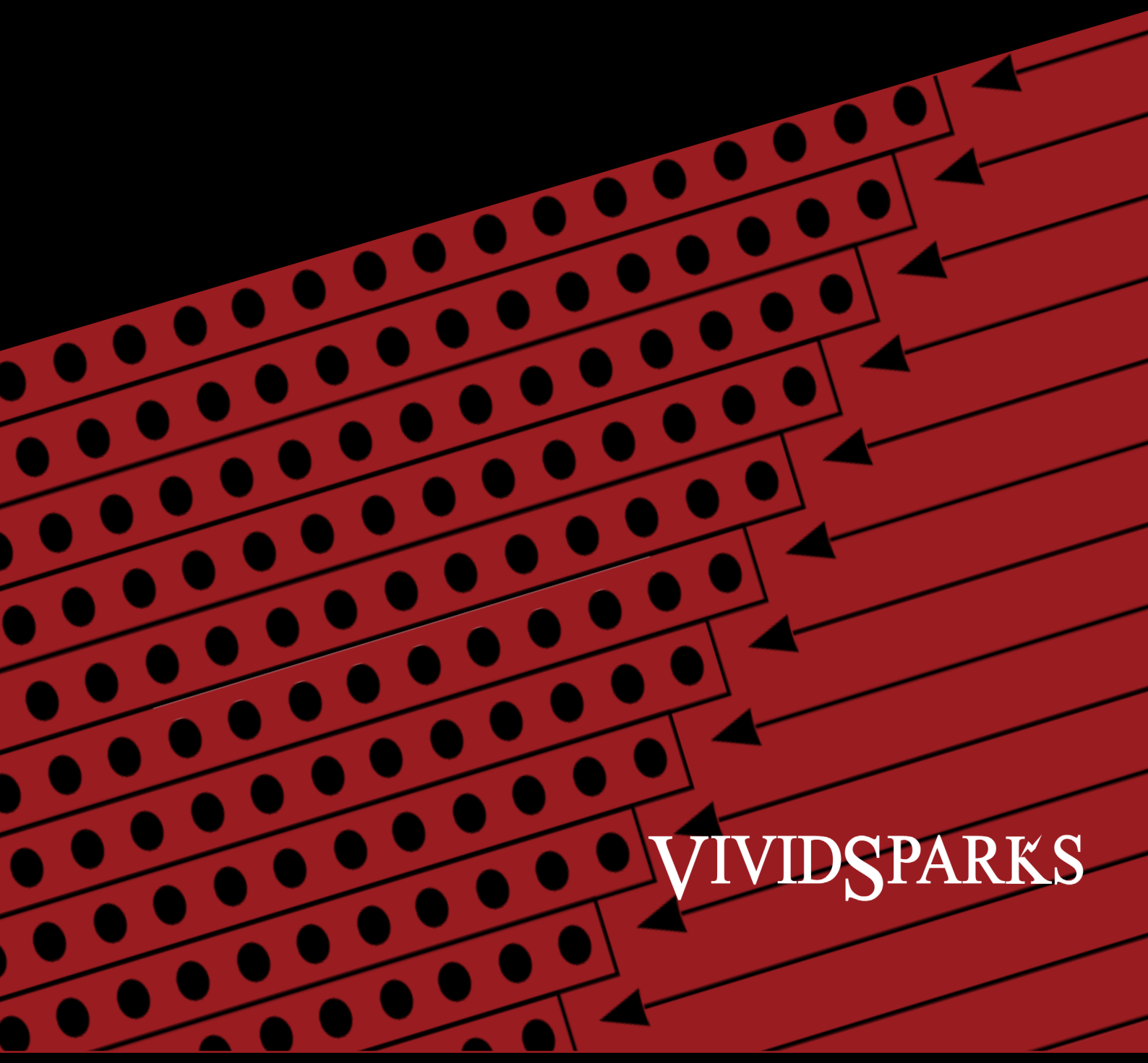
POSIT Arithmetic For Next Generation Applications



VIVIDSPARKS

Executive Summary

Leaders in computing are now profiting from their investments in new number systems initiated half a decade ago. NVIDIA transformed the AI ecosystem with their 16-bit, half-precision float providing a boost over Intel enabling them to gain market share in the valuable data center market. Google designed the Tensor Processing Unit (TPU) to accelerate AI cloud services; the TPU uses an 8-bit integer format to create a 100x benefit over its competitors. Microsoft is using an 8-bit floating point with 2-bit exponents for its Project Brainwave. And China's Huawei is using a fixed-point format for its 4G/5G base stations to gain performance per Watt benefit over its US competitors who still use IEEE floating point. All these companies realized that with Moore's Law and Denning's scaling having reached a plateau, IEEE floating point must be replaced with more efficient alternatives to support the accelerating demand for new applications in AI, mobility, and IoT. The posit number system described in this white paper is positioned to bring together all these applications and offer economies of scale for the rest of the industry. The posit number system is a tapered floating point with very efficient encodings for real numbers and two exceptional values, zero and infinity. The posit encoding leads to higher accuracy compared to floats at the same bit-width, which leads to higher performance and lower cost for big-data applications. Furthermore, the posit standard defines rules for reproducibility in concurrent environments enabling high-productivity and lower-cost for software application development for multi-core and many-core deployments.

Introduction

Modern applications in AI/Deep Learning, mobility, and IoT require vast amounts of computation but are memory limited. This includes applications in search, knowledge management, and big-data analytics, as well as foundational applications driving the digital transformation: IoT and Robotics. The first movers in AI and mobility have already built hardware accelerators that ditched IEEE Floating Point because of its inefficiency, and by doing so gained several orders of magnitude of performance over their competitors using general purpose processors from Intel, IBM, and NVIDIA. Their adoption of new arithmetic has translated into unbeatable scalability and price efficiency for next generation services for AI and Business Intelligence. In general, high-value applications can benefit from this trend of replacing floats with more efficient number systems. One such number system, posit arithmetic, invented by John Gustafson, and introduced in November 2016, has been tested by National Labs and found to beat IEEE floating point consistently in terms of accuracy, performance, memory efficiency, and performance per Watt. VividSparks has built a product portfolio of posit arithmetic solutions that customers can integrate into their products to gain a competitive advantage.

The performance problems with IEEE Floating Point stem from its inefficient encoding of the real numbers. 32-bit IEEE floats have around eight million ways to represent Not-A-Number (NaN), while 64-bit IEEE doubles have two quadrillion (2.251×10^{15}). A NaN is an exceptional value to represent the result of undefined or invalid operations, such as division by zero. It also has two representations for 0, and two representations for infinity. All these representations could be used to encode actual real number values, but these encodings are lost to represent meaningful information.

Furthermore, the IEEE floating point format defines gradual underflow, but not gradual overflow, breaking symmetry for numerical applications. The gradual underflow operations are represented by subnormal number representations, which need to follow a special execution path in the hardware because their arithmetic behavior does not conform to 'normal' numbers. Floats underflow to zero, and overflow to infinity.

Underflowing to zero destroys sign information leading to complications for algorithms that depend on this information. Many foundational algorithms in AI are defined in terms of gradient descent where the derivative of the objective function is used to minimize that objective function. When the derivative equals 0, it provides no information about which direction to move to continue the gradient descent optimization.

Such points are called critical, or stationary. Underflowing to zero is a numerical error that creates critical points where there are none. Therefore, all libraries for AI require special code to remove this IEEE floating point behavior to produce correct answers.

Whereas underflowing to zero creates errors in the interpretation of crucial functions, overflowing to infinity increases relative error by an infinite factor. When this overflow occurs, subsequent arithmetic operations will fail, again destroying valuable information. The lack of symmetry between underflow and overflow creates a very inefficient encoding, as all these overflow encodings are interpreted as Not-A-Number quantified above.

But maybe the most egregious failure of the IEEE encoding is the impractical relationship between precision and dynamic range. The dynamic range of 64-bit doubles, the workhorse of most numerical codes because of its 53-bit precision, is 2^{2047}; it is estimated that there are between 10^{78} to 10^{82} atoms in the known, observable universe, so 64-bit doubles overshoots that dynamic range by almost 600 orders of magnitude. All this encoding inefficiency, asymmetry, and special handling leads to very complicated and power-inefficient arithmetic circuitry and data flows.

The arithmetic problems with IEEE Floating Point arise from uncontrolled rounding. In the IEEE standard, there are no rules defined for how to compute simple arithmetic equations such as this example:

$$x := 10^8+1 -10^8-1$$

We will readily recognize that the value of x is zero, but an Intel or NVIDIA processor may answer −1, 0, or 1, and that is ok by the IEEE standard committee. This is an example of IEEE floats failing the associative property of addition. They also fail the associative property for multiplication, and the distributive property for addition and multiplication. In multi-core and many-core concurrent environments, when the computer hardware schedules the order of operations, this loss of associativity leads to numerical errors and irreproducibility of results. For example, when writing control software to drive a valve in the cooling core of a nuclear reactor, extraordinary care must be taken to be able to guarantee reproducibility and work around these limitations of IEEE Floating Point.

The IEEE standard also defines a multitude of rounding modes, from round-to-nearest (two forms: ties-round-to-nearest-even, and ties-round-away-from-zero), round-down and round-up, to round-towards-zero (= truncation). These alternative rounding modes are useful in diagnostic numerical instability but add a tremendous amount of hardware complexity for the uncommon use case.

Examples of Measured Posit Benefits

High-Performance Computing, Big Data, and Predictive Analytics

The most thorough study to date to quantify the benefits of posits over IEEE Floating Point has been produced by numerical analysts Peter Lindstrom, Scott Lloyd, and Jeffrey Hittinger at Lawrence Livermore National Laboratory. They have reported on an experiment to compare the new posit system against traditional IEEE floating point [7] and found the posit number system to outperform IEEE floating point across the board. The experiment modified a numerical simulation application, Euler2D, which implements an explicit, high-resolution Godunov algorithm to solve the Euler system of equations for compressible gas dynamics on an L-shaped domain. Such a solver is simple enough to instrument and comprehend while providing sufficient complexity in the numerical behavior of the solution, e.g. a nonlinear hyperbolic system with shock formations and minimal dissipation.

The problem solved in the Euler2D code is the propagation of a shock wave in air through an L-shaped conduit. The domain is the union of two rectangles: [(0,3), (2,4)] U [(1,0), (2,3)]. At the initial time, a shock, moving with dimensionless speed M_s = 2.5 relative to the quiescent state of (ρ, u_x, u_y, p) = (1,0,0,1), is positioned at x= 0.5. The inlet flow at x = 0 is constant. The code is run with uniform mesh of size h = 1 ⁄ n= 1 ⁄256 using a fixed time step of $\Delta t \approx 2.8 e^{-4}$, resulting in roughly 1.3 trillion floating-point operations over the entire run. The system dynamics are shown in Figure 1:

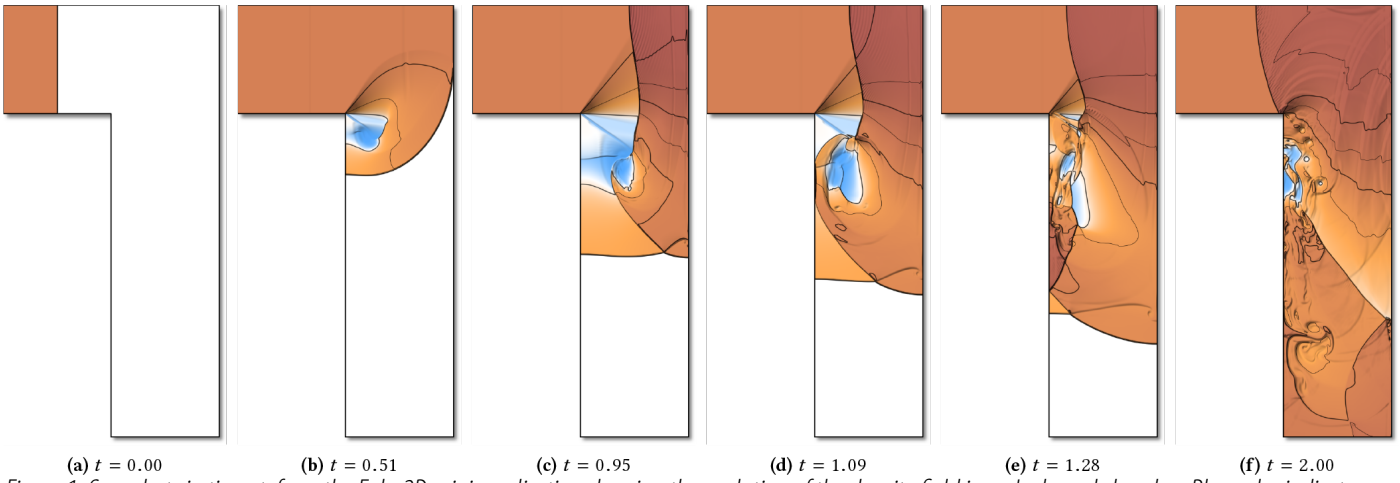


Figure 1: Snapshots in time, t, from the Euler2D mini-application showing the evolution of the density field in an L-shaped chamber. Blue color indicates density lower than the initial density (red) of the shock wave. (a) Initial state. (b) Shock reflects off of the far wall. (c) Reflected shock hits vortex. (d) Shock reflects off of near wall. (e) Second reflection hits vortex. (f) Final state.

As shown in Figure 1, the shock propagates into the chamber and diffracts around the corner, initiating the shedding of a vortex from the corner. At time t ≈ 0.51, the initial shock reflects off the far wall, and the reflected shock propagates back upstream, encountering the vortex around time t ≈ 0.95. The reflected shock breaks up the vortices shedding off the corner and reflects again off the near wall at several times. Eventually, the flow moves down the channel with a propagating sequence of oblique shock waves and a great deal of wave-wave interactions.

A pointwise, closed form solution to the Euler 2D hyperbolic PDE does not exist. To establish ground truth, the LLNL team used a quadruple precision floating point type to compute a high-precision solution. The team then computed the root mean square pointwise error in the density field to establish solution accuracy. The team reported that the RMS error was expected to be dominated by round-off error associated with each numerical type due to fixed discretization parameters, i.e., fixed truncation error. Plots of the pointwise error in the density field over time for 32-bit and 64-bit representations relative to the quadruple precision are shown in Figure 2 and Figure 3. We see spikes in error that correlate with events such as shock-wall and shock-vortex impact. These spikes are more pronounced in the 64-bit plot because of the additional precision provided in 64-bit arithmetic.

IEEE floating point and related types do quite poorly in relation to posits and other tapered precision numerical types, most evident in the 64-bit precision plot, where posit<64,2> outperforms IEEE double precision by nearly three orders of magnitude.

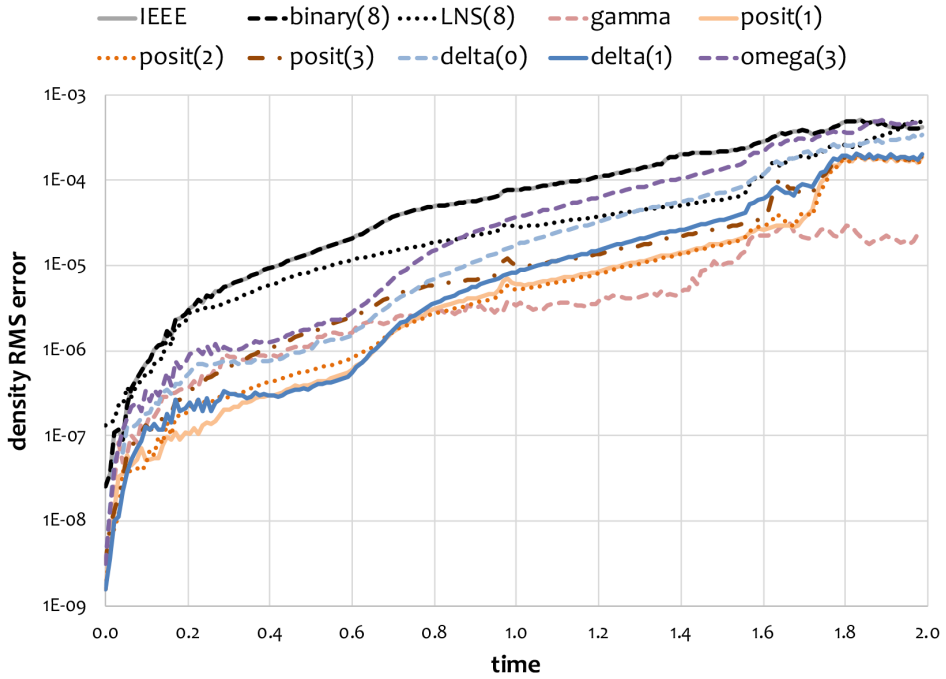


Figure 2: RMS error in Euler2D density field as a function of simulation time and 32-bit number representation.

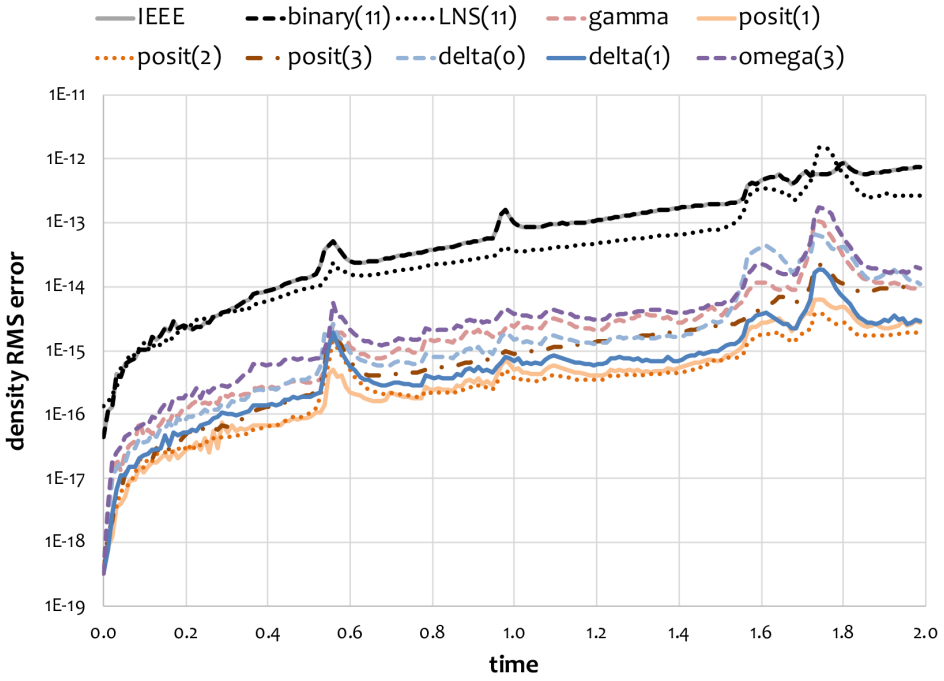


Figure 3: RMS error in Euler2D density field as a function of simulation time and 64-bit number representation.

Conclusion

With Dennard scaling and Moore's Law having run their course, performance improvements required to solve the computational demands of modern AI/Deep Learning, Big Data, and IoT analytics applications must come from innovations that improve the efficiency of computation. Replacing IEEE Floating Point with more efficient alternatives like posit arithmetic provides a key opportunity to improve our computational infrastructure. Leading innovators such as Google, Microsoft, Facebook, and China's Huawei, Alibaba, and Baidu, have already embarked on these initiatives and have gained significant benefits from these investments. Posit arithmetic provides a mechanism to consolidate the benefits of a standardized number system like IEEE Floating Point across the next generation application domains of AI/Deep Learning, Big Data/Deep Analytics, IoT, and Robotics (including autonomous vehicles).

Acknowledgment

We would like to thank Dr. Theodore Omtzigt , CEO and Founder of Stillwater Supercomputing, Inc., USA, assisting us preparing in white paper.

References

[1] David Goldberg, "What Every Scientist Should Know about Floating-Point Arithmetic," Computing Surveys, March 1991, Association for Computing Machinery, Inc. DOI: doi:10.1145/103162.103163.

[2] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic," IEEE Std. 754-2008. DOI:10.1109/IEEESTD.2008.4610935

[3] Kulisch and Miranker, A New Approach to Scientific Computation, Academic Press, New York, 1983

[4] John L. Gustafson, The End of Error: Unum Computing. CRC Press, 2015

[5] John L. Gustafson, "Beyond Floating Point: Next Generation Computer Arithmetic," Stanford University Seminar: https://www.youtube.com/watch?v=aP0Y1uAA-2Y

[6] John L. Gustafson, "A Radical Approach to Computation with Real Numbers," Supercomputing Frontiers and Innovations, Vol. 3, No. 2, 2016. DOI:https://doi.org/10.14529/jsfi160203.

[7] Peter Lindstrom, Scott Lloyd, Jeffrey Hittinger, "Universal Coding of the Reals: Alternatives to IEEE Floating Point," CoNGA 2018, March 28, 2018, Singapore. Association for Computing Machinery. ACM ISBN 978-1-4503-6414-0/18/03. DOI:https://doi. org/10.1145/3190339.3190344

[8] Intel Product briefs for Core/Core2, Nehalem/Westmere, Sandy Bridge/Ivy Bridge processors, www.intel.com

[9] Texas Instruments DSP Product briefs for TMS320C64x and TMS320C66x digital signal processors, www.ti.com

[10] NVIDIA GPU and GPGPU Product briefs for Tesla, Fermi, Kepler, and Volta GPGPUs, www.nvidia.com

VividSparks IT Solutions
#38 BSK Layout, Hubli-580031, India.
https://vividsparks.tech
info@vividsparks.tech