DOI: 10.31915/NWS.2025.9

A WAVELET-TRANSZFORMÁCIÓ PÁRHUZAMOSÍTÁSA MULTI-GPU KÖRNYEZETBEN A KOMONDOR SZUPERSZÁMÍTÓGÉPEN

Tóth Bálint
Pannon Egyetem, Műszaki Informatikai Kar
Villamosmérnöki és Információs Rendszerek Tanszék
toth.balint@phd.mik.uni-pannon.hu

Juhász Zoltán
Pannon Egyetem, Műszaki Informatikai Kar
Villamosmérnöki és Információs Rendszerek Tanszék
juhasz.zoltan@mik.uni-pannon.hu

Absztrakt

Az agyi EEG-mérések feldolgozásának egyik fontos lépése a Wavelet-transzformációval elvégzett idő-frekvencia elemzés, ami a nagy mennyiségű mérési adat miatt időigényes folyamat. Ennek oka az EEG-ben alkalmazott nagy számú mérőelektróda és a magas mintavételezési frekvencia, ami több száz csatornát és csatornánként több millió adatpontot eredményez. A számítási időt tovább növeli a megfelelő frekvencia felbontás biztosítása miatt szükséges, csatornánként több száz Wavelet-transzformáció végrehajtása. Mindezek miatt szükségessé vált több, az EEG előfeldolgozás során használatos algoritmus párhuzamos GPU megvalósítása szuperszámítógép környezetben. Cikkünkben bemutatjuk a párhuzamos Wavelet-transzformáció algoritmus megtervezésének és CUDA GPU implementációjának főbb lépéseit, valamint a Komondor szuperszámítógépen végzett mérésekre alapozva az elkészült multi-GPU program skálázhatóságát.

Kulcsszavak: Nagyteljesítményű számítás, EEG, GPU, Digitális jelfeldolgozás

Abstract

A Multi-GPU Parallel Implementation for the Wavelet Transform in HPC Environment Using the Komondor Supercomputer

The Wavelet transform-based time-frequency analysis is an important step of EEG data processing. High sampling rate and high-density electrode layouts results in measurements producing hundreds of MBs of data. Coupled with high frequency resolution requiring hundreds of Wavelet transforms to be executed per electrode, the execution time quickly becomes prohibitive. For these reasons, a need for parallel GPU versions has emerged for frequently used EEG algorithms that can be used in HPC environments. In this paper, we

present the main steps of the design and development of the CUDA GPU implementation of the parallel Wavelet transform algorithm and the scalability results of the multi-GPU version executed on the Komondor supercomputer.

Keywords: High Performance Computing, EEG, GPU, Digital Signal Processing

Bevezetés

A szuperszámítógépek napjaink elengedhetetlen tudományos eszközei, amik nélkül szimulációalapú kutatások nem lennének lehetségesek. Számos tudományterület új eredményei köszönhetők nagy teljesítményű számítógépeknek. A High Performance Computing (HPC) több módon is támogathatja a korszerű kutatásokat. Egyrészt, szimulációkkal nagy pontossággal tervezhetők előre kísérletek, amiknek az eredményei a későbbiekben valós eszközökkel igazolhatók, másrészt a mérések és kísérletek során keletkező adatmennyiséget szuperszámítógépek nélkül gyakran beláthatatlanul hosszú idő lenne feldolgozni. A továbbiakban az agykutatásban használt, elektroencefalográfia (EEG) segítségével szerzett mérési adatok feldolgozása során alkalmazott Wavelet-transzformáció párhuzamosítása kerül bemutatásra. Az EEG segítségével – noninvazív módon – nagy időbeli (< 1 ms) felbontású információ nyerhető az agykéregben lezajló bioelektromos folyamatokról. A nagy idő- és térbeli felbontás eléréséhez jellemzően magas mintavételezési frekvenciát (<1–2 kHz, esetenként akár 32 kHz) és minimum 64 vagy 128 mérőelektródát használnak. Ennek következtében egy néhány perces EEG-mérés több száz MB feldolgozandó adatot is generálhat, hosszabb vizsgálatoknál pedig már GB méretű adatfájlokról beszélünk. Ezek feldolgozása az EEG-kutatóközösségben tipikusan MATLAB-alapú keretrendszerekkel történik (példa EEGLAB [1]), ami órás nagyságrendű futási időket eredményez egy-egy komplexebb feldolgozási lépés végrehajtása során. Csoportos vizsgálatok esetén nem ritka a többnapos futási idő.

A párhuzamos GPU-implementációnk a Komondor szuperszámítógép [2] architektúráját figyelembe véve készült, ami 4,51 PFlop/s teljesítményével jelenleg a leggyorsabb számítógép Magyarországon. A számítógép négy partícióból (CPU, GPU, BigData, AI) áll. A GPU-partíció 58 számítási node-ot tartalmaz, mindegyikben egy AMD EPYC Milan processzor és négy darab Ampere architektúrájú NVIDIA A100 grafikus kártya található. A node-on belül a GPU-k NVLink kapcsolattal vannak összekötve, a node-ok közti kapcsolatot pedig a Slingshot interconnect technológia biztosítja.

A Wavelet-transzformáció és a kiindulási GPU-algoritmus

A Wavelet-transzformáció idő-frekvencia elemző módszer, amely átmenet a jelek idő- és frekvenciatartománybeli leírása között. Az idő-frekvencia határozatlansági reláció következtében mindkét tartomány egyidejű pontos ismerete nem lehetséges, ennek ellenére

számos közelítő módszer létezik idő-frekvencia elemzésre. Az egyik ilyen a Wavelet-transzformáció, amelyet kifejezetten a Fourier-transzformáción alapuló módszerek hátrányainak kiküszöbölése érdekében fejlesztettek ki. A fő eltérés a Fourier-transzformációtól az alkalmazott bázisfüggvények és a frekvenciával változó időablak hossz. Míg a Fourier-transzformáció esetén a szinusz- és koszinuszfüggvények végtelenek, a wavelet véges energiájú, rövid időtartamú oszcilláció.

$$W\{f(t)\}(a,b) = \int_{-\infty}^{\infty} f(t)\Psi(\frac{t-b}{a})\,dt \tag{1}$$

$$\Psi(t) = e^{j\omega_0 t}e^{-\frac{t^2}{2}} \tag{2}$$

Matematikailag a wavelet-transzformáció a vizsgált jel és a bázis-wavelet különböző skálázott változatai közötti konvolúciót jelenti. A művelet során az alkalmazott bázis-waveletet a vizsgálni kívánt frekvenciákra skálázzuk, az így előálló jelhalmazt wavelet-családnak nevezzük. A transzformáció folytonos esetben konvolúciós integrállal (1) írható fel, amelyben $\Psi(t)$ egy tetszőleges bázis-wavelet [3]. A skálázás miatt wavelet-ek hossza dinamikusan változik, amely egyfajta sávszűrőként működik, ami a kiválasztott frekvenciát ereszti át. A wavelet olyan függvény, ami teljesíti a véges energia (3) és zérus átlag (4) feltételt. Számos gyakran használt bázis-wavelet létezik, az EEG-feldolgozás során többnyire a Morlet-féle waveletet használjuk, de a GPU-algoritmusunk kialakítása lehetővé teszi egyéb bázis wavelet-ek megvalósítását is. A Morlet-wavelet egy haranggörbével tompított komplex szinuszoid (2).

$$\int_{-\infty}^{\infty} |f(t)|^2 dt \neq \infty \tag{3}$$

$$\int_{-\infty}^{\infty} f(t)\,dt = 0 \tag{4}$$

A vizsgálandó frekvenciakomponensek száma és értéke az algoritmus hiperparamétereiből következik. Az időtartomány mintavételezéséhez hasonlóan, a minimum, maximum frekvencia és a komponensek száma alapján állíthatók elő a frekvenciakomponensek, amelyek alapján végrehajtható a bázis-wavelet skálázása. A szekvenciális kiinduló algoritmus az előbbi integrál diszkretizálásával jön létre (1. algoritmus). A konvolúció számítás-

igényes művelet, azonban hatékonyan párhuzamosítható, mivel az eredmény minden eleme egymástól függetlenül kiszámítható. Az algoritmus által végrehajtott műveletek száma függ a bemeneti jel méretétől (M), a vizsgált frekvenciakomponensek számától (F), a csatornák számától (C) és a bázis-wavelet hosszától (N). Az algoritmus $O(C\,F\,N(M+N-1))$ lépést hajt végre.

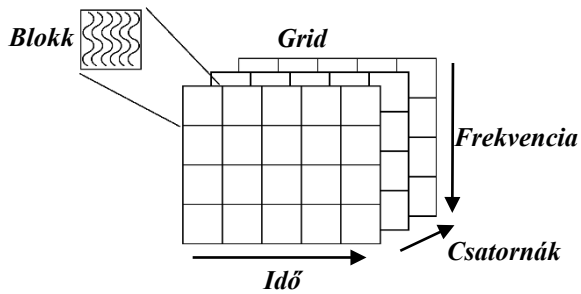
```
input
  w: Wavelet family in row-major layout
  N: Wavelet length
  s: Input signal in row-major layout
  M: Input signal length
  C: Number of channels
  F: Number of frequency bins
output
  wxx: Channel results
begin
  acc ← 0
  for c ← 0 to C do
      for f ← fmin to F do
          for m ← 0 to M + N - 1 do
              acc ← 0 + j0
              for n ← 0 to N - 1 do
                  acc ← acc + w[c, f, n] * s[c, m-n]
              end for
              wxx[c, f, m] ← acc
          end for
      end for
  end for
end
```

1. algoritmus: Wavelet-transzformáció

A szekvenciális algoritmus GPU párhuzamosítása az NVIDIA CUDA programozási modellen keresztül történik [4]. A CUDA-modell szorosan illeszkedik a GPU hardver hierarchiájára, így lehetővé teszi a GPU által nyújtott nagyfokú párhuzamosság maximális kihasználását. A hardver architektúrában az elemi végrehajtási egység a számítási mag, ebből több ezer található egy processzorban. Ez a programozási modellben egy utasítással formájában jelenik meg. Az egy szál által elvégzett műveleteket az ún. kernel függvény adja meg. A számítási magok *Streaming Multiprocessor* egységekbe szerveződnek, ami magába foglalja a magok mellett a gyorsítótárakat és a regiszter fájlt. Ez a szint a programozási modellben a szálakból felépülő *Block* formájában jelenik meg. Egy blokk lehet egy-, kettő- és háromdimenziós. Blokkokból tevődik össze a *Grid*, ami az egész processzort magába foglalja és logikailag szintén lehet többdimenziós.

A Wavelet-transzformáció esetén az eredmény minden eleme függetlenül számítható, amely egy kernel formájában került megvalósításra. A bemeneti adat sorfolytonosan tartalmazza a csatornák (elektródok) adatait, azonban a kimenet a frekvenciakomponensek

miatt már háromdimenziós. A grid és a benne lévő blokkok kétdimenziósak, a kimenet utolsó kettő dimenziója (a frekvenciakomponensek részeredményei) egybe redukálva, sorfolytonos indexeléssel történt (1. ábra). Ennek oka a későbbi multi-GPU kiterjesztés esetén a kommunikációs műveletek csökkentése. A kernel alapja az 1. algoritmus, ebben két ciklus található. A belső végzi el a konvolúciót, a külső a frekvenciakomponenseken iterál. A szekvenciális algoritmus *c* és *m* ciklusát ily módon párhuzamosság váltotta fel (1. ábra). A wavelet családot egy külön kernel a GPU memóriájába generálja a transzformáció előtt. Miután a Morlet-wavelet komplex értékű függvény, a wavelet generáló és a transzformációt elvégző kernelekben is komplex aritmetika került alkalmazásra.



1. ábra: A CUDA-szálhierarchia leképezése a Wavelet-transzformációra

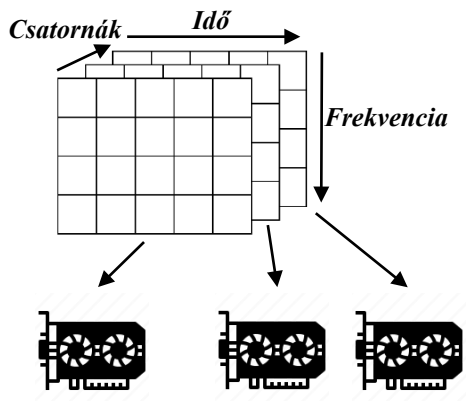
A multi-GPU algoritmus

Az elkészült CUDA-program többféleképpen integrálható multi-GPU rendszerekbe attól függően, hogy az alkalmazási környezet milyen párhuzamosságot tesz lehetővé. EEG-feldolgozás esetén további párhuzamosság adódik a csoportos mérések esetén a kísérleti alanyok mérési eredményeinek szétosztásával. Ilyenkor az algoritmus implementációja semmiben nem változik, a számítógép ütemezője segítségével párhuzamosan futtatunk job-okat, ahol egy alany feldolgozása egy GPU-n történik. A gyorsulás mértéke ilyenkor megegyezik a GPU-k számával, azonban a párhuzamosság korlátozott, további gyorsulás csak több alany hozzáadásával érhető el, így a módszer gyengén skálázható.

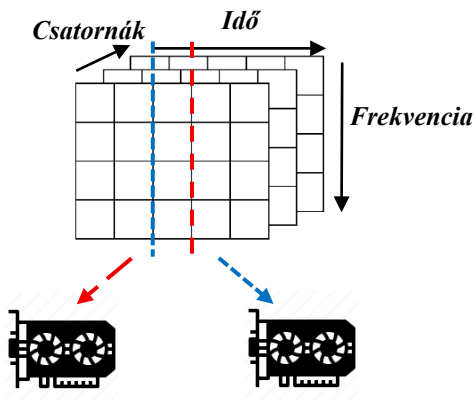
A további gyorsulás elérése érdekében szükséges az egy alanyra jutó adatok további szétosztása a GPU-k között. Ennek megvalósítására a Komondor szuperszámítógépen elérhető a *Message Passing Interface* (MPI) grafikus kártyákat támogató verzióját használtuk, aminek segítségével a node-on belüli és node-ok közti kommunikáció is megvalósítható. A CUDA-aware MPI egy lépésben képes közvetlenül egy GPU memóriájából a másikba adatot továbbítani a node központi memóriájának megkerülésével. Viszont az MPI használatának az a hátránya, hogy a kommunikációt kizárólag CPU-oldali kódból lehet megindítani, így minden adatcsere során az abban résztvevő összes GPU szinkronizációjára van szükség.

A kommunikációs hálózat közbeiktatása jelentősen nagyobb késleltetéssel jár, mint a PCI-e buszon keresztül elvégzett központi- és GPU-memória közötti másolás, ezért a GPU-k és node-ok közti kommunikációt így kizárólag akkor érdemes használni, ha a program futási ideje többszöröse az adatcsere idejének. A javított multi-GPU implementációnk működésének alapja, hogy egy node-on megtörténik az adatok beolvasása, ezután egy kollektív MPI-művelettel a kártyáknak kiosztott adatrészek szétosztása. Minden GPU az előzőekben bemutatott kernelt hajtja végre, miután a kiinduló node egy kollektív művelettel összegyűjti az eredményeket.

A hosszú jeleken elvégzett konvolúció elosztott rendszereken történő kiszámítására ismert megoldás az overlap-add és az overlap-save módszer. Az overlap-add egyenlő részekre osztja a jelet, így a szegmenseken egymástól függetlenül több folyamat elvégezheti a konvolúciót, azonban az eredmény nem teljes. A szegmensek határain lévő értékek előállításához a teljes wavelet-re szükség van, ami azonban a GPU számára nem lenne elérhető, emiatt az eredmények begyűjtésekor szükséges egy újabb összeadás. Mindezek miatt a módszert nem célszerű alkalmazni multi-GPU algoritmus esetén. Overlap-save konvolúció során a jelet átlapolódó ablakokra osztjuk fel. Ebben az esetben a kommunikációs mintázat aszimmetrikus, azonban az eredmények összeillesztéséhez egyetlen begyűjtő kollektív művelet szükséges. Ennek megfelelően a szétosztás során minden GPU megkapja az összes csatorna egy szeletét és azon hajtja végre a többcsatornás konvolúciót. A módszer a szükséges minimumnál jelentősen nagyobb adatmennyiséget cserél ki a GPU-k közt, emellett a csatornák memórián belüli elhelyezkedése miatt az egy GPU-ra jutó adat szegmentált. A módszer előnye elvileg abban áll, hogy a GPU-k számának elméleti korlátja az adatsor hosszával arányosan nő, azonban az extra kommunikáció miatt a gyakorlatban nem alkalmazható hatékonyan.



2. ábra: Csatornaalapú multi-GPU adatszétosztás



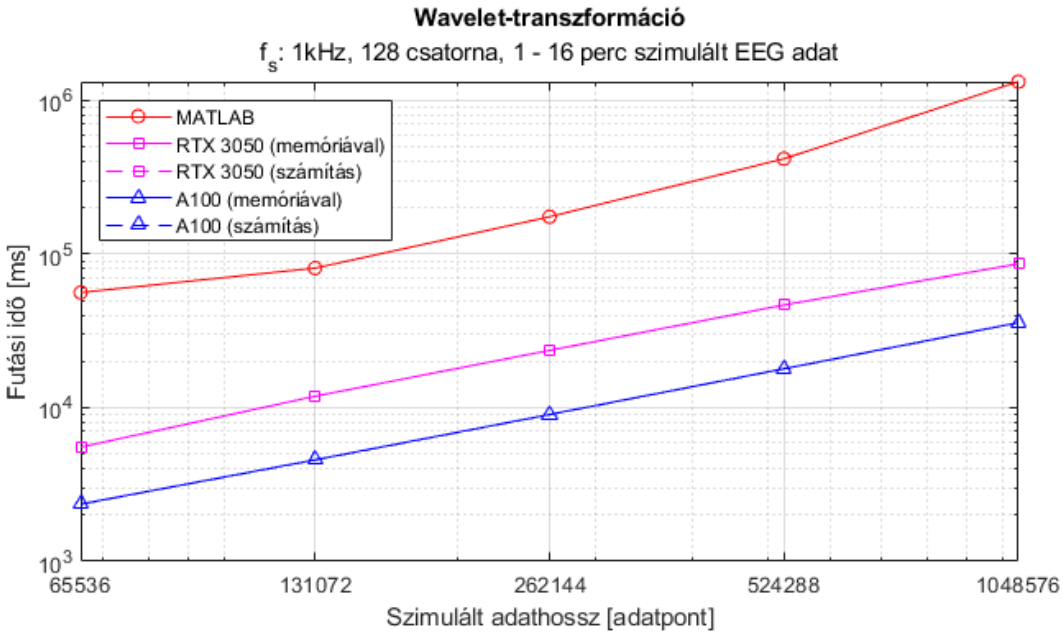
3. ábra: Overlap-save multi-GPU adatszétosztás

Az EEG-mérések struktúrájából következik a csatornák alapján történő GPU-k közötti szétosztás. A módszer lehetővé teszi legfeljebb annyi GPU használatát, amennyi csatornát fel szeretnénk dolgozni. Több száz csatorna esetén ez az elméleti korlát nem jár a várható teljesítmény jelentős visszaeséssel. Az implementációnk a csatornák adatait sorfolytonosan tárolja a memóriában, így a szétosztás- és begyűjtésműveletek az overlap-save és overlap-add módszerekkel ellentétben nem töredezettek. A minta további előnye, hogy amennyiben a csatornák száma megegyezik a rendelkezésre álló GPU-kártyákkal, a műveletvégző kernel egy csatornára optimalizálásával a teljesítmény tovább javítható. Mindezek miatt a multi-GPU algoritmusunk ezt a módszert használja.

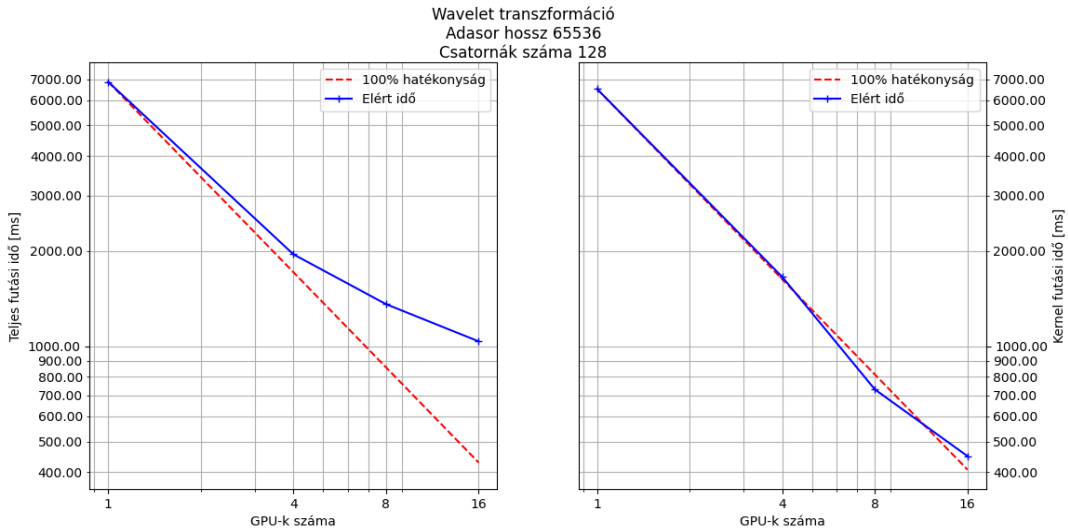
Eredmények

Az elkészült algoritmus GPU-változata két GPU-kártyán lett megvizsgálva. Az első egy NVIDIA RTX3050 mobil GPU, a második a Komondor szuperszámítógépben megtalálható NVIDIA A100 datacenter GPU. A numerikus pontosság ellenőrzése a MATLAB Signal Processing Toolbox implementációja alapján történt, amely egyben szekvenciális futási idő referenciaként is szolgált, mivel az EEG adatfeldolgozási gyakorlatban általános a MATLAB használata. A 4. ábra látható a három hardver környezetben mért futási idő 128 csatorna esetén a bemeneti jelhossz függvényében.

Az RTX3050-kártyán elvégzett részletes teljesítménymérések alapján a transzformációs kernel számításkorlátos, így a konvolúciók kiszámítása jelentősen tovább tart, mint a memória olvasása. A multi-GPU futási időmérések során bebizonyosodott, hogy az algoritmus teljes futási ideje a párhuzamosság növelésével csökken. A teljes futási idő az első szétosztási művelet megkezdésétől az utolsó összegyűjtési művelet végéig az 5. ábra bal oldalán, a kizárólag az egy GPU-kártyára jutó számítás ideje pedig a jobb oldalán látható. A multi-GPU algoritmus futási idejében a számítási idő dominál, így az szinte teljes mértékben elfedi a kommunikációs időt egészen 16 GPU-ig, amikor az egy GPU-ra eső adatmennyiség lecsökken és további jelentős gyorsulás már nem érhető el.



4. ábra: Az algoritmus futási idői a tesztkörnyezetekben, szimulált jelen mérve



5. ábra: A multi-GPU algoritmus gyorsulása a használt kártyák függvényében

Összefoglalás

Cikkünkben az EEG-adatfeldolgozás során használt Wavelet-transzformáció idő-frekvencia elemző módszer GPU és multi-GPU implementációját mutattuk be. Mindkét változat jelentős gyorsulást ért el a szekvenciális MATLAB referencia implementációhoz képest és a multi-GPU kiterjesztésnek köszönhetően tovább skálázható, így hatékonyan alkalmazható

szuperszámítógép környezetben is. Az elkészült implementáció jelentősen felgyorsítja majd az EEG idő-frekvencia elemzések és a konnektivitási hálózatok számításának elvégzését.

Irodalomjegyzék

[1] A. Delorme and S. Makeig, "EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis," *J Neurosci Methods*, vol. 134, no. 1, pp. 9–21, Mar. 2004, doi: 10.1016/J.JNEUMETH.2003.10.009.

[2] "Komondor Documentation." Accessed: Jun. 28, 2025. [Online]. Available: https://docs.hpc.kifu.hu/

[3] L. Chun-Lin, "A tutorial of the wavelet transform," *NTUEE, Taiwan*, vol. 21, no. 22, p. 2, 2010.

[4] "CUDA Programming Guide." Accessed: Jun. 28, 2025. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/contents.html