LA-UR-16-28383
Approved for public release; distribution is unlimited.

Title: Improving the Performance and Portability of VPIC

Author(s): Bird, Robert Francis
Nystrom, William David
Albright, Brian James
Peters, Evan

Intended for: APS DPP, 2016-11-01/2016-11-04 (San Jose, California, United States)

Issued: 2016-11-01

Improving the Performance and Portability of VPIC

R. F. Bird, **E. A. Peters, W. D. Nystrom, and B. J. Albright**
Los Alamos National Laboratory, Los Alamos, NM

Contact: bird@lanl.gov

November, 2016

Introduction:

VPIC

- VPIC is a single-precision 3D relativistic, electromagnetic particle-in-cell (PIC) plasma simulation code which has demonstrated world class performance
- VPIC expresses parallelism at three levels:
  - MPI;
  - Phreads;
  - Vector intrinsics;
- Excellent performance, reduced portability (and readability!)

Goals of this work

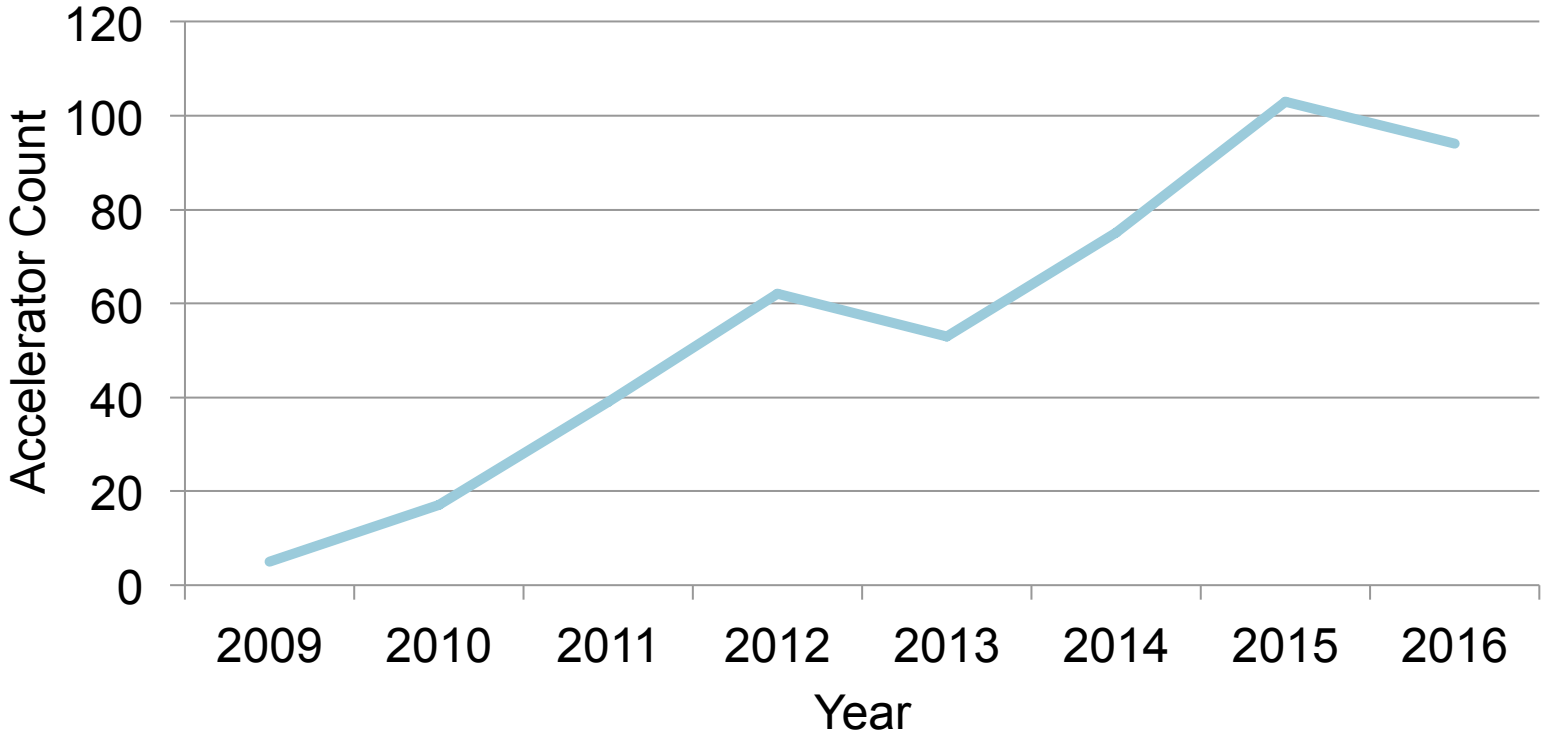
- *Investigate* techniques we can apply to VPIC in order to improve it's portability, readability, and usability
- *Evaluate* the effectiveness of such techniques, and assess their suitability for the main VPIC code branch
- *Understand* the wider implications of such codes changes, and how they can affect the overall code landscape as we move towards exascale

Heterogeneity

- Heterogeneous platforms are becoming increasingly common in the modern HPC landscape
  - Figure 1 shows the significant increase in accelerator based supercomputers in the TOP500. With the majority of the top 10 relying on non-traditional architectures
- This diversity offers increased computational performance, at the cost of programmability
  - Codes often need to be re-written to use the new hardware
  - Unclear if a good, single source, solution exists. Attempts such as OMP4.5 and OpenACC try and bridge this gap
- Cost of maintaining code scales with both it's complexity and also it's variability

Heterogeneity

Figure 1: Accelerator Count by Year for machines in the TOP500



Code Portability

- Many codes outlive the machines they will be run on, and have been around for decades (legacy codes)
- It is desirable for such codes to be portable -- able to effectively utilize a variety of hardware
  - Often this means be able to make use of: CPUs, GPUs, and Co-processors (Intel Xeon Phi)
- This can be achieved through a single-source solution (more desirable), or through the development of a code version for each hardware type (less desirable)

Code Performance

- It is not sufficient for codes to be portable, they also need to offer good performance
- Often these ideas are combined, referred to as *portable performance*
  - This is very hard to define concisely!
- Typically people say a code offers portable performance when it can:
  - Run on the required hardware
  - Achieve a reasonable level (“good enough”) percent of peak performance available for that hardware

Implementation Details:

VPIC: Performance Portability

- The current release of VPIC is high performance, but only for traditional architectures (CPUs)
  - See the work by Dave Nystrom (JP10) for an excellent summary of the cutting edge performance work done with VPIC, entitled: “Performance of VPIC on Trinity”
- VPIC’s performance portability is limited by its reliance on vector intrinsics and the explicit use of pthreads.
  - Current versions of VPIC need code changes to map to hardware which has an increased SIMD width, and the current code has no clear path forward to GPU acceleration

VPIC: Performance Portability

- This work incrementally improves the performance portability of VPIC by:
  - Replacing the explicit use of pthreads with OpenMP directives
  - Developing an auto-vectorising version of the main kernel
  - Investigating further code optimizations made possible by having a more malleable kernel

VPIC: OpenMP

- By enabling the use of OpenMP, we obtain the following benefits:
  - Programming at a higher level of abstraction
  - Better compatibility with many major tool suites
  - Ease of controlling thread placement through a standardized interface
  - Ability to exploit extensions to schedulers with regard to job placement
  - Use of high level constructs such as reductions
  - Finer grain SIMD control through #pragma omp simd

VPIC: OpenMP Implementation

- The process to replace pthreads with OpenMP was fairly simple to do at a high level of abstraction
- Previously calls to pthreads were dispatched by the use of a function pointer
- This function call can then instead be wrapped by an OpenMP parallel region
- Future efforts will work to convert this to a more typically loop-level invocation of OpenMP

VPIC: Auto-vectoring

- By enabling the use of an auto-vectorized kernel we obtain the following benefits:
  - Significantly more readable and maintainable code
  - Better expose code-intent to the compiler, potentially enabling more aggressive optimization
  - Automatic scaling to increased SIMD width

- By simplifying the code, it is more accessible to new team members and non-experts. Future code changes should be quicker to implement and less prone to human errors

VPIC: Auto-vectorizing Implementation

- Before the compiler could auto-vectorize and standard C expression of the kernel, the following changes needed to be made:
  - Remove any pointer aliasing (through the restrict keyword and the use of #pragma ivdep)
  - Performing loop fission to try isolate code which is safe to do in SIMD
  - Serialize code sections which are not SIMD safe
  - Enable reductions and other access marshaling for shared SIMD variables

Results:

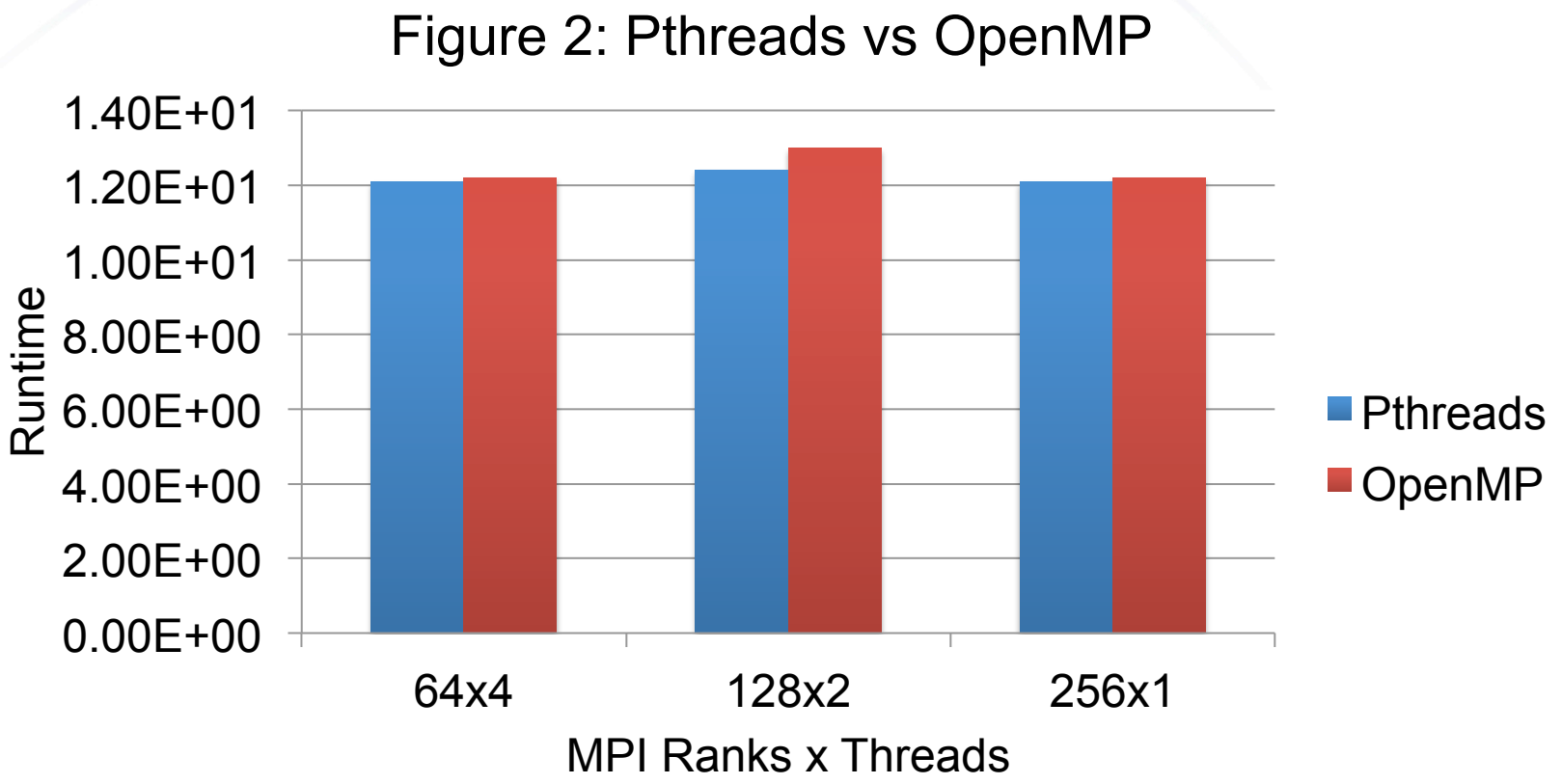
Problem Details

- Simulation of a Laser-Plasma Interaction (LPI) with 100 million particles, run for 12 electron plasma wavelengths during 76 time-steps
- As is typical for LPI simulations, this problem is particle-push dominated, accounting for around 80% of the runtime
- Supports 1-3D domain decompositions

Runtime System

- Trinitite, a Cray XC40 production development platform for Trinity
  - Features the latest generation of Intel Xeon Phi Co-processor, the Knights Landing (KNL)
  - Intel Xeon Phi Knights Landing processor which can be configured at run time into 20 different modes
  - Features on-package High Bandwidth Memory (HBM) for KNL and burst buffer technology to augment performance of I/O
- KNL results are gathered for the quad-flat memory configuration

Threading Model Results

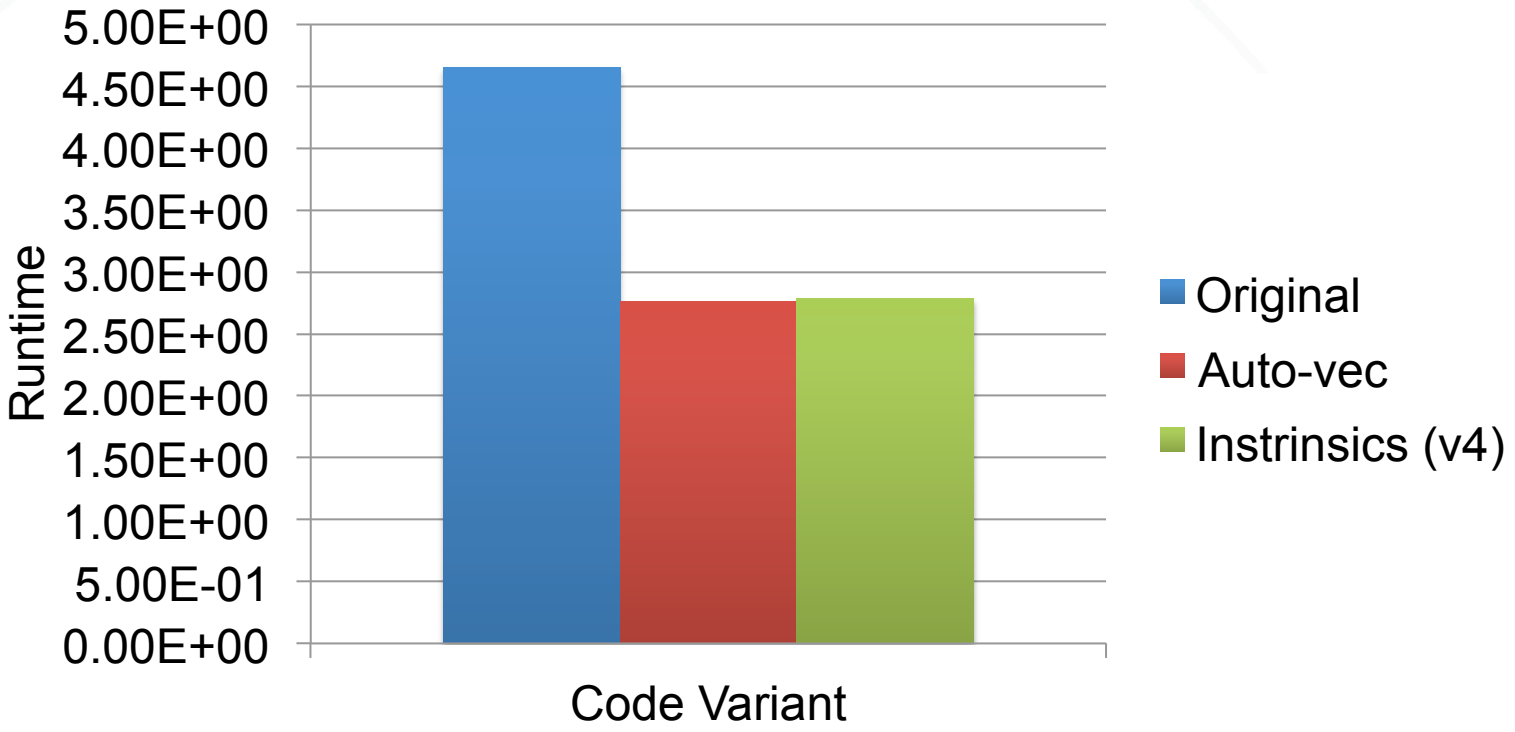


Threading Model Results

- As shown in Figure 2, the code was ported successfully to OpenMP with comparable performance
  - The slight variations in time are small enough that they do not represent a significant difference, with OpenMP also being marginally faster for some test cases
- The comparable performance and additional usability with many modern tool chains easily justifies the effort in making the conversion

Auto-vectorization Results

Figure 3: Auto-vectorization Results for KNL



Auto-Vectorization Results

- Figure 3 demonstrates that the auto-vectorized kernel demonstrates comparable performance compared to the release version on KNL, and a marked improvement over the previous non-intrinsics version
  - This is also true for traditional CPU architectures
  - Recent advances in the VPIC code base have further improved the intrinsics code version meaning further analysis is required
- The majority of this speedup is due to the vectorization enabled by the loop fission and dependency removal

Conclusion

- Overall this effort represents a significant success in enhancing the portable performance of VPIC.
- A code variant of comparable performance with significantly increased readability, portability, and usability has been developed
- This new code variant will be one of the release candidates for use during the Trinity Open Science Campaign
  - This work will be integrated into a future release of the main VPIC code branch, and release for use by the wider community

On-going and Future work

- Many further enhancements and optimization opportunities are currently being investigated, including:
  - Alternative data layouts (SoA, AoSoA)
  - Explicit strip mining for accumulation arrays
  - Per cell particle iteration
  - More aggressive loop fission
- These efforts require further validation, and will be presented in the near future

References

- VPIC Source Code: https://github.com/losalamos/vpic
- K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan, “Ultrahigh performance three-dimensional electromagnetic relativistic plasma simulation”, Physics of Plasma, vol 15, no. 5, 2008.
- K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, J. Barker, and D. J. Kerbyson, “0.374 Pflop/s Trillion-Particle Kinetic Modeling of Laser Plasma Interaction on Roadrunner”, SC 2008: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, (Piscataway, NJ, USA: IEEE Press) pp 1-11.
- K. J. Bowers, B. J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, and T. J. T. Kwan, “Advances in petascale kinetic plasma simulation with VPIC and Roadrunner”, Journal of Physics: Conference Series 180 (2009) 012055.
- TOP500: https://www.top500.org

Acknowledgements

- Work performed under the auspices of the U. S. Dept. of Energy by the Los Alamos National Security, LLC Los Alamos National Laboratory under contract DE-AC52-06NA25396 and supported by the LANL LDRD program.