VPIC Design for Modern Computer Architectures

W. D. Nystrom
Los Alamos National Laboratory

VPIC [1-3] has historically been developed by a small team of developers on an intermittent basis over a period of several years in a loosely integrated fashion. The original version of VPIC was designed and implemented by a single developer. The first significant team effort was a port to the LANL Roadrunner machine which was a dramatically new and challenging architecture at the time that it was deployed. Recent efforts have been taken to port and optimize VPIC for the LLNL Sequoia platform and the new LANL Trinity platform. An effort is currently underway to open-source VPIC to facilitate future development and university collaborations.

VPIC is a three-dimensional, relativistic, electromagnetic particle-in-cell code which uses an explicit time integration scheme to solve a variety of plasma physics problems on a structured mesh. VPIC is currently a mixture of about 40,000 lines of C and C++ code. VPIC currently exploits parallelism in three distinct ways. First, there is a distributed memory strategy which uses asynchronous MPI calls to hide inter-node communication latencies. MPI can be used at the node level, core level or hardware thread level. Second, there is a thread level implemented with Pthreads which will allow use of threads on a node at either the core level or hardware thread level. Finally, there is a vectorization level which is implemented as a light weight vector wrapper class which can use either a portable implementation or a platform specific hardware intrinsic implementation. VPIC is specially designed to use single precision floating point calculations in order to optimize use of the available memory bandwidth.

VPIC has been used extensively on petaflop scale systems such as LANL's Cielo and Roadrunner, ORNL's Titan and Kraken and NSF's BlueWaters. Simulations are routinely performed at a significant percentage of the full system capabilities of these machines.
VPIC is used to perform simulations of astrophysical plasmas, laser-plasma interaction for ICF plasmas, relativistic laser-plasma interaction for laser-based accelerators, and first-principles studies of the mixing of dense plasma. At this time, VPIC only uses low level libraries such as MPI, Pthreads and vendor specific vector hardware intrinsics.

VPIC was originally designed to run efficiently on modern cache-based processors. An efficient particle-in-cell code needs to make efficient use of the memory bandwidth available on a node. To do this, special design decisions were made to allow VPIC to use single precision floating point arithmetic. This included designing interpolation routines to use a logical integer coordinate to identify the cell containing the particle and then to only use floating point arithmetic to compute the coordinates of the particle relative to a local cell coordinate system. The particle-in-cell algorithm in its most basic description can be divided into four steps: push particles to the next time level using electromagnetic field values at the individual particle coordinates, accumulate charge and current densities from the particle coordinates to the grid coordinates using an interpolation scheme, compute the

electromagnetic field values at the next required time level using the charge and current densities and finally, interpolate the new electromagnetic field values to the individual particle coordinates using an interpolation scheme. An implementation in this high level conceptual approach requires three separate loops over the particles. By using special data structures and storing more data quantities per particle, it is possible to implement the particle-in-cell algorithm using only a single loop over particles. The result is a complex and long loop over the particles that can be challenging for compilers to vectorize. Part of the complexity is due to conditional logic to handle boundary conditions and account for particles which migrate from one cell to another and to different processor domains. VPIC assumes particles do not migrate to different cells or processor domains and marks the ones that do for a subsequent post-processing loop. With properly chosen simulation parameters, the number of particles which need to be subsequently post-processed will be a small fraction of the total number of particles and will not significantly increase the time to complete a step of the algorithm. Thus, VPIC is designed to process batches of particles with the particle data located contiguously in memory and this results in efficiently streaming the particle data from memory and maximizing the number of flops performed for each memory load/store. To facilitate vectorization, the various operations required in a loop over the particles are implemented using a class of C++ vector primitives that themselves are implemented as efficiently as possible using the vector hardware intrinsics available for the desired platform.

VPIC uses several abstractions. As a particle-in-cell code, two main abstractions are particles and fields. VPIC also uses a work pipeline algorithm in which independent chunks of work on either particles or fields is assigned to execution threads which may be either Pthreads or MPI ranks. These independent chunks of work are designed to optimize the ratio of flops to load/stores from memory. Finally, VPIC uses a vector class which has a portable implementation and can be implemented using vector hardware intrinsics for a target platform. The vector class generally gives about a 2x speedup over the portable implementation and so far has been shown to be a good way to get portable performance across different machines. The vector class has been successfully applied to the computationally intensive parts of VPIC. VPIC has not been ported to run on GPU hardware, however, so it remains to be seen whether the current VPIC design will perform and scale well on GPU based platforms such as Titan and the upcoming CORAL GPU platforms. So far, the current VPIC design has achieved significant fractions of peak performance on a number of different architectures.

[1] Bowers, K. J., B. J. Albright, L. Yin et al., *Phys. Plasmas*, **15**, 055703 (2008)
[2] K. J. Bowers, B. J. Albright, B. Bergen et al., Proceedings of the ACM/IEEE conference on Supercomputing (IEEE, New York, 2008), Austin, pp. 1-11, 2008 (Gordon Bell Prize Finalist)
[3] K.J. Bowers, B.J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, T.J.T. Kwan, Journal of Physics: Conference Series **180** (1), 012055 (2009).