, de én ezt a módszert sokkal könnyebben használhatónak és nyomon követhetőnek találtam. Szerencsére az, hogy a ciklusokat C-ben készítjük el, messze többet nyom a latban, mint tömbelérési módszerünk viszonylag kis vesztesége.

Egyéb tekintetben a MEX-fájlok készítése nem túl bonyolult feladat. Amikor eljön a fordítás ideje, a MEX-programot azokkal a C-fájlokkal futtassuk, amelyeket le szeretnénk fordítani. A MathWorks weblapján megtaláljuk a MEX-fordítási lehetőségek listáját. Miután az ***X.c Y.c Z.c*** fájlokat lefordítottuk, egy ***X.mexglx*** állományt kapunk, amit aztán (amennyiben az elérési utunkban található) a Matlab parancssorából `X()` néven használhatunk.

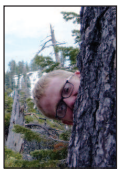
Mostantól kezdve kódunk egyre nagyobb és nagyobb részeit írhatjuk át C-be. Amikor aztán eljön a teljes C-megvalósítás ideje, a külső Matlab-kód átalakítására gyakran megéri a Matlab C-exportálási képességét használni, hiszen a lényeges részeket korábban már amúgy is egyszerűsítettük. Amennyiben a dolgok még mindig nem elég gyorsak, érdemes lehet a külső függvényt újraírni, hogy a memóriát C-barát módon kezelje. Ezáltal felgyorsíthatjuk a belső C-kód ciklusait, hiszen a tömböket egyszerűsített mutatólépegetéssel érhetjük el.

Általában véve jelentősen felgyorsíthatjuk a dolgokat, ha Matlab nyelven fejlesztünk és Matlab-prototípusokat használunk. Előfordulhat azonban, hogy éjszakákat kell várnunk, mire a Matlab kihozza az eredményt, majd kiderül, hogy elgépeltünk egy apró bemeneti értéket. Nos, az ilyesmit elkerülendő a kézzel egyszerűsített kódrészletek igencsak hasznosak lehetnek, ha algoritmusunkat célszerűen szeretnénk használni.

Irodalomjegyzék

Shah, A. S.-Knuth, K. H.-Truccolo, W. A.-Ding, M.-Bressler, S. L.-Schroeder, C. E.: A Bayesian approach to estimating coupling between neural components: evaluation of the multiple component event-related potential (mcERP) algorithm. Bayesian Inference and Maximum Entropy Methods in Science and Engineering: 22nd International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering, C. Williams, American Institute of Physics, Melville, New York, 2003

Linux Journal 2003. június, 110. szám



Sam Clanton

Jelenleg MD/PhD-tanuló a Pittsburgh/Carnegie Mellon Egyetemen, kapcsolatot tart fenn a NASA-val és a kaliforniai Mountain View-i QSS Corporation Ames Kutatóközponttal. Idejét biológiai jelfeldolgozási feladatok megoldásával, számítógépes látási és orvosi robotikai vizsgálatokkal tölti, és nagyon érdekli a természetben létrejött rendszerekhez hasonló informatikai rendszerek összeállítása. Sam idejének javát leveleinek átnézésével tölti, és túl sok kávét iszik.