US00RE49461E

(19) United States
(12) Reissued Patent
Gorchetchnikov et al.

(10) Patent Number: US RE49,461 E
(45) Date of Reissued Patent: Mar. 14, 2023

(54) **GRAPHIC PROCESSOR BASED ACCELERATOR SYSTEM AND METHOD**

(71) Applicant: **Neurala, Inc.**, Boston, MA (US)

(72) Inventors: **Anatoli Gorchetchnikov**, Newton, MA (US); **Heather Marie Ames**, Milton, MA (US); **Massimiliano Versace**, Milton, MA (US); **Fabrizio Santini**, Jamaica Plain, MA (US)

(73) Assignee: **Neurala, Inc.**, Boston, MA (US)

(21) Appl. No.: **17/136,343**

(22) Filed: **Dec. 29, 2020**

Related U.S. Patent Documents

Reissue of:

(64) Patent No.: **9,189,828**
Issued: **Nov. 17, 2015**
Appl. No.: **14/147,015**
Filed: **Jan. 3, 2014**

U.S. Applications:

(63) Continuation of application No. 15/808,201, filed on Nov. 9, 2017, now Pat. No. Re. 48,438, which is an

(Continued)

(51) **Int. Cl.**
G06T 1/60 (2006.01)
G06F 9/50 (2006.01)

(Continued)

(52) **U.S. Cl.**
CPC ***G06T 1/20*** (2013.01); ***G06F 9/5027*** (2013.01); ***G06T 1/60*** (2013.01);

(Continued)

(58) **Field of Classification Search**
CPC ... G06F 9/5027; G06F 2209/509; G06T 1/20; G06T 1/60; G06N 3/00; G06N 3/02;

(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,063,603 A 11/1991 Burt
5,136,687 A 8/1992 Edelman et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 1224622 B1 11/2004
WO 190208 11/2014

(Continued)

OTHER PUBLICATIONS

Hodgkin, A. L., and Huxley, A. F. 1952. Quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117, pp. 500-544.

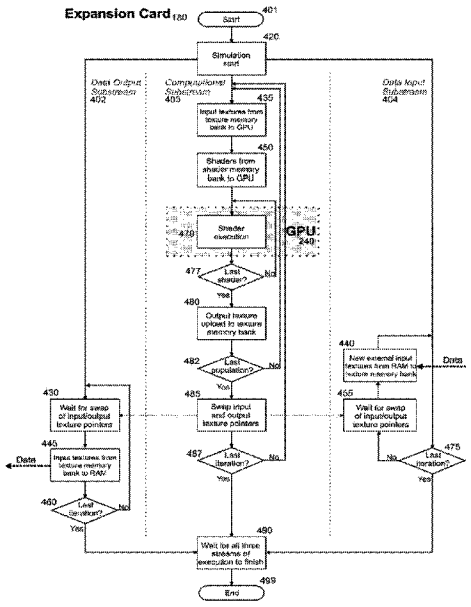
(Continued)

Primary Examiner — William H. Wood
(74) *Attorney, Agent, or Firm* — Smith Baluch LLP

(57) **ABSTRACT**

An accelerator system is implemented on an expansion card comprising a printed circuit board having (a) one or more graphics processing units (GPUs), (b) two or more associated memory banks (logically or physically partitioned), (c) a specialized controller, and (d) a local bus providing signal coupling compatible with the PCI industry standards. The controller handles most of the primitive operations to set up and control GPU computation. Thus, the computer's central processing unit (CPU) can be dedicated to other tasks. In this case a few controls (simulation start and stop signals from the CPU and the simulation completion signal back to CPU), GPU programs and input/output data are exchanged between CPU and the expansion card. Moreover, since on every time step of the simulation the results from the previous time step are used but not changed, the results are preferably transferred back to CPU in parallel with the computation.

21 Claims, 5 Drawing Sheets



Related U.S. Application Data

application for the reissue of Pat. No. 9,189,828, which is a continuation of application No. 11/860,254, filed on Sep. 24, 2007, now Pat. No. 8,648,867.

(60) Provisional application No. 60/826,892, filed on Sep. 25, 2006.

(51) **Int. Cl.**
G06T 1/20 (2006.01)
G06N 3/063 (2023.01)
G06N 20/00 (2019.01)

(52) **U.S. Cl.**
CPC *G06F 2209/509* (2013.01); *G06N 3/063* (2013.01); *G06N 20/00* (2019.01)

(58) **Field of Classification Search**
CPC G06N 3/04; G06N 3/06; G06N 3/063; G06N 3/08; G06N 3/10; G06N 5/00; G06N 7/00; G06N 7/02; G06N 7/04; G06N 7/046; G06N 7/06; G06N 20/00
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,142,665 A 8/1992 Bigus
5,172,253 A 12/1992 Lynne
5,388,206 A 2/1995 Poulton et al.
6,018,696 A 1/2000 Matsuoka et al.
6,336,051 B1 1/2002 Pangels et al.
6,647,508 B2 11/2003 Zalewski et al.
7,119,810 B2 10/2006 Sumanaweera et al.
7,219,085 B2 * 5/2007 Buck G06V 10/955 706/12
7,477,256 B1 * 1/2009 Johnson G06F 3/14 345/506
7,525,547 B1 * 4/2009 Diard G06T 1/20 345/522
7,765,029 B2 7/2010 Fleischer et al.
7,861,060 B1 * 12/2010 Nickolls et al. 712/22
7,873,650 B1 1/2011 Chapman et al.
8,392,346 B2 3/2013 Ueda et al.
8,510,244 B2 8/2013 Carson et al.
8,583,286 B2 11/2013 Fleischer et al.
8,648,867 B2 * 2/2014 Gorchetchnikov et al. .. 345/501
9,031,692 B2 5/2015 Zhu
9,177,246 B2 11/2015 Buibas et al.
9,189,828 B2 11/2015 Gorchetchnikov et al.
9,626,566 B2 4/2017 Versace et al.
10,083,523 B2 9/2018 Versace et al.
RE48,438 E * 2/2021 Gorchetchnikov ... G06F 9/5027
2001/0010034 A1 7/2001 Burton
2002/0046271 A1 4/2002 Huang
2002/0050518 A1 5/2002 Roustaei
2002/0064314 A1 5/2002 Comaniciu et al.
2002/0168100 A1 11/2002 Woodall
2003/0026588 A1 2/2003 Elder et al.
2003/0078754 A1 4/2003 Hamza
2004/0015334 A1 1/2004 Ditlow et al.
2005/0166042 A1 7/2005 Evans
2006/0129506 A1 * 6/2006 Edelman G05D 1/0088 706/12
2006/0184273 A1 8/2006 Sawada et al.
2007/0052713 A1 3/2007 Chung et al.
2007/0198222 A1 8/2007 Schuster et al.
2007/0279429 A1 12/2007 Ganzer
2008/0033897 A1 2/2008 Lloyd
2008/0066065 A1 3/2008 Kim et al.
2008/0258880 A1 10/2008 Smith et al.
2009/0080695 A1 3/2009 Yang
2009/0089030 A1 4/2009 Sturrock et al.
2009/0116688 A1 5/2009 Monacos et al.
2010/0048242 A1 2/2010 Rhoads et al.
2011/0004341 A1 1/2011 Sarvadevabhatla et al.
2011/0173015 A1 7/2011 Chapman et al.
2011/0279682 A1 11/2011 Li et al.
2012/0072215 A1 3/2012 Yu et al.
2012/0089552 A1 4/2012 Chang et al.
2012/0197596 A1 8/2012 Comi
2012/0316786 A1 12/2012 Liu et al.
2013/0126703 A1 5/2013 Caulfield
2013/0131985 A1 5/2013 Weiland et al.
2014/0019392 A1 1/2014 Buibas et al.
2014/0032461 A1 1/2014 Weng
2014/0052679 A1 2/2014 Sinyavskiy et al.
2014/0089232 A1 3/2014 Buibas et al.
2015/0127149 A1 5/2015 Sinyavskiy et al.
2015/0134232 A1 5/2015 Robinson
2015/0224648 A1 8/2015 Lee et al.
2016/0075017 A1 3/2016 Laurent et al.
2016/0082597 A1 3/2016 Gorshechnikov et al.
2016/0096270 A1 4/2016 Gabardos et al.
2016/0198000 A1 7/2016 Gorshechnikov et al.
2017/0024877 A1 1/2017 Versace et al.
2017/0076194 A1 3/2017 Versace et al.
2017/0193298 A1 7/2017 Versace et al.

FOREIGN PATENT DOCUMENTS

WO 2014204615 A2 12/2014
WO 2015143173 A2 9/2015
WO 2016014137 A2 1/2016

OTHER PUBLICATIONS

Hopfield, J. 1982. Neural networks and physical systems with emergent collective computational abilities. In Proc Natl Acad Sci USA, vol. 79, pp. 2554-2558.
Ilie, A. 2002. Optical character recognition on graphics hardware. Tech. Rep. integrative paper, UNCCH, Department of Computer Science, 9 pages.
International Preliminary Report on Patentability in related PCT Application No. PCT/US2014/039162 filed May 22, 2014, dated Nov. 24, 2015, 7 pages.
International Preliminary Report on Patentability in related PCT Application No. PCT/US2014/039239 filed May 22, 2014, dated Nov. 24, 2015, 8 pages.
International Preliminary Report on Patentability dated Nov. 8, 2016 from International Application No. PCT/US2015/029438, 7 pages.
International Search Report and Written Opinion dated Feb. 18, 2015 from International Application No. PCT/US2014/039162, 12 pages.
International Search Report and Written Opinion dated Feb. 23, 2016 from International Application No. PCT/US2015/029438, 11 pages.
International Search Report and Written Opinion dated Jul. 6, 2017 from International Application No. PCT/US2017/029866, 12 pages.
International Search Report and Written Opinion dated Nov. 26, 2014 from International Application No. PCT/US2014/039239, 14 pages.
International Search Report and Written Opinion dated Sep. 15, 2015 from International Application No. PCT/US2015/021492, 9 pages.
Itti, L., and Koch, C. (2001). Computational modelling of visual attention. Nature Reviews Neuroscience, 2 (3), 194-203.
Itti, L., Koch, C., and Niebur, E. (1998). A Model of Saliency-Based Visual Attention for Rapid Scene Analysis, 1-6.
Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., & LeCun, Y. (Sep. 2009). What is the best multi-stage architecture tor object recognition?. In Computer Vision, 2009 IEEE 12th International Conference on (pp. 2146-2153) IEEE.
Khaligh-Razavi, S.-M et al., Deep Supervised, but Not Unsupervised, Models May Explain IT Cortical Representation, PLoS Computational Biology, vol. 10, Issue 11, 29 pages (Nov. 2014).

(56) **References Cited**

OTHER PUBLICATIONS

Kim, S., Novel approaches to clustering, biclustering and algorithms based on adaptive resonance theory and Intelligent control, Doctoral Dissertations, Missouri University of Science and Technology, 125 pages (2016).
Kipfer, P., Segal, M., and Westermann, R. 2004. UberFlow: A GPU-Based Particle Engine. In Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware 2004, pp. 115-122.
Kolb, A., L. Latta, and C. RF7K-SALAMA. 2004. "Hardware-Based Simulation and Collision Detection for Large Particle Systems." In Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware 2004, pp. 123-131.
Kompella, Varun Raj, Luciw, Matthew, and Schmidhuber, Jurgen. Incremental slow feature analysis: Adaptive low-complexity slow feature updating from high-dimensional input streams Neural Computation, 24(11):2994-3024, 2012.
Kowler, E. (2011). Eye movements: The past 25years. Vision Research, 51(13), 1457-1483. doi:10.1016/j.visres.2010.12.014.
Larochelle H., & Hinton G. (2012). Learning to combine foveal glimpses with a third-order Boltzmann machine. NIPS 2010,1243-1251.
LeCun, Y., Kavukcuoglu, K., & Farabet, C. (May 2010). Convolutional networks and applications in vision. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on (pp. 253-256). IEEE.
Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401(6755):788-791.
Lee, D. D., and Seung, H. S. (1997). "Unsupervised learning by convex and conic coding." Advances in Neural Information Processing Systems, 9.
Legenstein, R., Wilbert, N., and Wiskott, L. Reinforcement learning on slow features of high-dimensional input streams. PLoS Computational Biology, 6(8), 2010. ISSN 1553-734X. 13 pages.
Léveillé, J., Ames, H., Chandler, B., Gorchetchnikov, A., Mingolla, E., Patrick, S., and Versace, M. (2010) Learning in a distributed software architecture for large-scale neural modeling. BIONETICS10, Boston, MA, USA. 8 pages.
Livitz G., Versace M., Gorchetchnikov A., Vasilkoski Z., Ames H., Chandler B., Leveille J. and Mingolla E. (2011) Adaptive, brain-like systems give robots complex behaviors, The Neuromorphic Engineer,: 10.2417/1201101.003500 Feb. 2011. 3 pages.
Livitz, G., Versace, M., Gorchetchnikov, A., Vasilkoski, Z., Ames, H., Chandler, B., Leveille, J., Mingolla, E., Snider, G., Amerson, R., Carter, D., Abdalla, H., and Qureshi, S. (2011) Visually-Guided Adaptive Robot (ViGuAR). Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2011, San Jose, CA, USA. 9 pages.
Lowe, D.G.(2004). Distinctive Image Features from Scale-Invariant Keypoints. Journal International Journal of Computer Vision archive vol. 60, 2, 91-110.
Lu, Z.L., Liu, J., and Dosher, B.A. (2010) Modeling mechanisms of perceptual learning with augmented Hebbian re-areighting Vision Research, 50(4). 375-390.
Luo et al., "Ailificial neural network computation on graphic process unit." Neural Networks, 2005. IJCNN'05. Proceedings 2005 IEEE International Joint Conference on vol. 1 IEEE, 2005 pp. 622-626.
Mahadevan, S. Proto-value functions: Developmental reinforcement learning. In Proceedings of the 22nd international conference on Machine learning, pp. 553-560. ACM, 2005.
Meuth, J.R. and Wunsch, D.C. (2007) A Survey of Neural Computation On Graphics Processing Hardware. 22nd IEEE International Symposium on Intelligent Control, Part of IEEE Multi-conference on Systems and Control, Singapore, Oct. 1-3, 2007, 5 pages.
Mishkin M, Ungerleider LG. (1982). "Contribution of striate inputs to the visuospatial functions of parieto-preoccipital cortex in monkeys," Behav Brain Res, 6 (1): 57-77.
Minih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529-533, Feb. 25, 2015.
Montrym et al., The GeForce 6800, in IEEE Micro, vol. 25, No. 2, pp. 41-51, March-Apr. 2005.
Moore, Andrew W and Atkeson, Christopher G. Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning, 13(1):103-130,1993.
Najemnik, J., and Geisler, W. (2009). Simple summation rule for optimal fixation selection in visual search. Vision Research. 49, 1286-1294.
Non-Final Office Action dated Jan. 4, 2018 from U.S. Appl. No. 15/262,637, 23 pages.
Non-Final Office Action dated May 31, 2018 from U.S. Appl. No. 14/947,516, 16 pages.
Notice of Alllowance dated May 22, 2018 from U.S. Appl. No. 15/262,637, 6 pages.
Notice of Allowance dated Jul. 27, 2016 from U.S. Appl. No. 14/662,657.
Notice of Allowance dated Dec. 16, 2016 from U.S. Appl. No. 14/662,657.
Oh, K.-S., and Jung, K. 2004. GPU implementation of neural networks. Pattern Recognition 37, pp. 1311-1314.
Oja, E. (1982). Simplified neuron model as a principal component analyzer. Journal of Mathematical Biology 15(3), 267-273.
Partial Supplementary European Search Report dated Jul. 4, 2017 from European Application No. 14800348.6, 13 pages.
Perumalla, "Discrete-event execution alternatives on general purpose graphical processing units (GPGPUs)." Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation. IEEE Computer Society, 2006.8 pages.
Raijmakers, M.E.J., and Molenaar, P. (1997). Exact Art: A complete implementation of an ART network Neural networks 10 (4), 649-669.
Ranzato, M. A., Huang, F. J., Boureau, Y. L., & Lecun, Y. (2007, June). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on (pp. 1-8). IEEE.
Raudies, F., Eldridge, S., Joshi, A., and Versace, M. (Aug. 20, 2014). Learning to navigate in a virtual world using optic flow and stereo disparity signals. Artificial Life and Robotics, DOI 10.1007/10015-014-0153-1. 15 pages.
Adelson, E. H , Anderson, C. H , Bergen, J R., Burt, P. J , & Ogden, J. M (1984) Pyramid methods in image processing. RCA engineer, 29(6), 33-41.
Aggarwal, Charu C, Hinneburg, Alexander, and Keim, Daniel A. On the surprising behavior of distance metrics in high dimensional space. Springer, 2001. 15 pages.
Al-Kaysi, A. M. et al., A Multichannel Deep Belief Network for the Classification of EEG Data, from Ontology-based Information Extraction for Residential Land Use Suitability: A Case Study of the City of Regina, Canada, DOI 10.1007/978-3-319-26561-2_5, 8 pages (Nov. 2015).
Ames, H, Versace, M., Gorchetchnikov, A., Chandler, B., Livitz, G., Léveillé, J., Mingolla, E., Carter, D., Abdalla, H., and Snider, G. (2012) Persuading computers to act more like brains. In Advances in Neuromorphic Memristor Science and Applications, Kozma, R.Pino,R., and Pazienza, G. (eds), Springer Verlag. 25 pages.
Ames, H. Mingolla, E., Sohail, A., Chandler, B., Gorchetchnikov, A., Leveille, J., Livitz, G. and Versace, M. (2012) The Animat. IEEE Pulse, Feb. 2012, 3(1), 47-50.
Apolloni, B. et al., Training a network of mobile neurons, Proceedings of International Joint Conference on Neural Networks, San Jose, CA, doi: 10.1109/IJCNN.2011.6033427, pp. 1683-1691 (Jul. 31-Aug. 5, 2011).
Artificial Intelligence as a Service. Invited talk, Defrag, Broomfield, CO, Nov. 4-6, 2013. 22 pages.
Aryananda, L. 2006. Attending to learn and learning to attend for a social robot. Humanoids 06, pp. 618-623.

(56) **References Cited**

OTHER PUBLICATIONS

Baraldi, A. and Alpaydin, E. (1998). Simplified Art: A new class of Art algorithms. International Computer Science Institute, Berkeley, CA, TR-98-004, 1998. 42 pages.
Baraldi, A. and Alpaydin, E. (2002). Constructive feedforward Art clustering networks—Part I. IEEE Transactions on Neural Networks 13(3), 645-661.
Baraldi, A. and Parmiggiani, F. (1997). Fuzzy combination of Kohonen's and ART neural network models to detect statistical regularities in a random sequence of multi-valued input patterns. In International Conference on Neural Networks, IEEE. 6 pages.
Baraldi, Andrea and Alpaydin, Ethem. Constructive feedforward ART clustering networks—part II IEEE Transactions on Neural Networks, 13(3):662-677, May 2002. ISSN 1045-9227. doi: 10.1109/tnn.2002.1000131. URL http://dx.doi.org/10.1109/tnn.2002.1000131.
Bengio, Y., Courville, A., & Vincent, P. Representation learning: A review and new perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35 Issue 8, Aug. 2013. pp. 1798-1828.
Berenson, D et al., A robot path planning framework that learns from experience, 2012 International Conference on Robotics and Automation, 2012, 9 pages [retrieved from the internet] URL:http://users.wpi.edu/-dberenson/lightning.pdf.
Bernhard, F., and Keriven, R. 2005. Spiking Neurons on GPUs. Tech. Rep. 05-15, Ecole Nationale des Ponts et Chauss'es, 8 pages.
Besl, P. J., & Jain, R. C. (1985). Three-dimensional object recognition. ACM Computing Surveys (CSUR), 17(1), 75-145.
Boddapati, V., Classifying Environmental Sounds with Image Networks, Thesis, Faculty of Computing Blekinge Institute of Technology, 37 pages (Feb. 2017).
Bohn, C.-A. Kohonen. 1998. Feature Mapping Through Graphics Hardware. In Proceedings of 3rd Int. Conference on Computational Intelligence and Neurosciences, 4 pages.
Bradski, G., & Grossberg, S. (1995). Fast-learning Viewnet architectures for recognizing three-dimensional objects from multiple two-dimensional views. Neural Networks, 8 (7-8), 1053-1080.
Canny, J.A. (1986). Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679-698.
Carpenter, G.A. and Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. Computer Vision, Graphics, and Image Processing 37, 54-115.
Carpenter, G.A., and Grossberg, S. (1995). Adaptive resonance theory (ART). In M. Arbib (Ed.), The handbook of brain theory and neural networks, (pp. 79-82). Cambridge, M.A.: MIT press.
Carpenter, G.A., Grossberg, S. and Rosen, D.B. (1991). Fuzzy Art: Fast stable learning and categorization of analog patterns by an adaptive resonance system Neural Networks 4, 759-771.
Carpenter, Gail A and Grossberg, Stephen. The art of adaptive pattern recognition by a self-organizing neural network. Computer, 21(3):77-88, 1988.
Coifman, R.R. and Maggioni, M. Diffusion wavelets. Applied and Computational Harmonic Analysis, 21(1):53-94, 2006.
Coifman, R.R., Lafon, S., Lee, A.B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S.W. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. Proceedings of the National Academy of Sciences of the United States of America, 102(21):7426, 2005. 21 pages.
Cornwall et al., Automatically translating a general purpose C++ image processing library for GPUs. Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, 2006, 8 pages.
Davis, C. E. 2005. Graphic Processing Unit Computation of Neural Networks. Master's thesis, University of New Mexico, Albuquerque, NM, 121 pages.
Dosher, B.A., and Lu, Z.L. (2010). Mechanisms of perceptual attention in precuing of location. Vision Res., 40(10-12). 1269-1292.
Ellias, S. A., and Grossberg, S. 1975. Pattern formation, contrast control and oscillations in the short term memory of shunting on-center off-surround networks Biol Cybern 20, pp. 69-98.
Extended European Search Report and Written Opinion dated Jun. 1, 2017 from European Application No. 14813864.7, 10 pages.
Extended European Search Report and Written Opinion dated Oct. 12, 2017 from European Application No. 14800348.6, 12 pages.
Extended European Search Report and Written Opinion Oct. 23, 2017 from European Application No. 15765396.5, 3 pages.
Fazl, A., Grossberg, S., and Mingolla, E. (2009). View-invariant object category learning, recognition, and search: How spatial and object attention are coordinated using surface-based attentional shrouds. Cognitive Psychology 58, 1-48.
Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning, Biological Cybernetics, vol. 64, pp. 165-170.
Friston K., Adams R., Perrinet L., & Breakspear M. (2012). Perceptions as hypotheses: saccades as experiments. Frontiers in Psychology, 3 (151), 1-20.
Galbraith, B.V, Guenther, F.H., and Versace, M. (2015) A neural network-based exploratory learning and motor planning system for co-robots Frontiers in Neuroscience, in press. 10 pages.
George, D. and Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. PLoS Computational Biology 5(10), 1-26.
Georgii, J., and Westermann, R. 2005. Mass-spring systems on the GPU. Simulation Modelling Practice and Theory 13, pp. 693-702.
Gorchetchnikov A., Hasselmo M. E. (2005). A biophysical implementation of a bidirectional graph search algorithm to solve multiple goal navigation tasks. Connection Science, 17(1-2), pp. 145-166.
Gorchetchnikov A., Hasselmo M. E. (2005). A simple rule for spike-timing-dependent plasticity: local influence of AHP current. Neurocomputing, 65-66, pp. 885-890.
Gorchetchnikov A., Versace M., Hasselmo M. E. (2005). A Model of STDP Based on Spatially and Temporally Local Information: Derivation and Combination with Gated Decay. Neural Networks, 18, pp. 458-466.
Gorchetchnikov A., Versace M., Hasselmo M. E. (2005). Spatially and temporally local spike-timing-dependent plasticity rule. In: Proceedings of the International Joint Conference on Neural Networks, No. 1568 in IEEE CD-ROM Catalog No. 05CH37662C, pp. 390-396.
Gorcheichnikov, A. 2017. An Approach to a Biologically Realistic Simulation of Natural Memory. Master's thesis, Middle Tennessee State University, Murfreesboro, TN, 70 pages.
Grossberg, S. (1973). Contour enhancement, short-term memory, and constancies in reverberating neural networks. Studies in Applied Mathematics 52, 213-257.
Grossberg, S., and Huang, T.R. (2009). Artscene: A neural system for natural scene classification. Journal of Mision, 9 (4), 6.1-19. doi:10.1167/9.4.6.
Grossberg, S., and Versace, M. (2008) Spikes, synchrony, and attentive learning by laminar thalamocortical circuits. Brain Research, 1218C, 278-312 [Authors listed alphabetically].
Hagen, T. R., Hjelmervik, J., Lie, K.-A., Natvig, J., and Ofstad Henriksen, M. 2005. Visual simulation of shallow-water waves. Simulation Modelling Practice and Theory 13, pp. 716-726.
Hasselt, Hado Van. Double q-learning. In Advances in Neural Information Processing Systems, pp. 2613-2621,2010.
Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18, 1527-1554.
Ren, Y et al., Ensemble Classification and Regression—Recent Developments, Applications and Future Directions, in IEEE Computational Intelligence Magazine, 10.1109/MCI.2015.2471235, 14 pages (2016).
Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. Nature Neuroscience, 2 (11), 1019-1025.
Riesenhuber, M., & Poggio, T. (2000). Models of object recognition. Nature neuroscience, 3, 1199-1204.
Rolfes, T. 2004. Artificial Neural Networks on Programmable Graphics Hardware. In Game Programming Gems 4, A. Kirmse, Ed. Charles River Media, Hingham, MA, pp. 373-378.

(56) **References Cited**

OTHER PUBLICATIONS

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In IEEE International Conference on Computer Vision (ICCV) 2011, 2564-2571.
Ruesch, J et al. 2008. Multimodal Saliency-Based Bottom-Up Attention a Framework for the Humanoid Robot iCub. 2008 IEEE International Conference on Robotics and Automation, pp. 962-965.
Rumelhart D., Hinton G., and Williams, R. (1986). Learning internal representations by error propagation. In Parallel distributed processing: explorations in the microstructure of cognition, vol. 1, MIT Press. 45 pages.
Rumpf, M. and Strzodka, R. Graphics processor units: New prospects for parallel computing. In Are Magnus Bruaset and Aslak Tveito, editors, Numerical Solution of Partial Differential Equations on Parallel Computers, vol. 51 of Lecture Notes in Computational Science and Engineering, pp. 89-134. Springer, 2005.
Salakhutdinov, R., & Hinton, G. E. (2009). Deep boltzmann machines. In International Conference on Artificial Intelligence and Statistics (pp. 448-455).
Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. arXiv preprint arXiv: 1511.05952, Nov. 18, 2015. 21 pages.
Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990-2010). Autonomous Mental Development, IEEE Transactions on, 2(3), 230-247.
Schmidhuber, Jürgen. Curious model-building control systems. In Neural Networks, 1991. 1991 IEEE International Joint Conference on, pp. 1458-1463. IEEE, 1991.
Seibert, M., & Waxman, A.M. (1992). Adaptive 3-D Object Recognition from Multiple Views. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (2), 107-124.
Setoain et al., "Parallel hyperspectral image processing on commodity graphics hardware." Parallel Processing Workshops, 2006. ICPP 2006 Workshops 2006 International Conference on. IEEE, 2006. 8 pages.
Sherbakov, L. and Versace, M. (2014) Computational principles for an autonomous active vision system. Ph.D., Boston University, http://search.proquest.com/docview/1558856407. 194 pages.
Sherbakov, L. et al. 2012. CogEye: from active vision to context identification, youtube, retrieved from the Internet an Oct. 10, 2017: URL://www.youtube.com/watch?v=i5PQk962B1k, 1 page.
Sherbakov, L. et al. 2013. CogEye: system diagram module brain area function algorithm approx # neurons, retrieved from the Internet on Oct. 12, 2017: URL://http://www-labsticc.univ-ubs.fr/~coussy/neucomp2013/index_fichiers/material/posters/NeuComp2013_final56x36.pdf, 1 page.
Sherbakov, L., Livitz, G., Sohail, A., Gorchetchnikov, A., Mingolla, E., Ames, H., and Versace, M (2013b) A computational model of the role of eye-movements in object disambiguation. Cosyne, Feb. 28-Mar. 3, 2013. Salt Lake City, UT, USA. 2 pages.
Sherbakov, L., Livitz, G., Sohail, A., Gorchetchnikov, A., Mingolla, E., Ames, H., and Versace, M. (2013a) CogEye: An online active vision system that disambiguates and recognizes objects NeuComp 2013.2 pages.
Smolensky, Paul. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. Colorado Univ At Boulder Dept of Computer Science, 1986. 88 pages.
Snider, Greg, et al. "From synapses to circuitry: Using memristive memory to explore the electronic brain." IEEE Computer, vol. 44(2). (2011): 21-28.
Spratling, M. W. (2008). Predictive coding as a model of biased competition in visual attention. Vision Research, 48(12):1391-1408.
Spratling, M. W. (2012). Unsupervised learning of generative and discriminative weights encoding elementary image components in a predictive coding model of cortical function. Neural Computation, 24(1):60-103.
Spratling, M. W., De Meyer, K., and Kompass, R. (2009). Unsupervised learning of overlapping image components using divisive input modulation. Computational intelligence and neuroscience. 20 pages.
Sprekeler, H. On the relation of slow feature analysis and laplacian eigenmaps. Neural Computation, pp. 1-16, 2011.
Sun, Z et al., Recognition of SAR target based on multilayer auto-encoder and SNN, International Journal of Innovative Computing, Information and Control, vol. 9, No. 11, pp. 4331-4341, Nov. 2013.
Sutton, R. S., and Barto, A. G. (1998). Reinforcement learning: An introduction(vol. 1, No. 1). Cambridge: MIT press. 10 pages.
Tong, F., Ze-Nian Li, (1995). Reciprocal-wedge transform for space-variant sensing, Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol. 17, No. 5, pp. 500-551 doi: 10 1109/34.391393.
Torralba, A., Oliva, A., Castelhano, M.S., Henderson, J.M. (2006). Contextual guidance of eye movements and attention in real-world scenes: the role of global features in object search Psychological Review, 113(4).766-786.
Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. arXiv preprint arXiv: 1509.06461, Sep. 22, 2015. 7 pages.
Versace, Brain-inspired computing. Invited keynote address, Bionetics 2010, Boston, MA, USA. 1 page.
Versace, M. (2006) From spikes to interareal synchrony: how attentive matching and resonance control learning and Information processing by laminar thalamocortical circuits. NSF Science of Learning Centers PI Meeting, Washington, DC, USA. 1 page.
Versace, M., (2010) Open-source software for computational neuroscience: Bridging the gap between models and behavior. In Horizons in Computer Science Research, vol. 3 43 pages.
Versace, M., Ames, H., Léveillé, J., Fortenberry, B., and Gorchetchnikov, A. (2008) KlnNeSS: A modular framework for computational neuroscience Neuroinformatics, 2008 Winter; 6(4):291-309. Epub Aug. 10, 2008.
Versace, M., and Chandler, B. (2010) MoNeta: A Mind Made from Memristors. IEEE Spectrum, Dec. 2010. 8 pages.
Versace, TEDx Fulbright, Invited talk, Washington DC, Apr. 5, 2014. 30 pages.
Webster, Bachevalier, Ungerleider (1994). Connections of IT areas TEO and TE with parietal and frontal cortex in macaque monkeys. Cerebal Cortex, 4(5), 470-483.
Wiskott, Laurenz and Sejnowski, Terrence. Slow feature analysis: Unsupervised learning of invariances. Neural Computation, 14(4):715-770, 2002.
Wu, Yan & J. Cai, H. (2010). A Simulation Study of Deep Belief Network Combined with the Self-Organizing Mechanism of Adaptive Resonance Theory. 10.1109/CISE.2010.56//265, 4 pages.

* cited by examiner

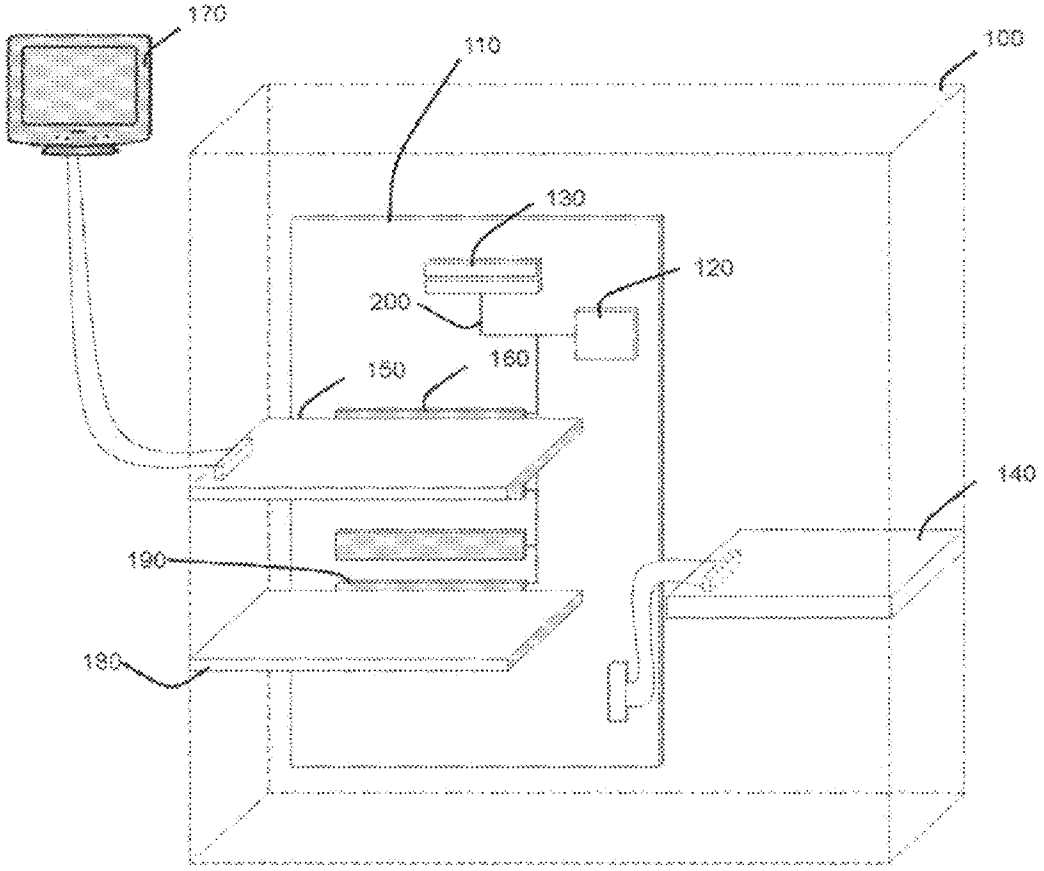


FIG. 1

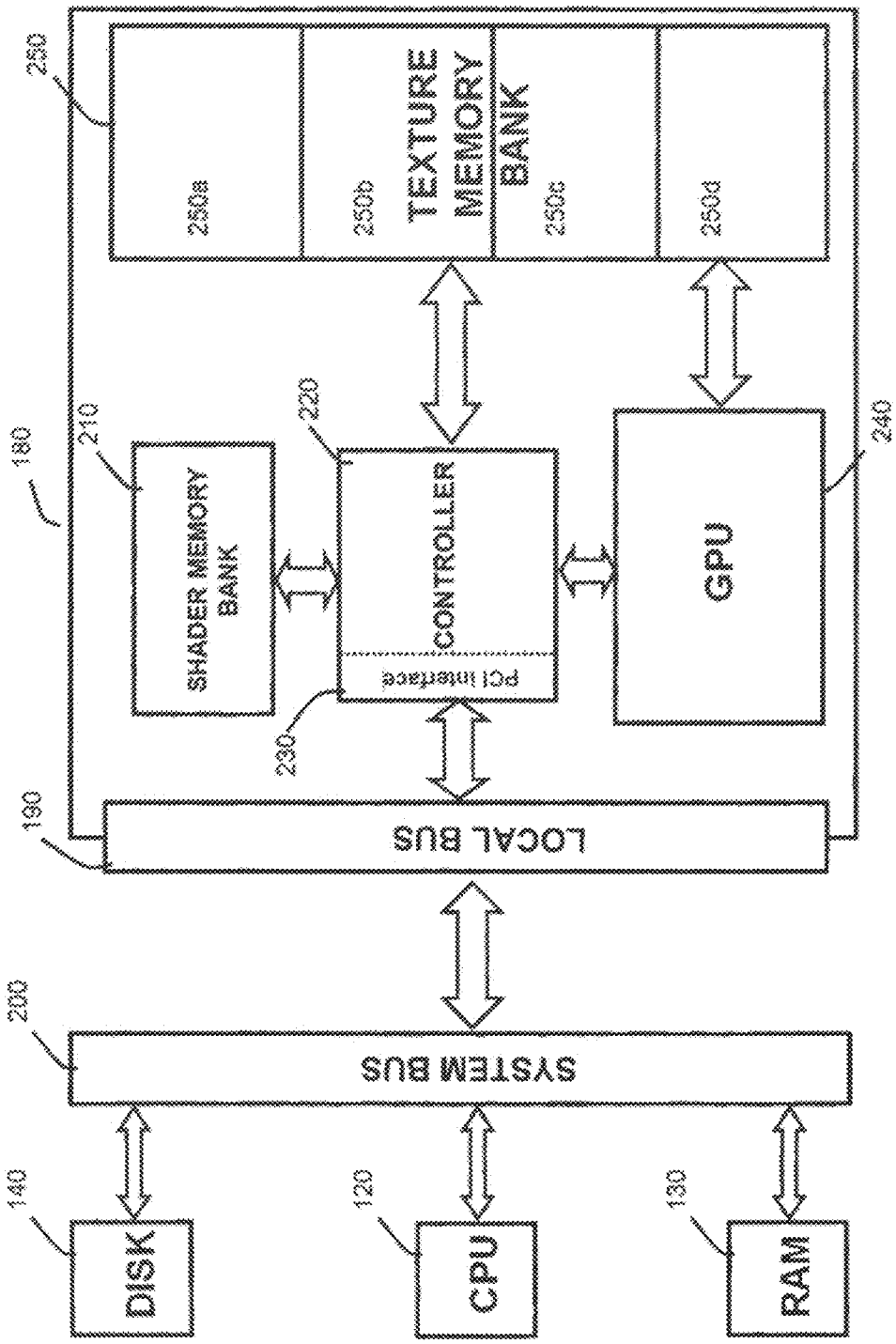

250
250a
250b
TEXTURE MEMORY BANK
250c
250d
180
210
SHADER MEMORY BANK
220
CONTROLLER
PCI Interface
230
GPU
240
190
LOCAL BUS
200
SYSTEM BUS
140
DISK
120
CPU
130
RAM

FIG. 2

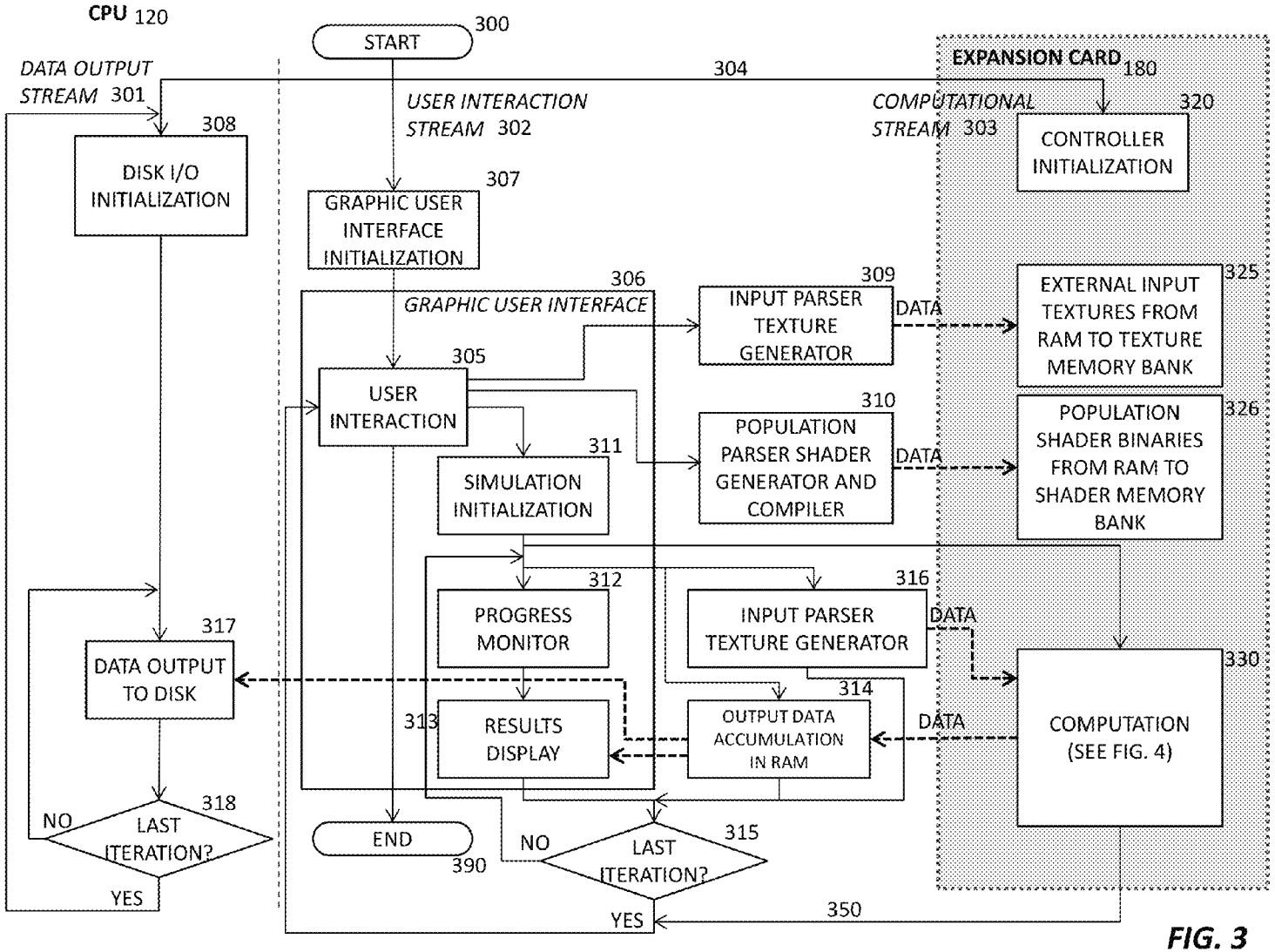

CPU 120
START
300
DATA OUTPUT STREAM 301
304
EXPANSION CARD 180
USER INTERACTION STREAM 302
COMPUTATIONAL STREAM 303
320
CONTROLLER INITIALIZATION
308
DISK I/O INITIALIZATION
307
GRAPHIC USER INTERFACE INITIALIZATION
306
GRAPHIC USER INTERFACE
309
INPUT PARSER TEXTURE GENERATOR
DATA
325
EXTERNAL INPUT TEXTURES FROM RAM TO TEXTURE MEMORY BANK
305
USER INTERACTION
310
POPULATION PARSER SHADER GENERATOR AND COMPILER
DATA
326
POPULATION SHADER BINARIES FROM RAM TO SHADER MEMORY BANK
311
SIMULATION INITIALIZATION
312
PROGRESS MONITOR
316
INPUT PARSER TEXTURE GENERATOR
DATA
317
DATA OUTPUT TO DISK
330
COMPUTATION (SEE FIG. 4)
314
OUTPUT DATA ACCUMULATION IN RAM
DATA
313
RESULTS DISPLAY
318
NO
LAST ITERATION?
YES
END
390
NO
LAST ITERATION?
315
YES
350

FIG. 3

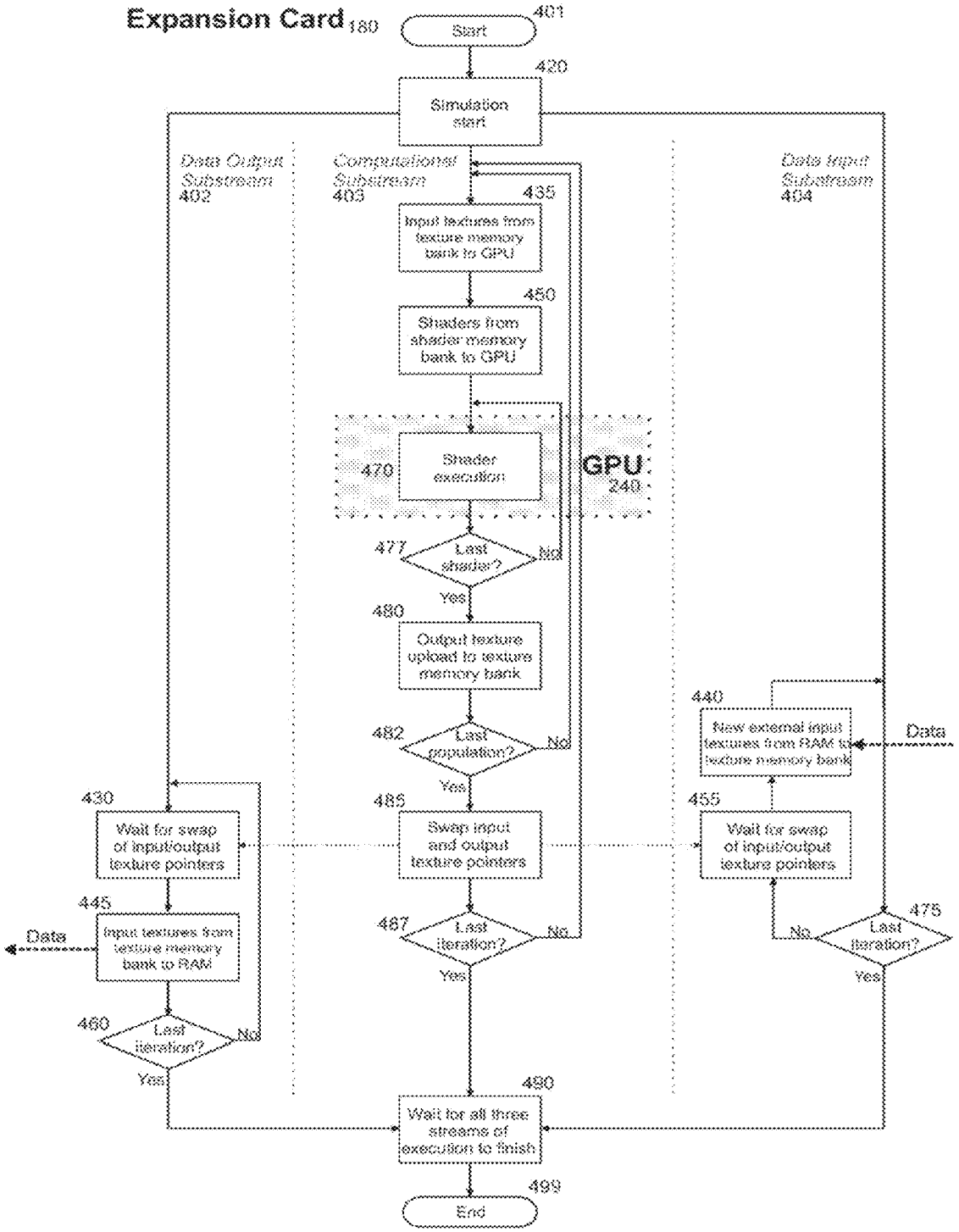
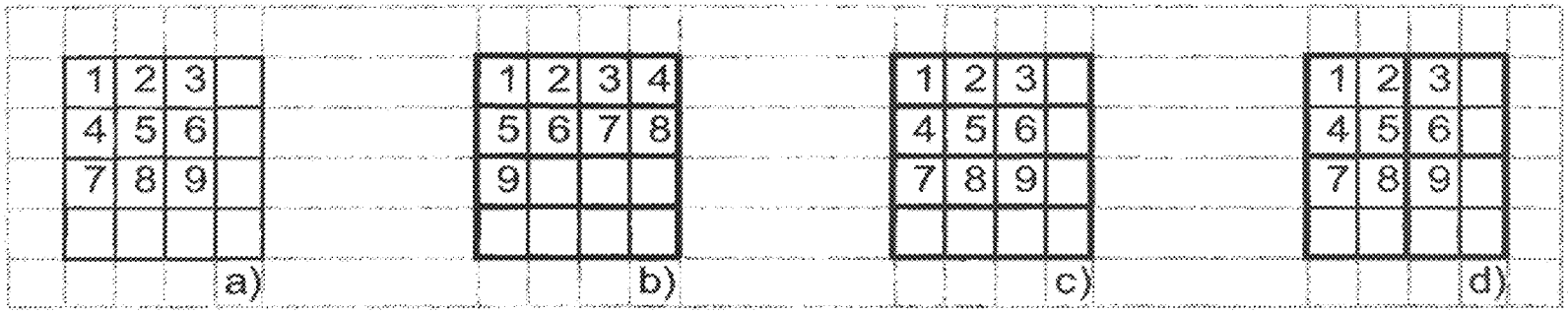

Expansion Card 180
Start
401
420
Simulation start
Data Output Substream 402
Computational Substream 403
Data Input Substream 404
435
Input textures from texture memory bank to GPU
450
Shaders from shader memory bank to GPU
470
Shader execution
GPU 240
477
Last shader?
No
Yes
480
Output texture upload to texture memory bank
482
Last population?
No
Yes
440
New external input textures from RAM to texture memory bank
Data
430
Wait for swap of input/output texture pointers
485
Swap input and output texture pointers
455
Wait for swap of input/output texture pointers
445
Input textures from texture memory bank to RAM
Data
467
Last iteration?
No
Yes
475
Last iteration?
No
Yes
460
Last iteration?
No
Yes
490
Wait for all three streams of execution to finish
499
End

FIG 4



Bold lines show the packing of data into pixels with four color components.

FIG. 5

GRAPHIC PROCESSOR BASED ACCELERATOR SYSTEM AND METHOD

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue; a claim printed with strikethrough indicates that the claim was canceled, disclaimed, or held invalid by a prior post-patent action or proceeding.

RELATED APPLICATIONS

The present application *is a reissue continuation application of U.S. application Ser. No. 15/808,201, which was filed on Nov. 9, 2017 as a broadening reissue application of U.S. Pat. No. 9,189,828, filed Jan. 3, 2014, which* claims a priority benefit, under 35 U.S.C. §120, as a continuation of U.S. application Ser. No. 11/860,254, now U.S. Pat. No. 8,648,867 B2, filed Sep. 24, 2007, entitled "Graphic Processor Based Accelerator System and Method," which *in turn* claims the priority benefit, under 35 U.S.C. §119(e), of U.S. Application No. 60/826,892, filed Sep. 25, 2006. *The present application is also a broadening reissue application of U.S. Pat. No. 9,189,828, filed Jan. 3, 2014, which is a continuation of U.S. application Ser. No. 11/860,254, now U.S. Pat. No. 8,648,867 B2, filed Sep. 24, 2007, which in turn claims the priority benefit, under 35 U.S.C. § 119(e), of U.S. Application No. 60/826,892, filed Sep. 25, 2006.* Each of the above-identified applications is incorporated herein by reference in its entirety. *More than one reissue application has been filed for the reissue of U.S. Pat. No. 9,189,828, including this application and U.S. application Ser. No. 15/808,201.*

BACKGROUND

Graphics Processing Units (GPUs) are found in video adapters (graphic cards) of most personal computers (PCs), video game consoles, workstations, etc. and are considered highly parallel processors dedicated to fast computation of graphical content. With the advances of the computer and console gaming industries, the need for efficient manipulation and display of 3D graphics has accelerated the development of GPUs.

In addition, manufacturers of GPUs have included general purpose programmability into the GPU architecture leading to the increased popularity of using GPUs for highly parallelizable and computationally expensive algorithms outside of the computer graphics domain. When implemented on conventional video card architectures, these general purpose GPU (GPGPU) applications are not able to achieve optimal performance, however. There is overhead for graphics-related features and algorithms that are not necessary for these non-video applications.

SUMMARY

Numerical simulations, e.g., finite element analysis, of large systems of similar elements (e.g. neural networks, genetic algorithms, particle systems, mechanical systems) are one example of an application that can benefit from GPGPU computation. During numerical simulations, disk and user input/output can be performed independently of computation because these two processes require interactions with peripheral hardware (disk, screen, keyboard, mouse, etc) and put relatively low load on the central

processing unit/system (CPU). Complete independence is not desirable, however; user input might affect how the computation is performed and even interrupt it if necessary. Furthermore, the user output and the disk output are dependent on the results of the computation. A reasonable solution would be to separate input/output into threads, so that it is interacting with hardware occurs in parallel with the computation. In this case whatever CPU processing is required for input/output should be designed so that it provides the synchronization with computation.

In the case of GPGPU, the computation itself is performed outside of the CPU, so the complete system comprises three "peripheral" components: user interactive hardware, disk hardware, and computational hardware. The central processing unit (CPU) establishes communication and synchronization between peripherals. Each of the peripherals is preferably controlled by a dedicated thread that is executed in parallel with minimal interactions and dependencies on the other threads.

A GPU on a conventional video card is usually controlled through OpenGL, DirectX, or similar graphic application programming interfaces (APIs). Such APIs establish the context of graphic operations, within which all calls to the GPU are made. This context only works when initialized within the same thread of execution that uses it. As a result, in a preferred embodiment, the context is initialized within a computational thread. This creates complications, however, in the interaction between the user interface thread that changes parameters of simulations and the computational thread that uses these parameters.

A solution as proposed here is an implementation of the computational stream of execution in hardware, so that thread and context initialization are replaced by hardware initialization. This hardware implementation includes an expansion card comprising a printed circuit board having (a) one or more graphics processing units, (b) two or more associated memory banks that are logically or physically partitioned, (c) a specialized controller, and (d) a local bus providing signal coupling compatible with the PCI industry standards (this includes but is not limited to PCI-Express, PCI-X, USB 2.0, or functionally similar technologies). The controller handles most of the primitive operations needed to set up and control GPU computation. As a result, the CPU is freed from this function and is dedicated to other tasks. In this case a few controls (simulation start and stop signals from the CPU and the simulation completion signal back to CPU), GPU programs and input/output data are the information exchanged between CPU and the expansion card. Moreover, since on every time step of the simulation the results from the previous time step are used but not changed, the results are preferably transferred back to CPU in parallel with the computation.

In general, according to one aspect, the invention features a computer system. This system comprises a central processing unit, main memory accessed by the central processing unit, and a video system for driving a video monitor in response to the central processing unit as is common. The computer system further comprises an accelerator that uses input data from and provides output data to the central processing unit. This accelerator comprises at least one graphics processing unit, accelerator memory for the graphic processing unit, and an accelerator controller that moves the input data into the at least one graphics processing unit and the accelerator memory to generate the output data.

In the preferred, the central processing unit transfers the input data for a simulation to the accelerator, after which the accelerator executes simulation computations to generate

the output data, which is transferred to the central processing unit. Preferably, the accelerator controller dictates an order of execution of instructions to the at least one graphics processing unit. The use of the separate controller enables data transfer during execution such that the accelerator controller transfers output data from the accelerator memory to main memory of the central processing unit.

In the preferred embodiment, the accelerator controller comprises an interface controller that enables the accelerator to communicate over a bus of the computer system with the central processing unit.

In general according to another aspect, the invention also features an accelerator system for a computer system, which comprises at least one graphics processing unit, accelerator memory for the graphic processing unit and an accelerator controller for moving data between the at least one graphics processing unit and the accelerator memory.

In general according to another aspect, the invention also features a method for performing numerical simulations in a computer system. This method comprises a central processing unit loading input data into an accelerator system from main memory of the central processing unit and an accelerator controller transferring the input data to a graphics processing unit with instructions to be performed on the input data. The accelerator controller then transfers output data generated by the graphic processing unit to the central processing unit as output data.

The above and other features of the invention including various novel details of construction and combinations of parts, and other advantages, will now be more particularly described with reference to the accompanying drawings and pointed out in the claims. It will be understood that the particular method and device embodying the invention are shown by way of illustration and not as a limitation of the invention. The principles and features of this invention may be employed in various and numerous embodiments without departing from the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

In the accompanying drawings, reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale; emphasis has instead been placed upon illustrating the principles of the invention. Of the drawings:

FIG. **1** is a schematic diagram illustrating a computer system including the GPU accelerator according to an embodiment of the present invention;

FIG. **2** is block diagram illustrating the architecture for the GPU accelerator according to an embodiment of the present invention;

FIG. **3** is a block/flow diagram illustrating an exemplary implementation of the top level control of the GPU accelerator system;

FIG. **4** is a flow diagram illustrating an exemplary implementation of the bottom level control of the GPU accelerator system that is used to execute the target computation; and

FIG. **5** is an example population of nine computational elements arranged in a 3×3 square and a potential packing scheme for texture pixels, according to an implementation of the present invention.

DETAILED DESCRIPTION

FIG. **1** shows a computer system **100** that has been constructed according to the principles of the present invention.

In more detail, the computer system **100** in one example is a standard personal computer (PC). However, this only serves as an example environment as computing environment **100** does not necessarily depend on or require any combination of the components that are illustrated and described herein. In fact, there are many other suitable computing environments for this invention, including, but not limited to, workstations, server computers, supercomputers, notebook computers, hand-held electronic devices such as cell phones, mp3 players, or personal digital assistants (PDAs), multiprocessor systems, programmable consumer electronics, networks of any of the above-mentioned computing devices, and distributed computing environments that including any of the above-mentioned computing devices.

In one implementation the GPU accelerator is implemented as an expansion card **180** includes connections with the motherboard **110**, on which the one or more CPU's **120** are installed along with main, or system memory **130** and mass/non volatile data storage **140**, such as hard drive or redundant array of independent drives (RAID) array, for the computer system **100**. In the current example, the expansion card **180** communicates to the motherboard **110** via a local bus **190**. This local bus **190** could be PCI, PCI Express, PCI-X, or any other functionally similar technology (depending upon the availability on the motherboard **110**). An external version GPU accelerator is also a possible implementation. In this example, the external GPU accelerator is connected to the motherboard **110** through USB-2.0, IEEE 1394 (Firewire), or similar external/peripheral device interface.

The CPU **120** and the system memory **130** on the motherboard **110** and the mass data storage system **140** are preferably independent of the expansion card **180** and only communicate with each other and the expansion card **180** through the system bus **200** located in the motherboard **110**. A system bus **200** in current generations of computers have bandwidths from 3.2 GB/s (Pentium 4 with AGTL+, Athlon XP with EV6) to around 15 GB/s (Xeon Woodcrest with AGTL+, Athlon 64/Opteron with Hypertransport), while the local bus has maximal peak data transfer rates of 4 GB/s (PCI Express 16) or 2 GB/s (PCI-X 2.0). Thus the local bus **190** becomes a bottleneck in the information exchange between the system bus **200** and the expansion card **180**. The design of the expansion card and methods proposed herein minimizes the data transfer through the local bus **190** to reduce the effect of this bottleneck.

The system memory **130** is referred to as the main random-access memory (RAM) in the description herein. However, this is not intended to limit the system memory **130** to only RAM technology. Other possible computer storage media include, but are not limited to ROM, EEPROM, flash memory, or any other memory technology.

In the illustrated example, the GPU accelerator system is implemented on an expansion card **180** on which the one or more GPU's **240** are mounted. It should be noted that the GPU accelerator system GPU **240** is separate from and independent of any GPU on the standard video card **150** or other video driving hardware such as integrated graphics systems. Thus the computations performed on the expansion card **180** do not interfere with graphics display (including but not limited to manipulation and rendering of images).

Various brand of GPU are relevant. Under current technology, GPU's based on the GeForce series from NVIDIA Corporation or the Catalyst series from ATI/Advanced Micro Devices, Inc.

The output to a video monitor **170** is preferably through the video card **150** and not the GPU accelerator system **180**. The video card **150** is dedicated to the transfer of graphical information and connects to the motherboard **110** through a local bus **160** that is sometimes physically separate from the local bus **190** that connects the expansion card **180** to the motherboard **110**.

FIG. **2** is a block diagram illustrating the general architecture of the GPU accelerator system and specifically the expansion card **180** in which at least one GPU **240** and associated memories **210** and **250** are mounted. Electrical (signal) and mechanical coupling with a local bus **190** provides signal coupling compatible with the PCI industry standards (this includes but is not limited to PCI, PCI-X, PCI Express, or functionally similar technology).

The GPU accelerator further preferably comprises one specifically designed accelerator controller **220**. Depending upon the implementation, the accelerator controller **220** is field programmable gate array (FPGA) logic, or custom built application-specific (ASIC) chip mounted in the expansion card **180**, and in mechanical and signal coupling with the GPU **240** and the associated memories **210** and **250**. During initial design, a controller can be partially or even fully implemented in software, in one example.

The controller **220** commands the storage and retrieval of arrays of data (on a conventional video card the arrays of data are represented as textures, hence the term 'texture' in this document refers to a data array unless specified otherwise and each element of the texture is a pixel of color information), execution of GPU programs (on a conventional video card these programs are called shaders, hence the term 'shader' in this document refers to a GPU program unless specified otherwise), and data transfer between the system bus **200** and the expansion card **180** through the local bus **190** which allows communication between the main CPU **120**, RAM **130**, and disk **140**.

Two memory banks **210** and **250** are mounted on the expansion card **180**. In some example, these memory banks separated in the hardware, as shown, or alternatively implemented as a single, logically partitioned memory component.

The reason to separate the memory into two partitions **210** **250** stems from the nature of the computations to which the GPU accelerator system is applied. The elements of computation (computational elements) are characterized by a single output variable. Such computational elements often include one or more equations. Computational elements are same or similar within a large population and are computed in parallel. An example of such a population is a layer of neurons in an artificial neural network (ANN), where all neurons are described by the same equation. As a result, some data and most of the algorithms are common to all computational elements within population, while most of the data and some algorithms are specific for each equation. Thus, one memory, the shader memory bank **210**, is used to store the shaders needed for the execution of the required computations and the parameters that are common for all computational elements and is coupled with the controller **220** only. The second memory, the texture memory bank **250**, is used to store all the necessary data that are specific for every computational element (including, but not limited to, input data, output data, intermediate results, and parameters) and is coupled with both the controller **220** and the GPU **240**.

The texture memory bank **250** is preferably further partitioned into four sections. The first partition **250** a is designed to hold the external input data patterns. The second partition **250**b is designed to hold the data textures representing internal variables. The third partition **250**c is designed to hold the data textures used as input at a particular computation step on the GPU **240**. The fourth partition **250**d holds the data textures used to accommodate the output of a particular computational step on the GPU **240**. This partitioning scheme can be done logically, does not require hardware implementation. Also the partitioning scheme is also altered based on new designs or needs of the algorithms being employed. The reason for this partitioning is further explained in the Data Organization section, below.

A local bus interface **230** on the controller **220** serves as a driver that allows the controller **220** to communicate through the local bus **190** with the system bus **200** and thus the CPU **120** and RAM **130**. This local bus interface **230** is not intended to be limited to PCI related technology. Other drivers can be used to interface with comparable technology as a local bus **190**.

Data Organization

Each computational element discussed above has output variables that affect the rest of the system. For example in the case of a neural network it is the output of a neuron. A computational element also usually has several internal variables that are used to compute output variables, but are not exposed to the rest of the system, not even to other elements of the same population, typically. Each of these variables is represented as a texture. The important difference between output variables and internal variables is their access.

Output variables are usually accessed by any element in the system during every time step. The value of the output variable that is accessed by other elements of the system corresponds to the value computed on the previous, not the current, time step. This is realized by dedicating two textures to output variables—one holds the value computed during the previous time step and is accessible to all computational elements during the current time step, another is not accessible to other elements and is used to accumulate new values for the variable computed during the current time step. In-between time steps these two textures are switched, so that newly accumulated values serve as accessible input during the next time step, while the old input is replaced with new values of the variable. This switch is implemented by swapping the address pointers to respective textures as described in the System and Framework section.

Internal variables are computed and used within the same computational element. There is no chance of a race condition in which the value is used before it is computed or after it has already changed on the next time step because within an element the processing is sequential. Therefore, it is possible to render the new value of internal variable into the same texture where the old was read from in the texture memory bank. Rendering to more than one texture from a single shader is not implemented in current GPU architectures, so computational elements that track internal variables would have to have one shader per variable. These shaders can be executed in order with internal variables computed first, followed by output variables.

Further savings of texture memory is achieved through using multiple color components per pixel (texture element) to hold data. Textures can have up to four color components that are all processed in parallel on a GPU. Thus, to maximize the use of GPU architecture it is desirable to pack the data in such a way that all four components are used by the algorithm. Even though each computational element can have multiple variables, designating one texture pixel per element is ineffective because internal variables require one

texture and output variables require two textures. Furthermore, different element types have different numbers of variables and unless this number is precisely a multiple of four, texture memory can be wasted.

A more reasonable packing scheme would be to pack four computational elements into a pixel and have separate textures for every variable associated with each computational element. In this case the packing scheme is identical for all textures, and therefore can be accessed using the same algorithm. Several ways to approach this packing scheme are outlined here. An example population of nine computational elements arranged in a 3×3 square (FIG. **5**a) can be packed by element (FIG. **5**b), by row (FIG. **5**c), or by square (FIG. **5**d).

Packing by element (FIG. **5**b) means that elements **1**,**2**,**3**,**4** go into first pixel; **5**,**6**,**7**,**8** go into second pixel; **9** goes into third pixel. This is the most compact scheme, but not convenient because the geometrical relationship is not preserved during packing and its extraction depends on the size of the population.

Packing by row (column; FIG. **5**c) means that elements **1**,**2**,**3** go into pixel (**1**,**1**); **3**,**4**,**5** go into pixel (**2**,**1**), **7**,**8**,**9** go into pixel (**3**,**1**). With this scheme the element's y coordinate in the population is the pixel's y coordinate, while the element's x coordinate in the population is the pixel's x coordinate times four plus the index of color component. Five by five populations in this case will use 2×5 texture, or 10 pixels. Five of these pixels will only use one out of four components, so it wastes 37.5% of this texture. 25×1 population will use 6×1 texture (six pixels) and will waste 12.5% of it.

Packing by square (FIG. **5**d) means that elements **1**,**2**,**4**,**5** go into pixel (**1**,**1**); **3**,**6** go into pixel (**1**,**2**); **7**,**8** go into pixel (**2**,**1**), and **9** goes into pixel (**2**,**2**). Both the row and the column of the element are determined from the row (column) of the pixel times two plus the second (first) bit of the color component index. Five by five populations in this case will use 3×3 texture, or 9 pixels. Four of these pixels will only use two out of four components, and one will only use one component, so it wastes 34.4% of this texture. This is more advantageous than packing by row, since the texture is smaller and the waste is also lower. 25×1 population on the other hand will use 13×1 texture (thirteen pixels) and waste >50% of it, which is much worse than packing by row.

In order to eliminate waste altogether the population should have even dimensions in the square packing, and it should have a number of columns divisible by four in row packing. Theoretically, the chances are approximately equivalent for both of these cases to occur, so the particular task and data sizes should determine which packing scheme is preferable in each individual case.

The System and Framework

FIG. **3** shows an exemplary implementation of the top level system and method that is used to control the computation. It is a representation of one of several ways in which a system and method for processing numerical techniques can be implemented in the invention described herein and so the implementation is not intended to be limited to the following description and accompanying figure.

The method presented herein includes two execution streams that run on the CPU **120**—User Interaction Stream **302** and Data Output Stream **301**. These two streams preferably do not interact directly, but depend on the same data accumulated during simulations. They can be implemented as separate threads with shared memory access and executed on different CPUs in the case of multi-CPU computing environment. The third execution stream—Computational Stream **303**—runs on the GPU accelerator of the expansion card **180** and interacts with the User Interaction Stream **302** through initialization routines and data exchange in between simulations. The Computational Stream **303** interacts with the User Interaction Stream and the Data Output Stream through synchronization procedures during simulations.

The crucial feature of the interaction between the User Interaction Stream **302** and the Computational Stream **303** is the shift of priorities. Outside of the simulation, the system **100** is driven by the user input, thus the User Interaction Stream **302** has the priority and controls the data exchange **304** between streams. After the user starts the simulation, the Computational Stream **303** takes the priority and controls the data exchange between streams until the simulation is finished or interrupted **350**.

The user starts **300** the framework through the means of an operating system and interacts with the software through the user interaction section **305** of the graphic user interface **306** executed on the CPU **120**. The start **300** of the implementation begins with a user action that causes a GUI initialization **307**, Disk input/output initialization **308** on the CPU **120**, and controller initialization **320** of the GPU accelerator on the expansion card **180**. GUI initialization includes opening of the main application window and setting the interface tools that allow the user to control the framework. Disk I/O initialization can be performed at the start of the framework, or at the start of each individual simulation.

The user interaction **305** controls the setting and editing of the computational elements, parameters, and sources of external inputs. It specifies which equations should have their output saved to disk and/or displayed on the screen. It allows the user to start and stop the simulation. And it performs standard interface functions such as file loading and saving, interactive help, general preferences and others.

The user interaction **305** directs the CPU **120** to acquire the new external input textures needed (this includes but is not limited to loading from disk **140** or receiving them in real time from a recording device), parses them if necessary **309**, and initializes their transfer to the expansion card **180**, where they are stored **325** in the texture memory bank **250** by the controller **220**. The user interaction **305** also directs the CPU **120** to parse populations of elements that will be used in the simulation, convert them to GPU programs (shaders), compile them **310**, and initializes their transfer to the expansion card **180**, where they are stored **326** in the shader memory bank **210** by the controller **220**. This operation is accompanied by the upload **309** of the initial data into the input partition of the texture memory bank **250**, and stores the shader order of execution in the controller **220**. The user can perform operations **309** and **310** as many times as necessary prior to starting the simulation or between simulations.

The editing of the system between simulations is difficult to accomplish without the hardware implementation of the computational thread suggested herein. The system of equations (computational elements) is represented by textures that track variables plus shaders that define processing algorithms. As mentioned above, textures, shaders and other graphics related constructs can only be initialized within the rendering context, which is thread specific. Therefore textures and shaders can only be initialized in the computational thread.

Network editing is a user-interactive process, which according to the scheme suggested above happens in the User Interaction Stream **302**. The simulation software thus has to take the new parameters from the User Interaction Stream **302**, communicate them to the Computational

Stream **303** and regenerate the necessary shaders and textures. This is hard to accomplish without a hardware implementation of the Computational Stream **303**. The Computational Stream **303** is forked from the User Interaction Stream and it can access the memory of the parent thread, but the reverse communication is harder to achieve. The controller **220** allows operations **309** and **310** to be performed as many times as necessary by providing the necessary communication to the User Interaction Stream **302**.

After execution of the input parser texture generation **309** and population parser shader generator and compiler **310** are performed at least once, the user has the option to initialize the simulation **311**. During this initialization the main control of the framework is transferred to the GPU accelerator system's accelerator controller **220** and computation **330** is started (see FIG. **4**; **420**). The user retains the ability to interrupt the simulation, change the input, or to change the display properties of the framework, but these interactions are queued to be performed at times determined by the controller-driven data exchange **314** and **316** to avoid the corruption of the data.

The progress monitor **312** is not necessary for performance, but adds convenience. It displays the percentage of completed time steps of the simulation and allows the user to plan the schedule using the estimates of the simulation wall clock times. Controller-driven data exchange **314** updates the display of the results **313**. Online screen output for the user selected population allows the user to monitor the activity and evaluate the qualitative behavior of the network. Simulations with unsatisfactory behavior can be terminated early to change parameters and restart. Controller-driven data exchange **314** also drives the output of the results to disk **317**. Data output to disk for convenience can be done on an element per file basis. A suggested file format includes a leftmost column that displays a simulated time for each of the simulation steps and subsequent columns that display variable values during this time step in all elements with identical equations (e.g. all neurons in a layer of a neural network).

Controller-driven data exchange or input parser texture generator **316** allows the user to change input that is generated on the fly during the simulation. This allows the framework monitoring of the input that is coming from a recording device (video camera, microphone, cell recording electrode, etc) in real time. Similar to the initial input parser **309**, it preprocesses the input into a universal format of the data array suitable for texture generation and generates textures. Unlike the initial parser **309**, here the textures are transferred to hardware not whenever ready but upon the request of the controller **220**.

The controller **220** also drives the conditional testing **315** and **318** informs the CPU-bound streams whether the simulation is finished. If so, the control returns to the User Interaction Stream. The user then can change parameters or inputs (**309** and **310**), restart the simulation (**311**) or quit the framework (**390**).

SANNDRA (Synchronous Artificial Neuronal Network Distributed Runtime Algorithm; http://www.kinness.net/Docs/SANNDRA/html) was developed to accelerate and optimize processing of numerical integration of large non-homogenous systems of differential equations. This library is fully reworked in its version 2.x.x to support multiple computational backends including those based on multicore CPUs, GPUs and other processing systems. GPU based backend for SANNDRA-2.x.x can serve as an example practical software implementation of the method and architecture described above and pictorially represented in FIG. **3**.

To use SANNDRA, the application should create a TSimulator object either directly or through inheritance. This object will handle global simulation properties and control the User Interaction Stream, Data Output Stream, and Computational Stream. Through TSimulator::timestep( ) TSimulator::outfileInterval( ), and TSimulator::outmode( ), the application can set the time step of the simulation, the time step of disk output, and the mode of the disk output. The external input pattern should be packed into a TPattern object and bound to the simulation object through TSimulator::resetInputs( ) method. TSimulator::simLength( ) sets the length of the simulation.

The second step is to create at least one population of equations (Tpopulation object). Population holds one equation object TEquation. This object contains only a formula and does not hold element-specific data, so all elements of the population can share single TEquation.

The TEquation object is converted to a GPU program before execution. GPU programs have to be executed within a graphical context, which is stream specific. TSimulator creates this context within a Computational Stream, therefore all programs and data arrays that are necessary for computation have to be initialized within Computational Stream. Constructor of TPopulation is called from User Interaction Stream, so no GPU-related objects can be initialized in this constructor.

TPopulation::fillElements( ) is a virtual method designed to overcome this difficulty. It is called from within the Computational Stream after TSimulator::networkCreate( ) is called in the User Interaction Stream. A user has to override TPopulation::fillElements( ) to create TEquation and other computation related objects both element independent and element-specific. Element independent objects include subcomponents of TEquation and objects that describe how to handle interdependencies between variables implemented through derivatives of TGate class.

Element-specific data is held in TElement objects. These objects hold references to TEquation and a set of TGate objects. There is one TElement per population, but the size of data arrays within this object corresponds to population size. All TElement objects have to be added to the TSimulator list of elements by calling TSimulator::addUnit( ) method from TPopulation::fillElements( ).

Finally, TPopulation::fillElements( ) should contain a set of TElement::add*Dependency( ) calls for each element. Each of these calls sets a corresponding dependency for every TGate object. Here TGate object holds element independent part of dependency and TElement::add*Dependency( ) sets element-specific details.

System provided TPopulation handles the output of computational elements, both when they need to exchange the data and when they need to output it to disk. User implementation of TPopulation derivative can add screen output.

Listing 1 is an example code of the user program that uses a recurrent competitive field (RCF) equation:

LISTING 1

```
uint16_t w = 3, h = 3;
static float m_compet = 0.5;
static float m_persist = 1.0;
class TCablePopRCF : public TPopulation
{
TEq_RCF* m_equation;
```

LISTING 1-continued

```
TGate* m_gatel;
TGate* m_gate2;
void createGatingStructure( )
{
m_gatel = new TGate(0);
m_gate2 = new TGate(1);
};
void createUnitStructure(TBasicUnit* u)
{
u->addO2OPInputDependency(m_gatel, 0., 0., 0.004, 0., 0, 0);
u->addFullDependency(m_gate2, population( ));
}
public: TCablePopRCF( ) : TPopulation("compCPU RCF", w, h, true) { };
~TCablePopRCF( ) {if(m_equation) delete m_equation;
   if(m_gatel) delete m_gatel;
   if(m_gate2) delete m_gate2;};
bool fillElements(TSimulator* sim);
};
bool TCablePopRCF::fillElements(TSimulatior* sim)
{
m_equation = new TEq_RCF(this, m_compet, m_persist);
createGatingStructure( );
for(size_t i = 0; i < xSize( ); ++i)
for(size_t j = 0; j < ySize( ); ++j)
{
TElement* u = new TCPUElement(this, m_equation, i, j);
sim->addUnit(u);
createUnitStructure(u);
}
Return true;
}
int
main( )
{
// Input pattern generation (309 in FIG.3)
uint32_t* pat = new uint32_t[w*h];
TRandom<float> randGen (0);
for(uint32_t I = 0; I < w*h; ++i)
pat[i] = randGen.random( );
Tpattern* p = new Tpattern(pat, w, h);
// Setting up the simulation
TSimulator* cableSim = new TSimulator("data"); //(308 and 320 in
FIG. 3)
cableSim->timestep(0.05); //(320 in FIG. 3)
cableSim->resetInputs(p); //(325 in FIG. 3)
cableSim->outfileInterval(0.1); //(308 in FIG. 3)
cableSim->outmode(SANNDRA::timefunc); //(308 in FIG. 3)
cableSim->simLength(60.0); //(320 in FIG. 3)
// Preparing the population
TPopulation* cablePop = new TCablePopRCF( ); //(310 in FIG. 3)
cableSim->networkCreate( ); //(326 in FIG. 3)
uint16_t user= 1;
while(user)
{
if(! cableSim->simulationStart(true, 1)) //(311 in FIG. 3)
exit(1);
std::cout<<"Repeat?\n"; //(305 in FIG. 3)
std::cin>>user; //(305 in FIG. 3)
if(user == 1)
cableSim->networkReset( ); //(305 in FIG. 3)
{
If(cableSim)
Delete cableSim; //Also deletes cablePop and its internals
exit(0);
};
```

FIG. **4** is a detailed flow diagram illustrating a part of an exemplary implementation of the bottom level system and method performed during the computation on the GPU accelerator of the expansion card **180** and is a more detailed view of the computational box **330** in FIG. **3**. FIG. **4** is a representation of one of several ways in which a system and method for processing numerical techniques can be implemented.

With systems of equations that have complex interdependencies it is likely that the variable in some equation from a previous time step has to be used by some other equation after the new values of this variable are already computed for new time step. To avoid data confusion, the new values of variables should be rendered in a separate texture. After the time step is completed for all equations, these new values should be copied over old values so that they are used as input during the next time step. Copying textures is an expensive operation, computationally, but since the textures are referred to by texture IDs (pointers), swapping these pointers for input and output textures after each time step achieves the same result at a much lesser cost.

In the hardware solution suggested herein, ID swapping is equivalent to swapping the base memory address for two partitions of the texture memory bank **250**. They are swapped **485** during synchronization (**485**, **430**, and **455**) so that data transfer **445** and the computation **435-487** proceeds immediately and in parallel with data transfer as shown in FIG. **4**. A hardware solution allows this parallelism through access of the controller **220** to the onboard texture memory bank **250**.

The main computation and data exchange are executed by the controller **220**. It runs three parallel substreams of execution: Computational Substream **403**, Data Output Substream **402**, and Data Input Substream **404**. These streams are synchronized with each other during the swap of pointers **485** to the input and output texture memory partitions of the texture memory bank **250** and the check for the last iteration **487**. Algorithmically, these two operations are a single atomic operation, but the block diagram shows them as two separate blocks for clarity.

The Computational Substream **403** performs a computational cycle including a sequential execution of all shaders that were stored in the shader memory bank **210** using the appropriate input and output textures. To begin the simulation the controller **220** initializes three execution sub streams **403**, **402**, and **404**. On every simulation step, the Computational Substream **403** determines which textures the GPU **240** will need to perform the computations and initiates the upload **435** of them onto the GPU **240**. The GPU **240** can communicate directly with the texture memory bank **250** to upload the appropriate texture to perform the computations. The controller **220** also pulls the first shader (known by the stored order) from the shader memory bank **210** and uploads **450** it onto the GPU **240**.

The GPU **240** executes the following operations in this order: performs the computation (execution of the shader) **470**; tells the controller **220** that it is done with the computations for the current shader; and after all shaders for this particular equation are executed sends **480** the output textures to the output portion of the texture memory bank **250**. This cycle continues through all of the equations based on the branching step **482**.

An example shader that performs fourth order Runge-Kutta numerical integration is shown in Listing 2 using GLSL notation;

LISTING 2

```
uniform sampler2DRect Variable;
uniform float integration_step;
float halfstep = integration_step*0.5;
float fl_6step = integration_step/6.0;
vec4 output = texture2DRect(Variable, gl_TexCoord[0].st);
// define equation( ) here
vec4 rungekutta4(vec4 x)
{
const vec4 kl = equation(x);
const vec4 k2 = equation(x + halfstep*kl);
const vec4 k3 = equation(x + halfstep*k2);
const vec4 k4 = equation(x + integration step*k3);
return fl_6step*(k1 + 2.0*(k2 + k3) + k4);
}
Void main(void)
{
```

LISTING 2-continued

```
output += rungekutta4(output);
gl_FragColor = output;
}
```

The shader in Listing 2 can be executed on conventional video card. Using the controller **220** this code can be further optimized, however. Since the integration step does not change during the simulation, the step itself as well as the halfstep and ⅙ of the step can be computed once per simulation, and updated in all shaders by a shader update procedures **310**, **326** discussed above.

After all of the equations in the computational cycle are computed the main execution substream **403** on the controller **220** can switch **485** the reference pointers of the input and output portions of the texture memory bank **250**.

The two other substreams of execution on the controller **220** are waiting (blocks **430** and **455**, respectively) for this switch to begin their execution. The Data Input Substream **404** is controlling **440** the input of additional data from the CPU **120**. This is necessary in cases where the simulation is monitoring the changing input, for example input from a video camera or other recording device in the real time. This substream uploads new external input from the CPU **120** to the texture memory bank **250** so it can be used by the main computational substream **403** on the next computational step and waits for the next iteration **475**. The Data Output Substream **445** controls the output of simulation results to the CPU **120** if requested by the user. This substream uploads the results of the previous step to the main RAM **130** so that the CPU **120** can save them on disk **140** or show them on the results display **313** and waits for the next iteration **460**.

Since the Computational Substream **403** determines the timing of input **440** and output **445** data transfers, these data transfers are driven by the controller **220**. To further reduce the data transfer overhead (and disk **140** overhead also) the controller **220** initiates transfer only after selected computational steps. For example, if the experimental data that is simulated was recorded every 10 milliseconds (msec) and the simulation for better precision was computed every 1 msec, then only every tenth result has to be transferred to match the experimental frequency.

This solution stores two copies of output data, one in the expansion card texture memory bank **250** and another in the system RAM **130**. The copy in the system RAM **130** is accessed twice: for disk I/O and screen visualization **313**. An alternative solution would be to provide CPU **120** with a direct read access to the onboard texture memory bank **250** by mapping the memory of the hardware onto a global memory space. The alternative solution will double the communication through the local bus **190**. Since the goal discussed herein is reducing the information transfer through the local bus **190**, the former solution is favored.

The main substream **403** determines if this is the last iteration **487**. If it is the last iteration, the controller **220** waits for the all of the execution substreams to finish **490** and then returns the control to the CPU **120**, otherwise it begins the next computational cycle.

This repeats through all of the computational cycles of the simulation.

CONCLUSION

This GPU accelerator system offers the following potential advantages:

1. Limited computations on the CPU **120**. The CPU **120** is only used for user input, sending information to the controller **220**, receiving output after each computational cycle (or less frequently as defined by the user), writing this output to disk **140**, and displaying this output on the monitor **170**. This frees the CPU **120** to execute other applications and allows the expansion card to run at its full capacity without being slowed down by extensive interactions with the CPU **120**.

2. Minimizing data transfer between the expansion card **180** and the system bus **200**. All of the information needed to perform the simulations will be stored on the expansion card **180** and all simulations will take place on it. Furthermore, whatever data transfer remains necessary will take place in parallel with the computation, thus reducing the impact of this transfer on the performance.

3. New way to execute GPU programs (shaders). Previously, the CPU **120** had full control over the order of shader's execution and was required to produce specific commands on every cycle to tell the GPU **240** which shader to use. With the invention disclosed herein, shaders will initially be stored on the shader memory bank **210** on the expansion card **180** and will be sent to the GPU **240** for execution by the general purpose controller **220** located on the expansion card.

4. Multiple parallelisms. The GPU **240** is inherently parallel and is well suited to perform parallel computations. In parallel with the GPU **240** performing the next calculation, the controller **220** is uploading the data from the previous calculation into main memory **130**. Furthermore, the CPU **120** at the same time uses uploaded previous results to save them onto disk **140** and to display them on the screen through the system bus **200**.

5. Reuse of existing and affordable technology. All hardware used in the invention and mentioned here-in are based on currently available and reliable components. Further advance of these components will provide straightforward improvements of the invention.

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims.

What is claimed is:

[1. A computer system, comprising:

a central processing unit to receive input data;

main memory, operably coupled to the central processing unit via a bus, to store the input data received by the central processing unit;

an accelerator, operably coupled to the central processing unit and the first memory via the bus, to receive at least a portion of the input data from the main memory, the accelerator comprising:

at least one graphics processing unit to perform a sequence of computations on the at least a portion of the input data so as to generate output data, intermediate computations in the sequence of computations yielding intermediate results; and

accelerator memory, operably coupled to the graphic processing unit, to store the results of the plurality of sequential computations; and

a controller, operably coupled to the at least one graphics processing unit and the accelerator memory, to transfer the at least a portion of the input data into the accelerator memory, and to transfer at least a portion of the output data from the accelerator memory to the main memory during performance of the sequence of computations by the at least one graphic processing unit.**]**

[2. The computer system of claim 1, wherein the central processing unit is configured to receive the input data in response to a user interaction.]

[3. The computer system of claim 1, wherein:

the central processing unit is configured to receive the input data at a first rate; and

the at least one graphics processing unit is configured to perform the sequence of computations at a second rate different than the first rate.]

[4. The computer system of claim 1, wherein the main memory is configured to store a copy of the output data stored in the accelerator memory.]

[5. The computer system of claim 1, wherein an output of at least one computation in the sequence of computations represents an output of at least one neuron in an artificial neural network.]

[6. The computer system of claim 1, wherein accelerator memory comprises:

a first memory bank to store parameters common to all of the computations in the sequence of computations; and

a second memory bank to store data specific to at least one computation in the sequence of computations.]

[7. The computer system of claim 1, wherein the controller is configured to transfer the output data from the accelerator memory to the main memory without transferring any of the intermediate results from the accelerator memory to the main memory so as to reduce data transfer via the bus.]

[8. The computer system of claim 1, wherein the controller is configured to transfer at least a portion of the output data from the accelerator memory to the main memory after the at least one graphics processing unit has begun to perform another sequence of computations.]

[9. The computer system of claim 8, wherein the controller is configured to initiate transfer of the at least a portion of the input data and to transfer the at least a portion of the output data in parallel with performance of at least one computation in the other sequence of computations by the at least one graphics processing unit.]

[10. The computer system of claim 1, wherein the controller is configured to control execution of the sequence of computations by the at least one graphics processing unit.]

[11. The computer system of claim 1, further comprising:

at least one of a video camera, a microphone, or a cell recording electrode, operably coupled to the central processor unit, to acquire the input data in real time.]

[12. A method of performing a sequence of computations on a computer system comprising a central processing unit (CPU), a main memory operably coupled to the central processing unit via a bus, an accelerator operably coupled to the CPU and the main memory via the bus, the accelerator comprising a graphics processing unit (GPU) and an accelerator memory, the method comprising:

(A) performing, by the GPU, the sequence of computations on a first portion of the input data so as to generate a first portion of the output data, intermediate computations in the sequence of computations yielding intermediate results;

(B) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the input data from the main memory to the accelerator via the bus; and

(C) in parallel with performing the sequence of computations by the GPU in (A), transferring a second portion of the output data from the accelerator memory to the main memory via the bus.]

[13. The method of claim 12, further comprising:

storing the input data in the main memory in response to a user interaction.]

[14. The method of claim 12, further comprising:

receiving the input data at a first rate; and

wherein (A) comprises performing the sequence of computations at a second rate different than the first rate.]

[15. The method of claim 12, wherein (A) comprises:

generating an output representative of an output of at least one neuron in an artificial neural network.]

[16. The method of claim 12, wherein (C) comprises:

transferring the second portion of the output data from the accelerator memory to the main memory without transferring any of the intermediate results of the plurality of sequential computations from the accelerator memory to the main memory so as to reduce data transfer via the bus.]

[17. The method of claim 12, wherein (C) comprises:

transferring the second portion of the output data from the accelerator memory to the main memory after the GPU has begun to perform another sequence of computations.]

[18. The method of claim 17, wherein (C) further comprises:

initiating transfer of the second portion of the output data in parallel with performance of at least one computation in the other sequence of computations.]

[19. The method of claim 12, further comprising:

acquiring the input data in real time with at least one of a video camera, a microphone, or a cell recording electrode operably coupled to the CPU.]

[20. The method of claim 12, further comprising:

storing parameters common to all of the computations in the sequence of computations in a first memory bank in the accelerator memory; and

storing data specific to at least one computation in the sequence of computations in a second memory bank in the accelerator memory.]

21. A method of executing computations representing an artificial neural network on a computer system comprising at least one central processing unit (CPU), a processing unit, a first memory partition, and a second memory partition, the method comprising:

executing, by the at least one CPU, a user interaction stream, the user interaction stream controlling transfer of inputs to the artificial neural network to the first memory partition and the second memory partition;

executing, by the processing unit, a computational stream, the computational stream controlling data exchange between the user interaction stream and the computational stream during execution of the computations representing the artificial neural network;

shifting control of a data exchange between the user interaction stream and the computational stream to the computational stream in response to starting execution of the computations representing the artificial neural network;

shifting control of the data exchange between the user interaction stream and the computational stream to the user interaction stream in response to completion or interruption of the computations representing the artificial neural network;

queueing a user command received by the user interaction stream during execution of the computations representing the artificial neural network; and

executing the user command during execution of the computations representing the artificial neural network at times determined by the computational stream.

22. The method of claim 21, wherein the user interaction stream controls the data exchange between the user interaction stream and the computational stream outside of execution of the computations representing the artificial neural network.

23. The method of claim 21, wherein executing the user interaction stream comprises:

controlling setting and editing of computational elements of the computations representing the artificial neural network.

24. The method of claim 21, wherein executing the user interaction stream comprises:

controlling setting and editing of parameters of the computations representing the artificial neural network.

25. The method of claim 21, wherein executing the user interaction stream comprises:

controlling setting and editing of parameters of the inputs to the artificial neural network.

26. The method of claim 21, wherein executing the user interaction stream comprises:

specifying an output to be saved to disk and/or displayed on a screen.

27. The method of claim 21, wherein executing the user interaction stream comprises:

parsing elements to be used in the computations representing the artificial neural network.

28. The method of claim 27, wherein the processing unit comprises a graphics processing unit (GPU) and executing the user interaction stream further comprises:

converting the elements into GPU programs.

29. The method of claim 28, wherein executing the user interaction stream comprises:

compiling the GPU programs.

30. The method of claim 29, wherein executing the user interaction stream comprises:

transferring the GPU programs to the second memory partition.

31. The method of claim 21, further comprising:

executing, by the at least one CPU, a data output stream, the data output stream controlling transfer of outputs of the computations representing the artificial neural network to disk.

32. The method of claim 21, further comprising:

generating the inputs with a video camera during execution of the computations.

33. A system for executing computations representing an artificial neural network, the system comprising:

a camera to acquire input data for the artificial neural network;

a first memory partition;

a second memory partition;

at least one central processing unit (CPU), operably coupled to the camera, the first memory partition, and the second memory partition, to execute a user interaction stream, the user interaction stream controlling transfer of the input data acquired by the camera to the first memory partition and the second memory partition during execution of the computations representing the artificial neural network;

a processing unit, operably coupled to the first memory partition, the second memory partition, and the at least one CPU, to execute a computational stream, the computational stream controlling transfer of the input data from the first memory partition and the second memory partition during execution of the computations representing the artificial neural network, the execution of the computations representing the artificial neural network occurring while the camera is acquiring the input data; and

a controller, operably coupled to the at least one CPU and the processing unit, to queue user interactions received by the user interaction stream during the execution of the computations representing the artificial neural network for performance at times selected to avoid data corruption.

34. The system of claim 33, wherein the user interactions cause interruption of the computations representing the artificial neural network.

35. The system of claim 33, wherein the user interactions cause a change in inputs to the artificial neural network.

36. The system of claim 33, wherein the user interactions cause a change in display properties of an output of the computations representing the artificial neural network.

37. The system of claim 33, wherein the controller is configured to request the input data during the execution of the computations representing the artificial neural network.

38. The method of claim 21, wherein the user command causes interruption of the computations representing the artificial neural network.

39. The method of claim 21, wherein the user command causes a change in the inputs to the artificial neural network.

40. The method of claim 21, wherein the user command causes a change in display properties of an output of the computations representing the artificial neural network.

41. The method of claim 21, wherein, during execution of the computations representing the artificial neural network, the computational stream controls the data exchange between the user interaction stream and the computational stream by requesting the inputs to the artificial neural network.

* * * * *