| | |
|---|---|
| Title: | Addressing Performance Issues with VPIC on Trinity |
| Author(s): | Nystrom, William David; Bird, Robert Francis; Rust, William Newton III; Pang, Xiaoying; Nam, Hai Ah; Stark, David James; Daughton, William Scott; Yin, Lin; Guo, Fan; Li, Hui; Albright, Brian James; Bowers, Kevin J. |
| Intended for: | Nuclear Explosives Code Developers Conference 2016, 2016-10-17/2016-10-21 (Livermore, California, United States) |
| Issued: | 2016-10-12 |

# (U) Addressing Performance Issues with VPIC on Trinity

Nystrom, W. D., Bird, R. F., Rust, W. N. III, Pang, X., Nam, H. A., Stark, D. J., Daughton, W. S., Yin, L., Guo, F., Li, H., Bowers, K. J., Albright, B. J.
Los Alamos National Laboratory, Los Alamos, NM

October 20, 2016

## Overview of VPIC

- VPIC is an open source, 3D explicit, relativistic, charge-conserving, electromagnetic PIC code developed at Los Alamos National Laboratory.
- See https://github.com/losalamos/vpic
- VPIC uses single precision and a 3D Cartesian uniform structured mesh.
- Particle and field data structures are arranged as Array of Structures (AoS).
- VPIC uses asynchronous MPI for top level parallelism, OpenMP or Pthreads for intermediate level parallelism and vectorization via explicit use of hardware intrinsics for low level parallelism.
- VPIC was highly optimized for Roadrunner and was able to achieve 0.374 Pflops/s (single precision) overall and 0.488 Pflops/s (single precision) for main particle processing loop.
- This level of optimization and performance for VPIC allowed for significant calculations to be performed with numbers of particles and cells that had not been achievable previously.

## Overview of VPIC (cont)

- Figure 1 shows an example of a VPIC astrophysical calculation performed during the Trinity Phase 1 Open Science Campaign in February, 2016.
- More information on VPIC is available from the following references:
- 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.

## Figure 1: Trinity Phase 1 Open Science Calculation

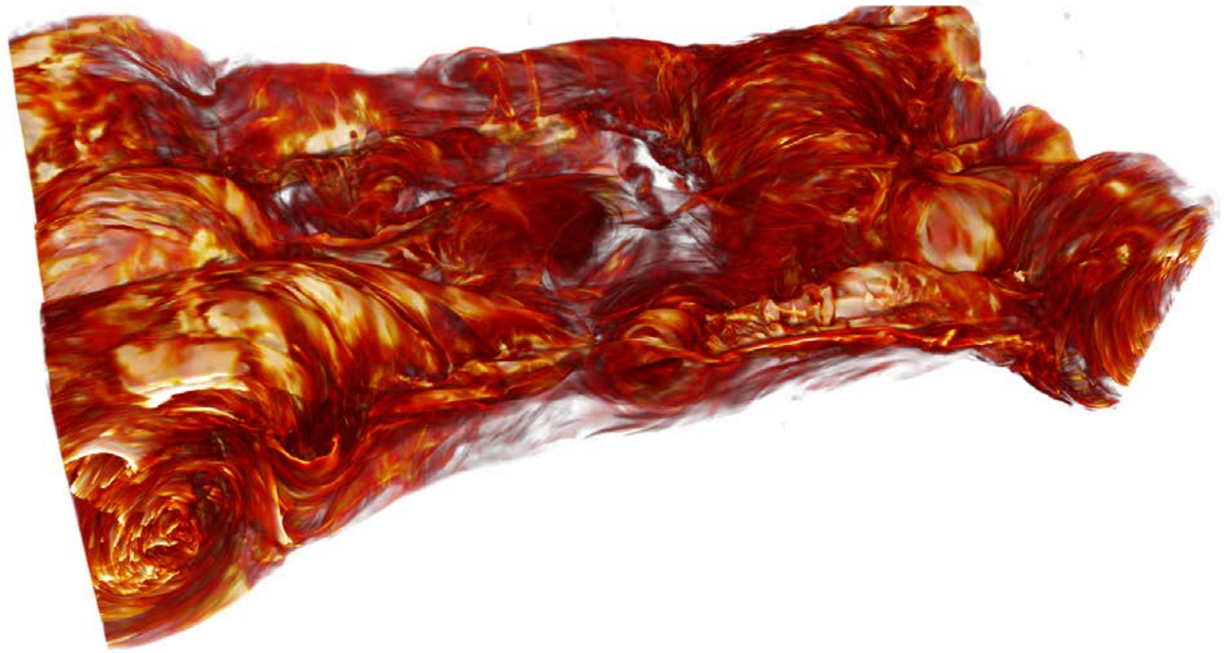


Figure 1: Volume rendering of the current density from one of the reconnection runs performed during the Trinity Phase 1 Open Science campaign.

## Overview of Trinity

- Trinity is the new capability class HPC machine at LANL that is managed by the LANL/Sandia ACES Consortium and replaces Cielo.
- Trinity has two partitions of roughly equal size, an Intel Haswell partition and an Intel Knights Landing (KNL) partition.
- Trinity Phase 1 consisting of the Haswell partition, half the burst buffers and a parallel file system was delivered and accepted in 2015.
- Trinity Phase 2 consisting of the KNL partition and the remainder of the burst buffers is currently undergoing acceptance.
- Unique and new features of Trinity include the Intel Xeon Phi Knights Landing processor which can be configured at run time into 20 different modes, on package High Bandwidth Memory (HBM) for KNL and burst buffer technology to augment performance of I/O.

## Trinity by the Numbers

| Parameter | Phase 1 | Phase 2 | Total |
|---|---|---|---|
| Nodes | 9436 Haswell | 9984 KNL | 19420 Nodes |
| Cores/Node | 32 | 68 | |
| HW Threads/Node | 64 | 272 | |
| Memory/Node | 128 GiB | 96 GiB (+16G HBM) | |
| Total Memory | 1.15 PiB | 0.91 PiB | 2.07 PiB |
| Node Peak Perf | 1.18 Tflops | 3.01 Tflops | |
| System Peak | 11.1 Pflops | 30.7 Pflops | 41.8 Pflops |
| PFS Capacity | 78 PB | Unchanged | 78 PB |
| PFS Bandwidth | ~ 0.8 TB/S | ~ 0.8 TB/S | 1.6 TB/S |
| Burst Buffer Nodes | 300 | 276 | 576 |
| BB Capacity | 1.92 PB | 1.77 PB | 3.65 PB |
| BB Bandwidth | 1.71 TB/S | 1.57 TB/S | 3.28 TB/S |

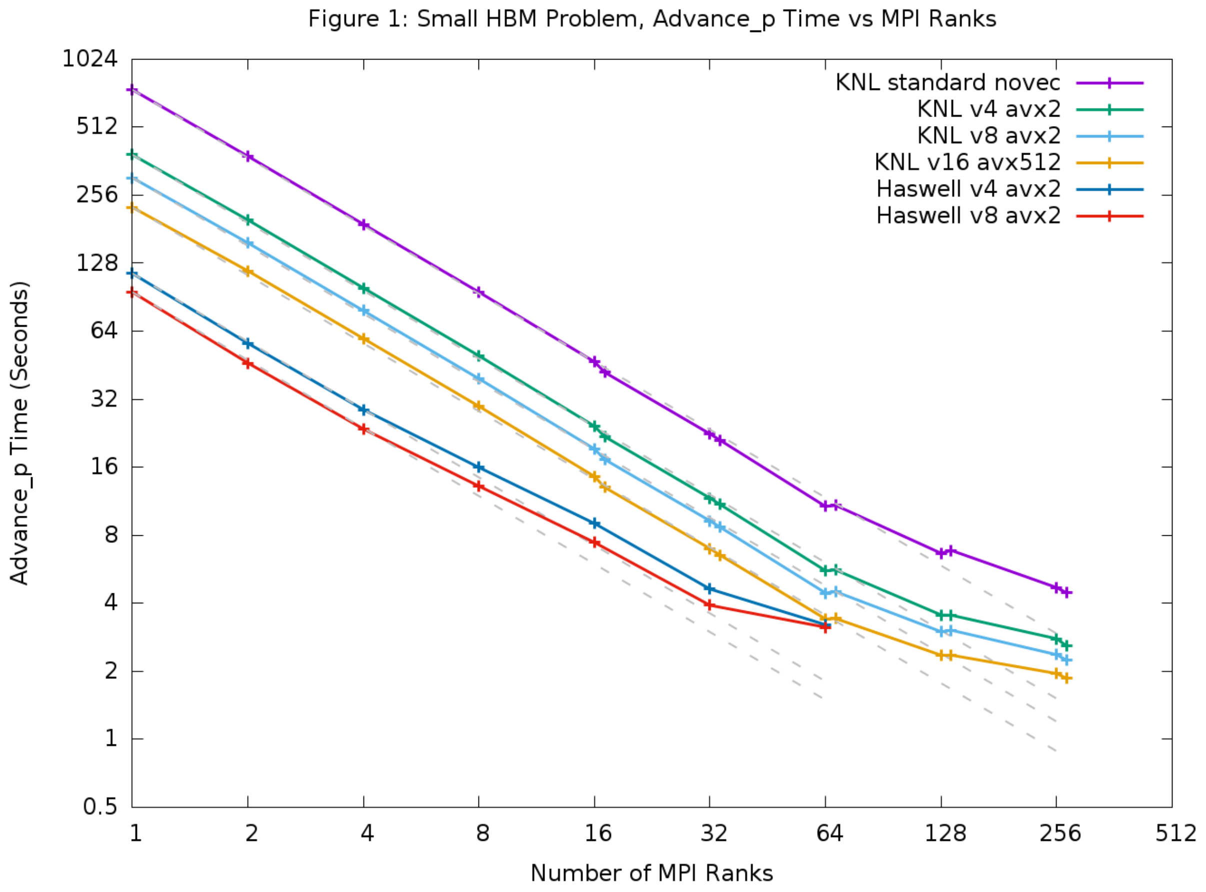
## Single Node Performance

- Primary focus of VPIC Trinity optimization is single node performance.
- Figures 2 and 3 show strong scaling results for Haswell and KNL nodes.
- MPI only is used for 2 problem sizes, one that fits in HBM and one that is too large for HBM only and uses most of DDR memory.
- 1 MPI rank/core used up to 68 ranks on KNL, 32 ranks on Haswell.
- 2 MPI ranks/core used for 128 and 136 ranks on KNL, 64 ranks on Haswell.
- 4 MPI ranks/core used for 256 and 272 ranks on KNL.
- Results shown for reference implementation that does not vectorize and hardware intrinsics versions with SIMD vector lengths of 4, 8 and 16.
- On KNL, quad flat mode is used with numactl –preferred=1 option to allow any data structures that will fit in HBM to be placed there.
- With a large number of particles/cell, many or all grid sized arrays should fit in HBM for the large DDR problem.
- VPIC scales perfectly for both problem sizes on KNL up to 64 ranks.

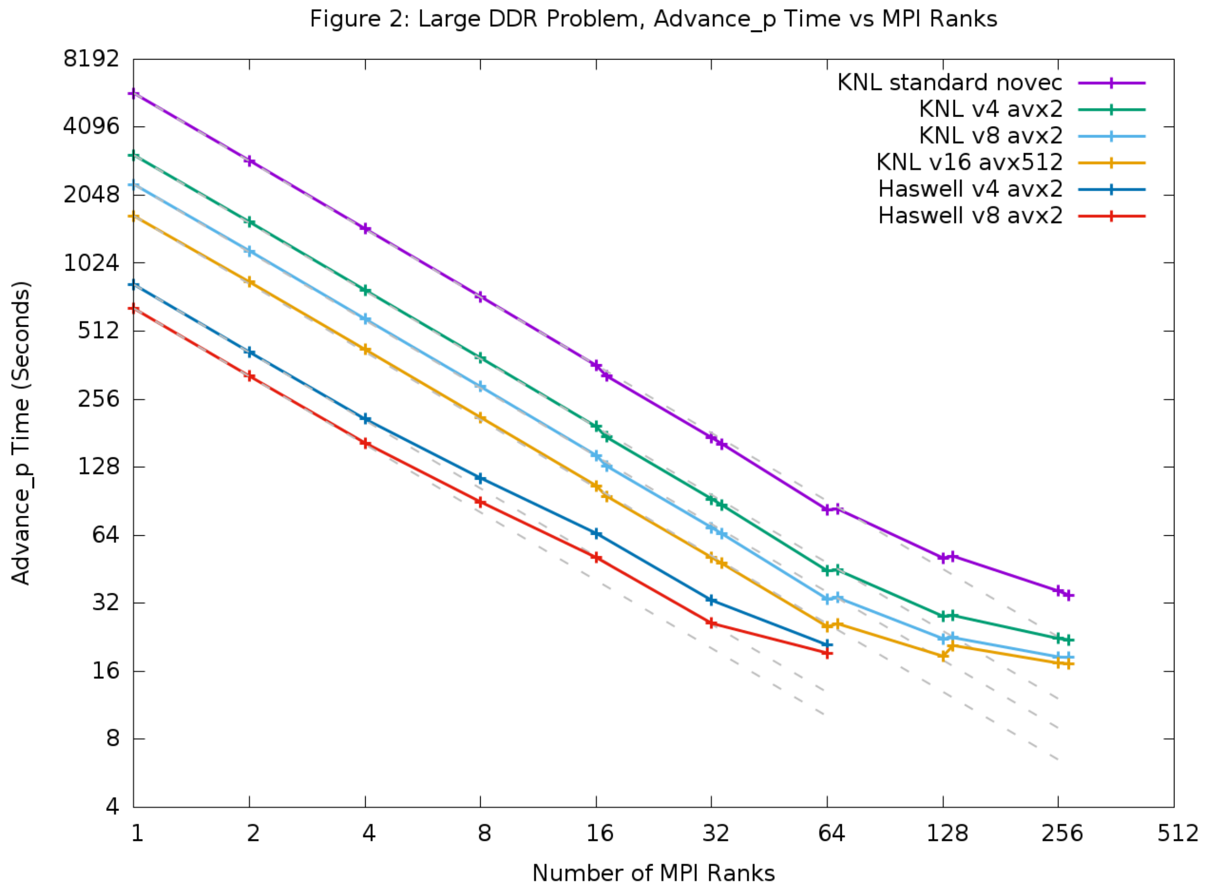
## Single Node Performance (cont)

- On KNL, for 68 ranks VPIC is competing for shared resources with the OS.
- On KNL, for 128 ranks and above, VPIC execution threads are competing for shared resources on a core such as L1 data and instruction caches.
- On KNL, scaling of SIMD vector implementations is not as good as that of the no vectorization case for 128 ranks and greater – there is more competition for shared resources because of higher computational intensity.
- On Haswell, scaling is not nearly as good as KNL up to 32 ranks.
- On Haswell, V4 with SIMD vector length of 4 scales better than V8 with SIMD vector length of 8 between 32 and 64 ranks.

# Figure 2: Small HBM Problem, Advance_p Time vs MPI Ranks



# Figure 3: Large DDR Problem, Advance_p Time vs MPI Ranks

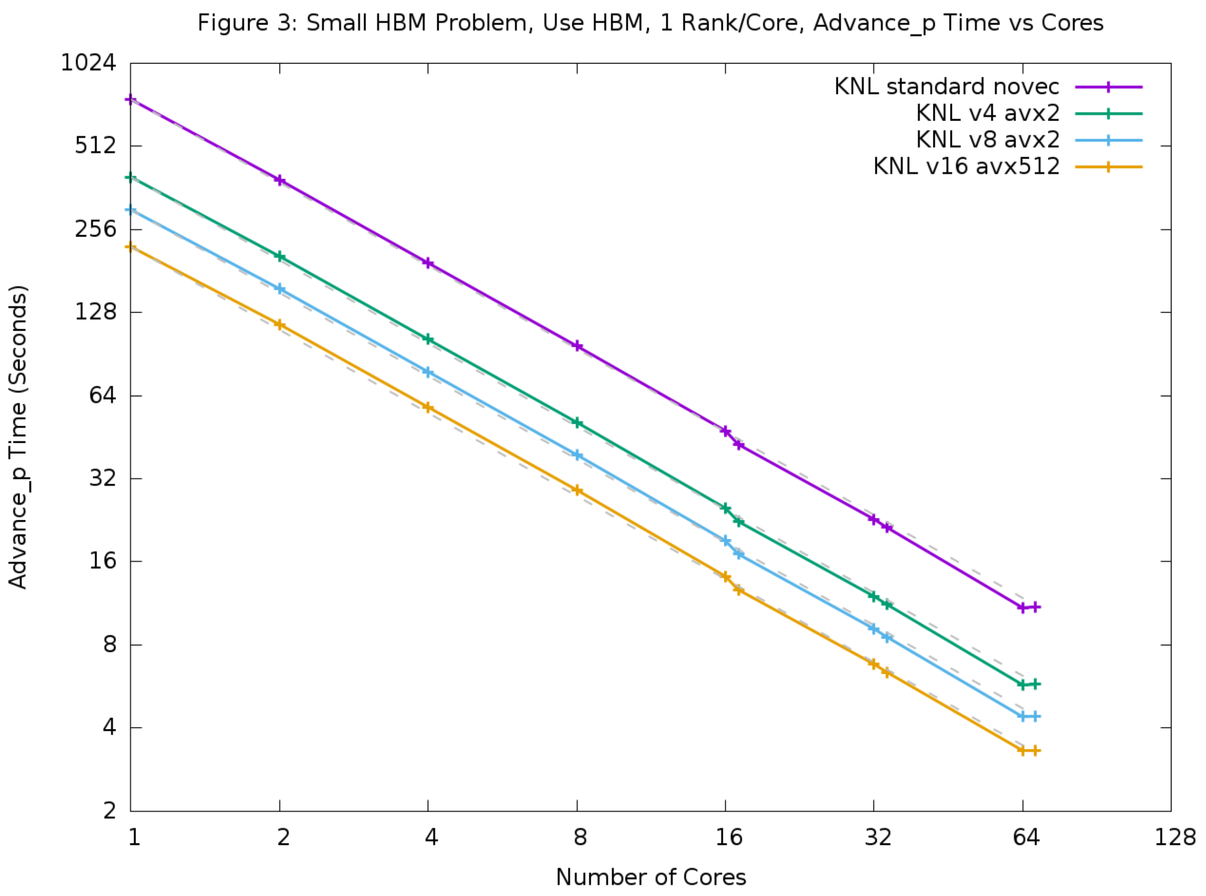


UNCLASSIFIED

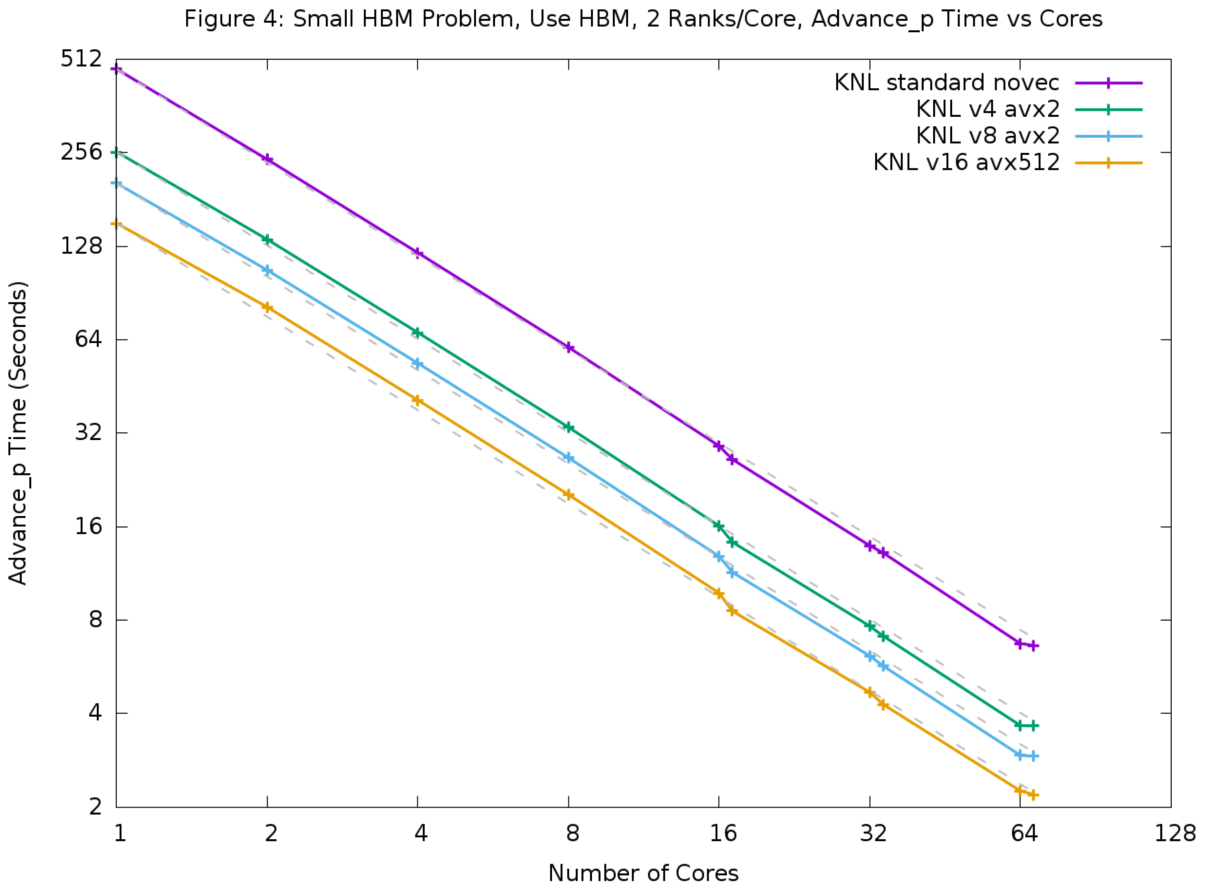
## Performance vs Ranks/Core

- On KNL, Figures 4-6 show performance versus cores for the small HBM problem when running with 1, 2 or 4 MPI ranks/core.
- On KNL, Figures 7-9 show performance versus cores for the large DDR problem when running with 1, 2 or 4 MPI ranks/core.
- On KNL, for the problem which fits in HBM, VPIC scales nearly perfectly whether using 1, 2 or 4 MPI ranks/core.
- This suggests that to improve VPIC performance on KNL for small problems running out of HBM, it is necessary to improve single thread and single core performance. Doing this is part of an Intel/LANL collaboration.
- On KNL, for the large problem that must use DDR memory, the V16 version with SIMD vector length of 16 shows degraded scaling for 2 MPI ranks/core which worsens for 4 MPI ranks/core. The V8 version starts to show degraded scaling for 4 MPI ranks/core.
- Degraded scaling for the DDR case suggests investigation of memory bandwidth.

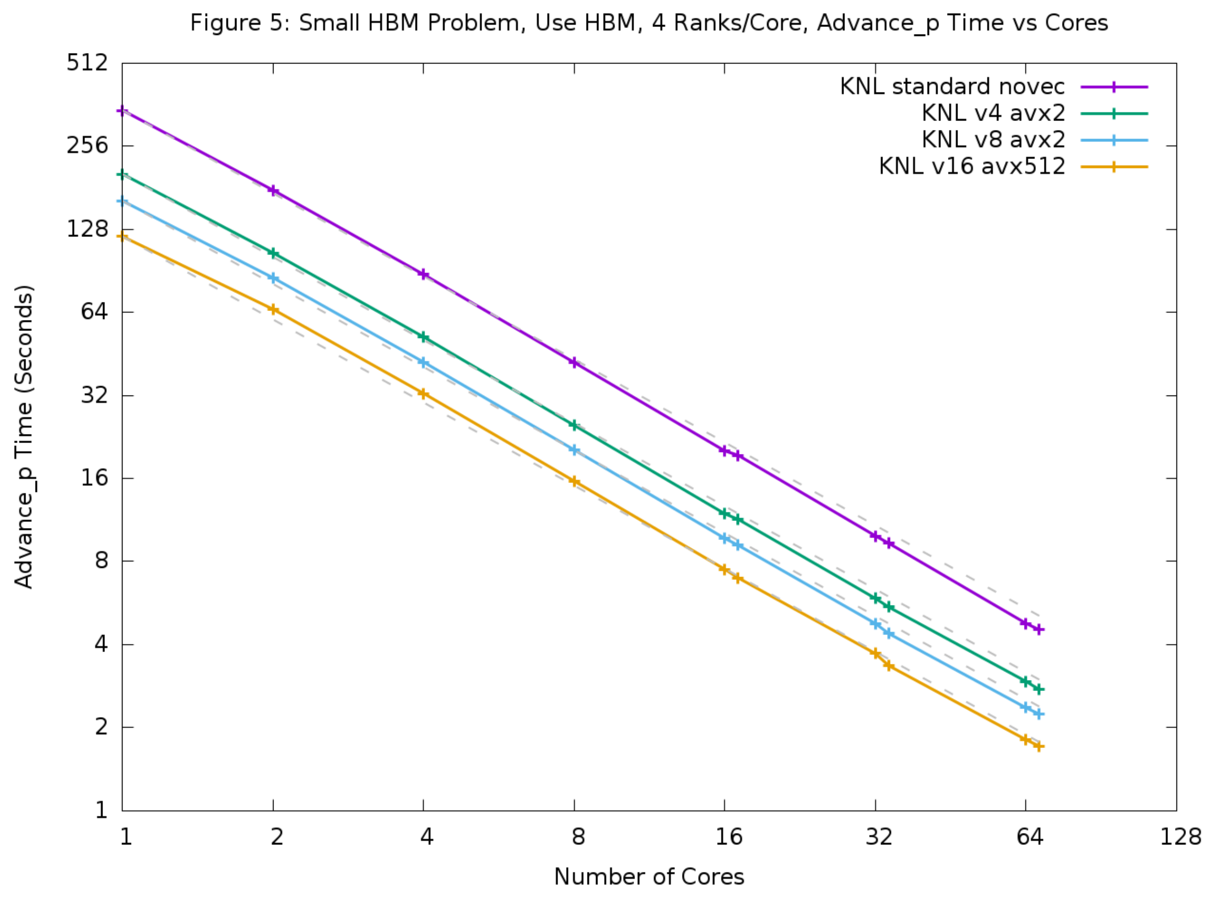
# Figure 4: Small HBM Problem, Use HBM, 1 Rank/Core, Advance_p Time vs Cores



## Figure 5: Small HBM Problem, Use HBM, 2 Ranks/Core, Advance_p Time vs Cores

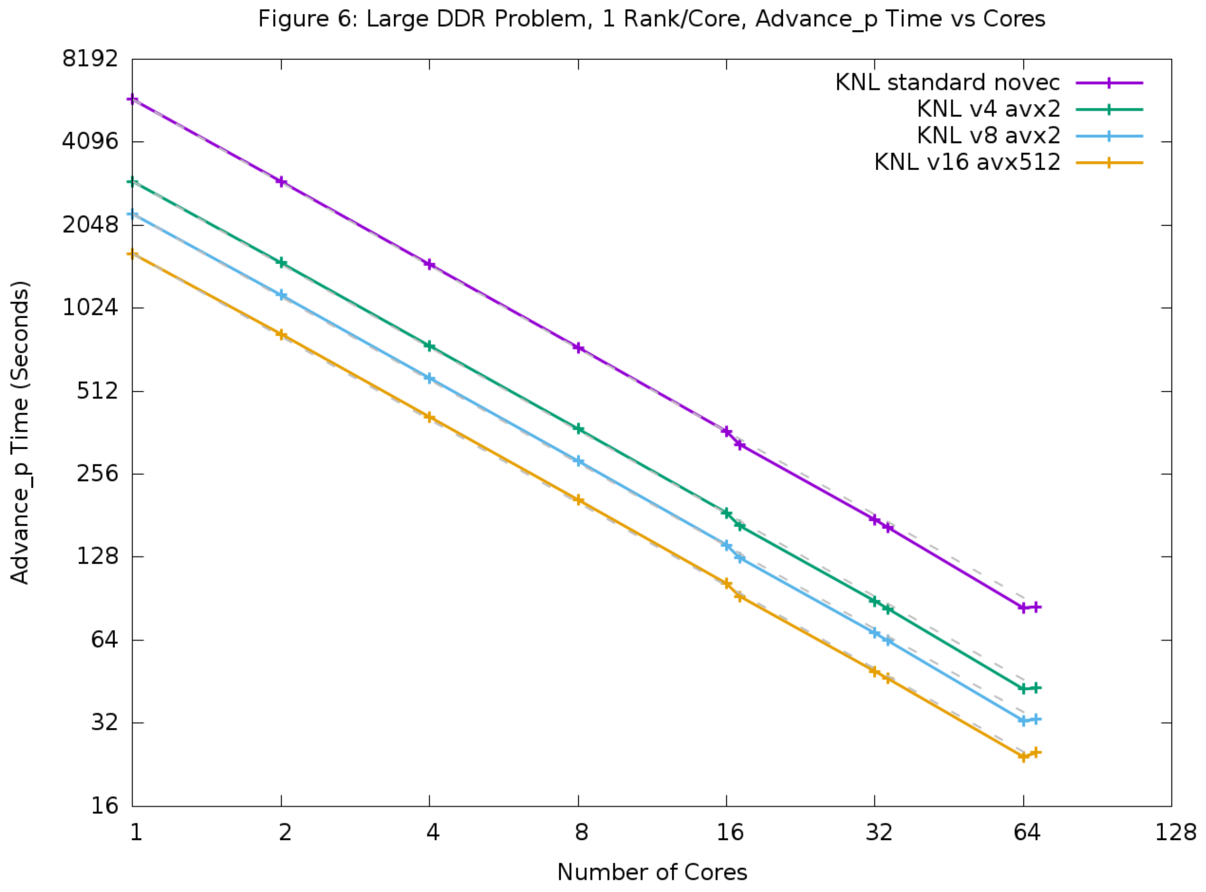


# Figure 6: Small HBM Problem, Use HBM, 4 Ranks/Core, Advance_p Time vs Cores



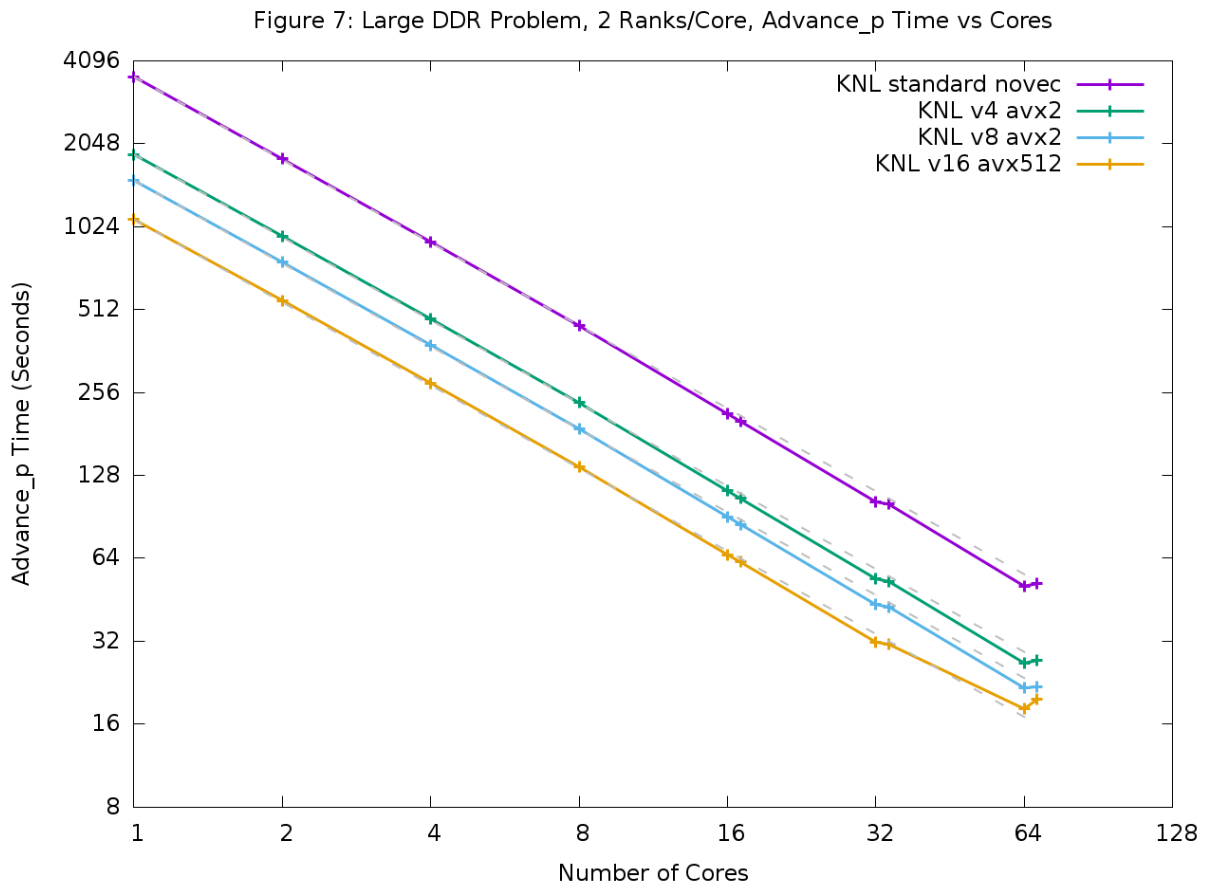
UNCLASSIFIED

# Figure 7: Large DDR Problem, 1 Rank/Core, Advance_p Time vs Cores



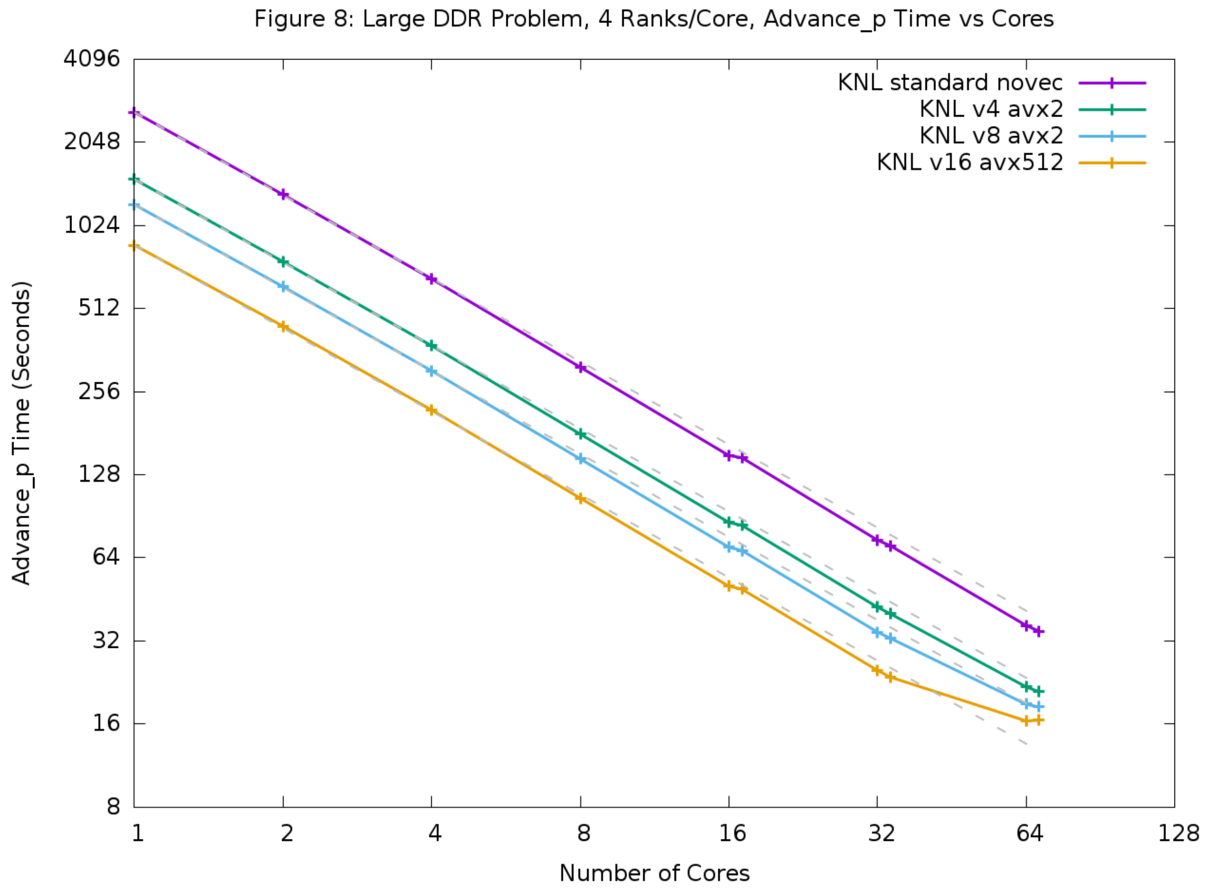
UNCLASSIFIED

# Figure 8: Large DDR Problem, 2 Ranks/Core, Advance_p Time vs Cores



UNCLASSIFIED

## Figure 9: Large DDR Problem, 4 Ranks/Core, Advance_p Time vs Cores

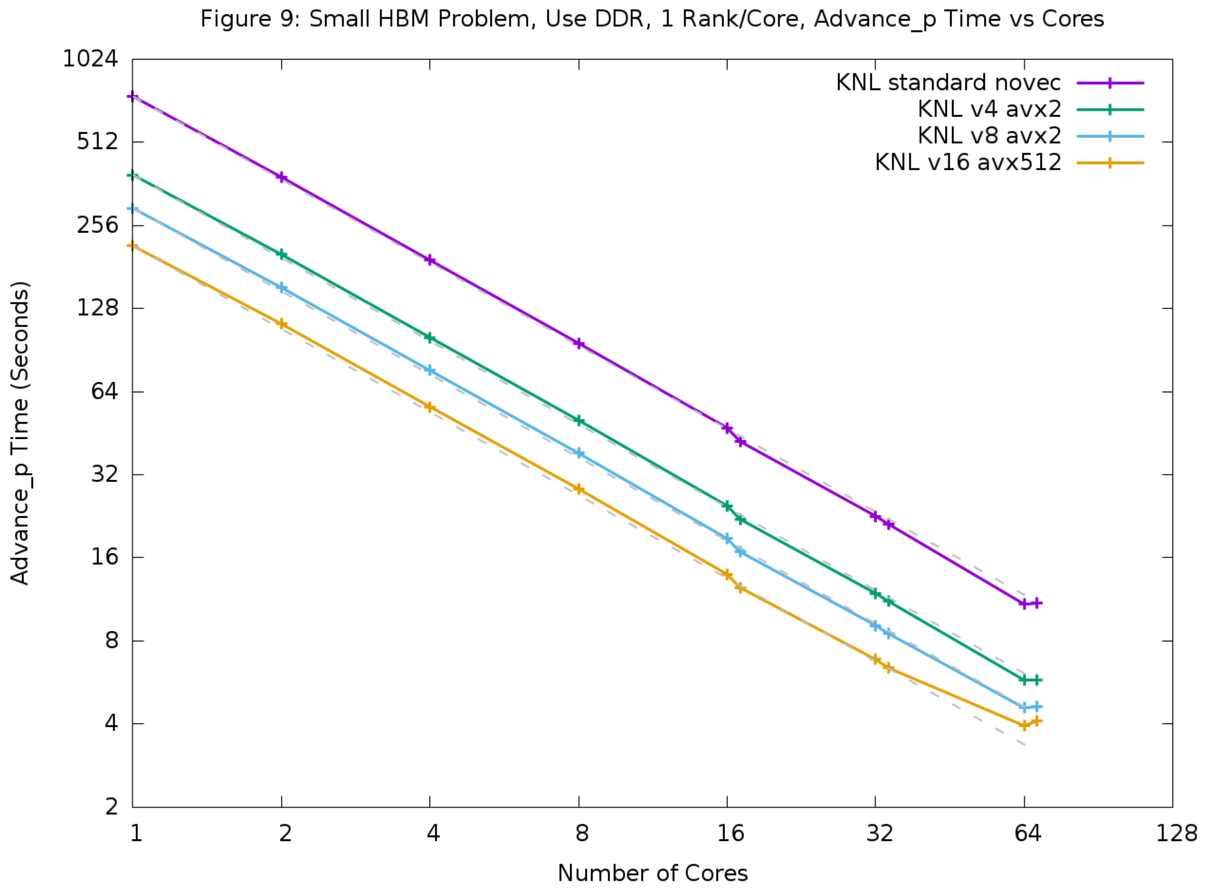


UNCLASSIFIED

## Memory Bandwidth, DDR vs HBM

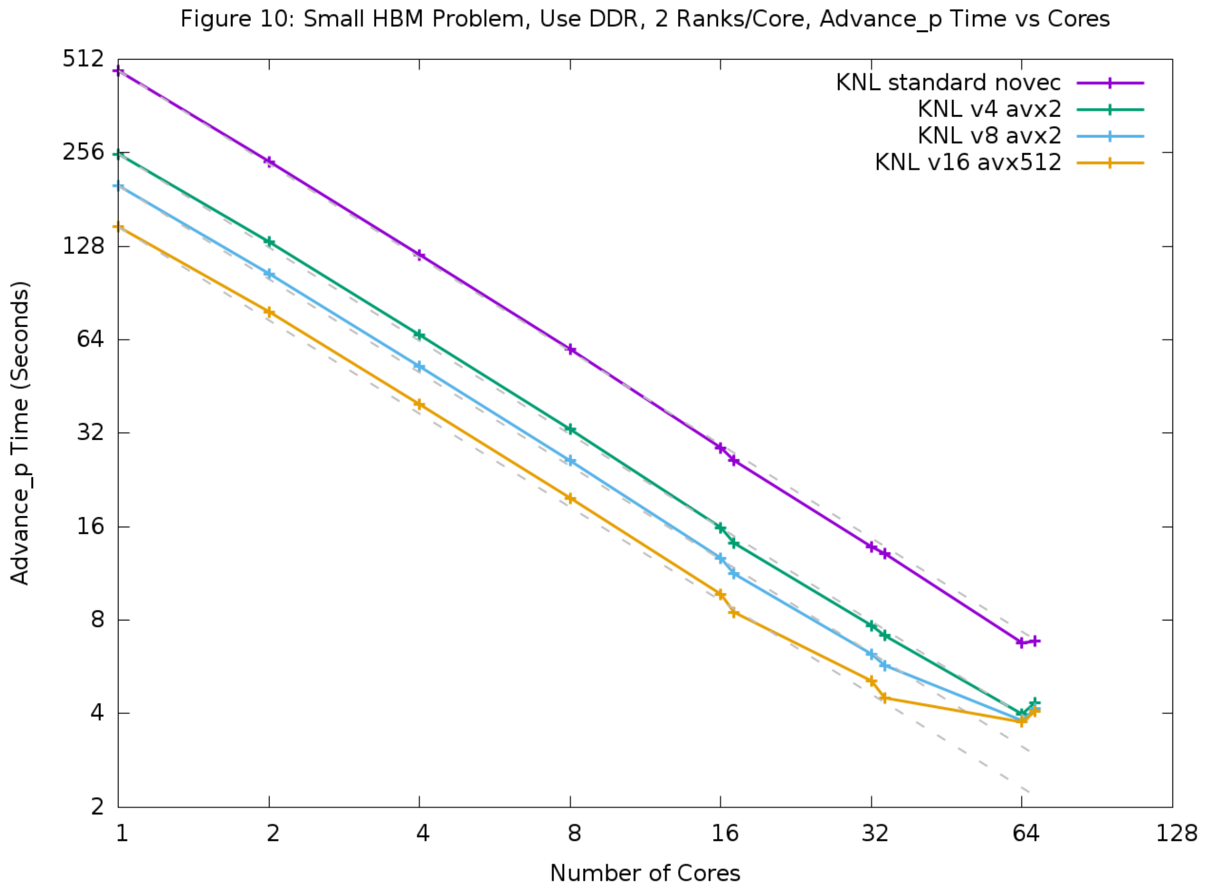
- Figures 10-12 show results of repeating runs shown in Figures 4-6 for the small HBM problem only using numactl –preferred=0 option to require using DDR memory instead of HBM memory.
- V16 version shows degraded scaling for 1 MPI rank/core which worsens for 2 MPI ranks/core and worsens still more for 4 MPI ranks/core.
- V8 version shows degraded scaling starting with 2 MPI ranks/core which worsens for 4 MPI ranks/core.
- V4 version shows hint of degraded scaling starting with 2 MPI ranks/core which worsens for 4 MPI ranks/core.
- These results show the V4, V8 and V16 versions are all memory bandwidth limited when running purely out of DDR.
- There is no indication of being memory bandwidth limited when running purely out of HBM which has ~5X the memory bandwidth of DDR.
- Improving the use of HBM for large DDR problem types seems key to improving performance for the typical VPIC use case.

# Figure 10: Small HBM Problem, Use DDR, 1 Rank/Core, Advance_p Time vs Cores



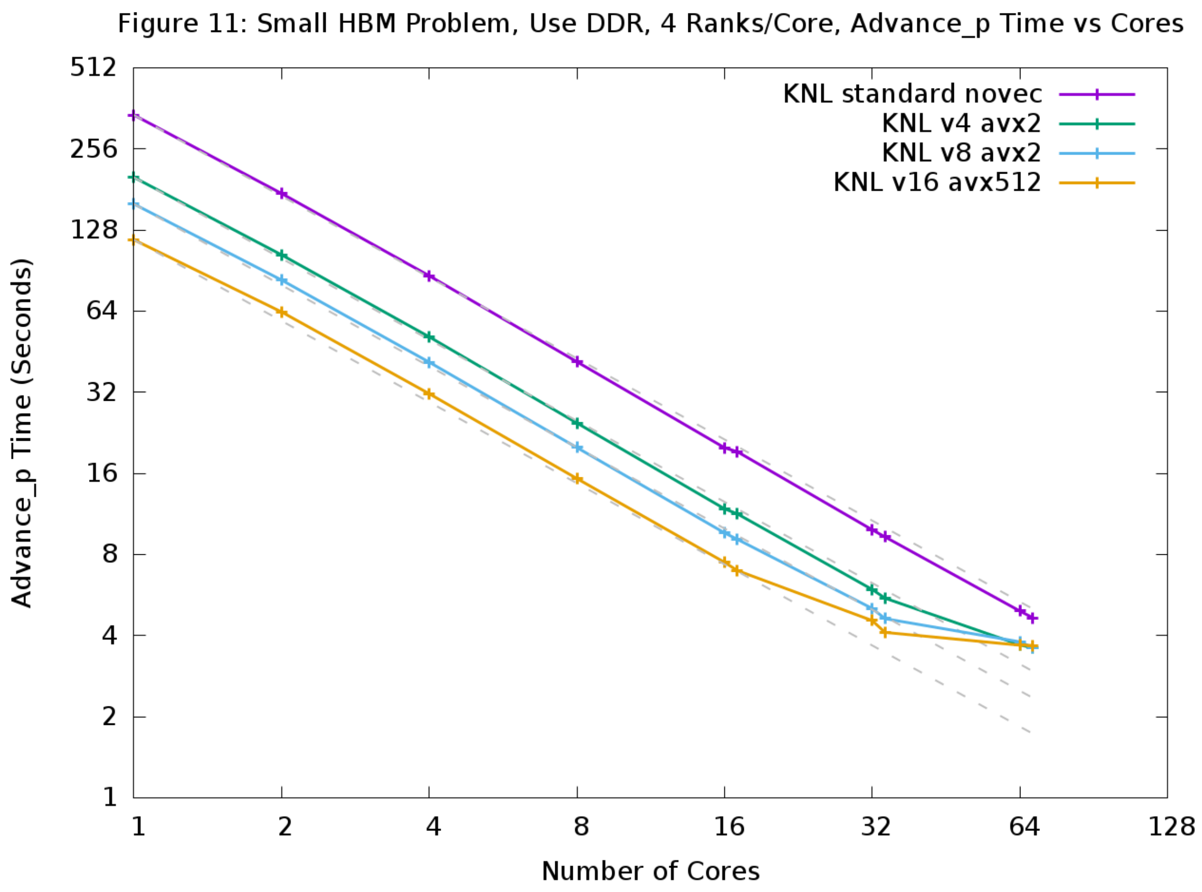
UNCLASSIFIED

# Figure 11: Small HBM Problem, Use DDR, 2 Ranks/Core, Advance_p Time vs Cores



UNCLASSIFIED

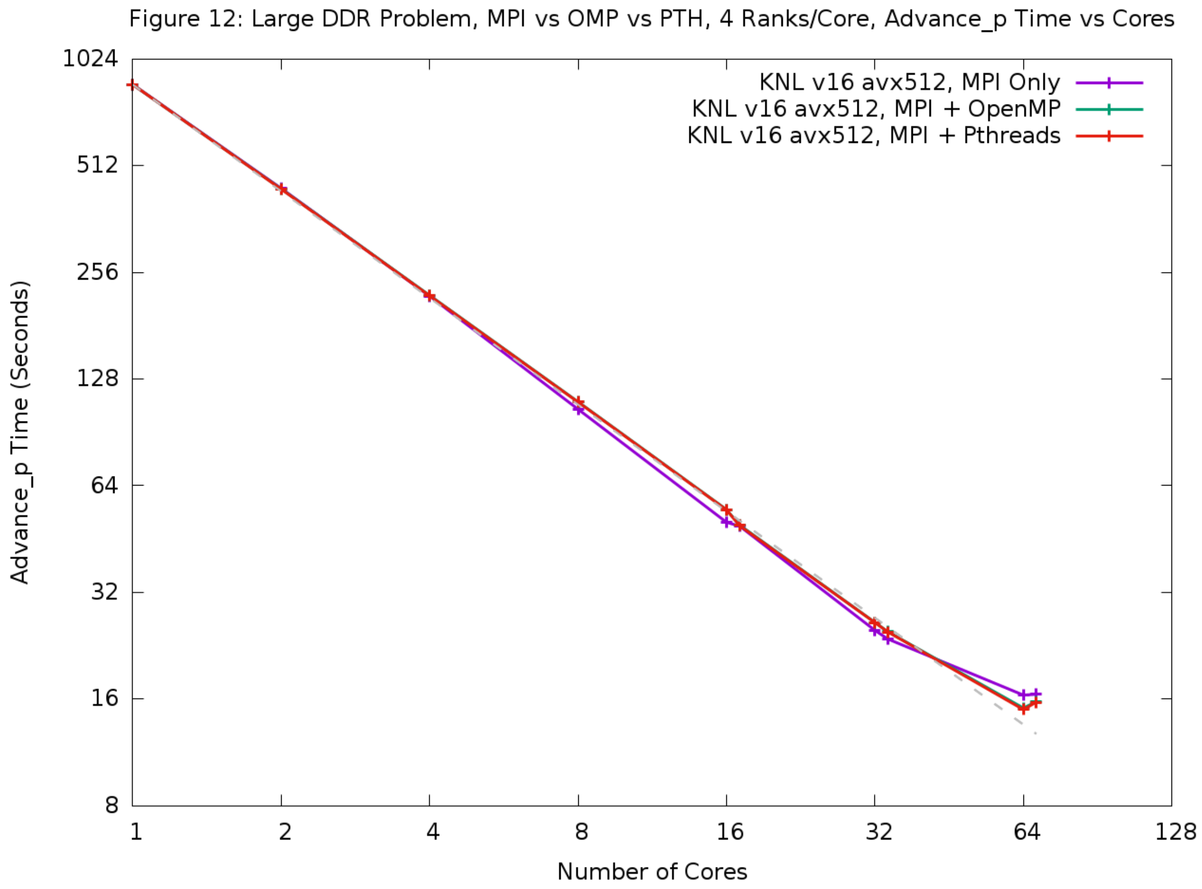
## Figure 12: Small HBM Problem, Use DDR, 4 Ranks/Core, Advance_p Time vs Cores



## OpenMP and Pthreads Performance

- Figure 13 presents strong scaling study of performance of VPIC OpenMP and Pthreads implementations compared to MPI only performance when using V16 implementation which has highest computational intensity.
- Both OpenMP and Pthreads give slightly better performance, ~10 percent, for a fully utilized KNL processor running the large DDR problem.
- For other cases not plotted, such as for running the small HBM problem or running with the no vectorization implementation or the V4 implementation, OpenMP and Pthreads perform slightly worse than the MPI only case.
- These performance results are not understood.  One speculation is that MPI only results in more contention for shared resources for the 4 execution threads/core case compared to either OpenMP or Pthreads.

# Figure 13: MPI vs OpenMP vs Pthreads, 4 Ranks/Core



UNCLASSIFIED

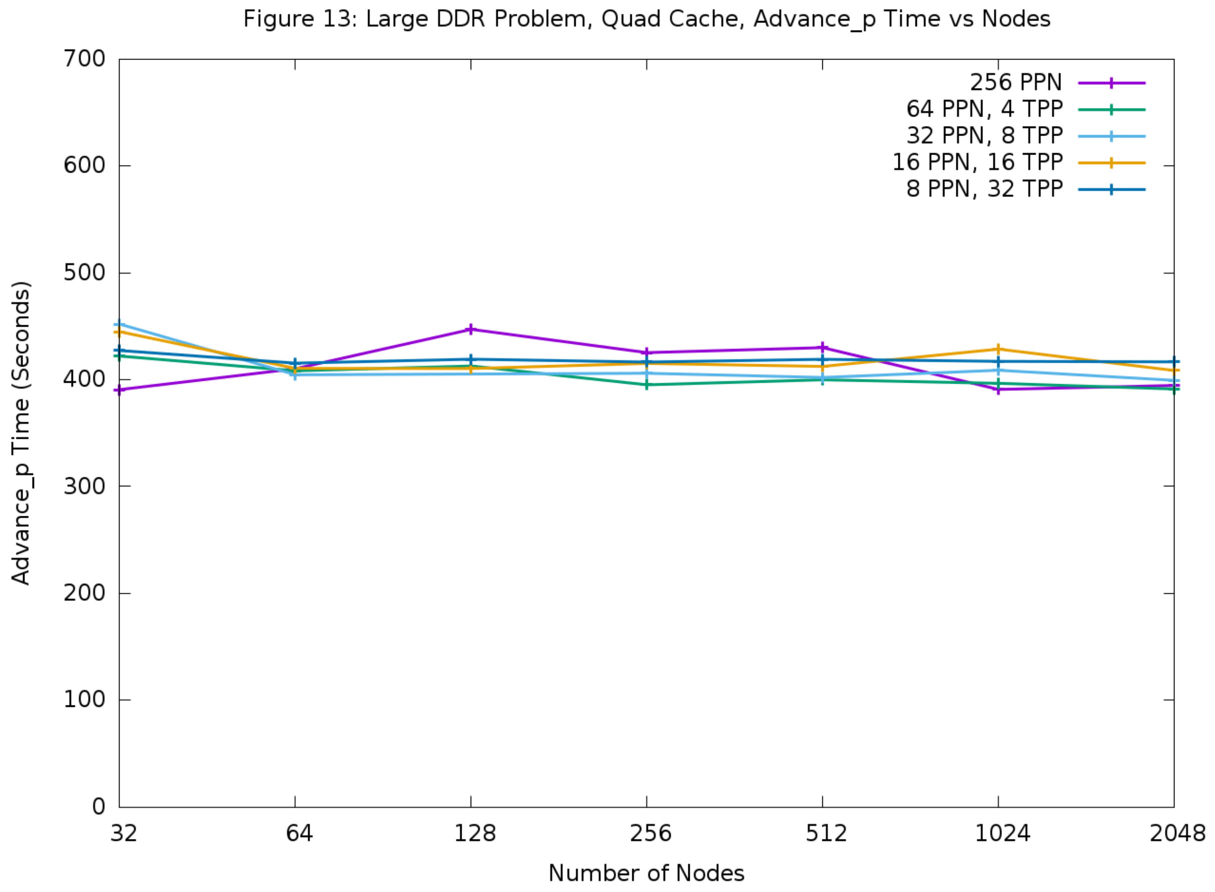
## Weak Scaling Results at Scale

- Figures 14 and 15 show performance as measured by time spent in the advance_p function versus number of nodes for several sets of KNL runs which were sized to use most of the DDR memory available on a node and were weak scaled across nodes.
- Runs were performed at 32, 64, 128, 256, 512, 1024 and 2048 nodes using 256 execution threads/node and 64 cores/node.
- Figure 14 shows runs performed using KNL quad cache mode.
- Figure 15 shows runs performed using KNL quad flat mode and using numactl –preferred=1 option to put any data structures in HBM that would fit. With 300 particles/cell, most or all of the grid sized data structures should fit in HBM.
- Both sets of runs included cases for MPI only and also MPI+Pthreads with 2, 4, 8, 16 and 32 Pthreads/rank.

## Weak Scaling Results at Scale (cont)

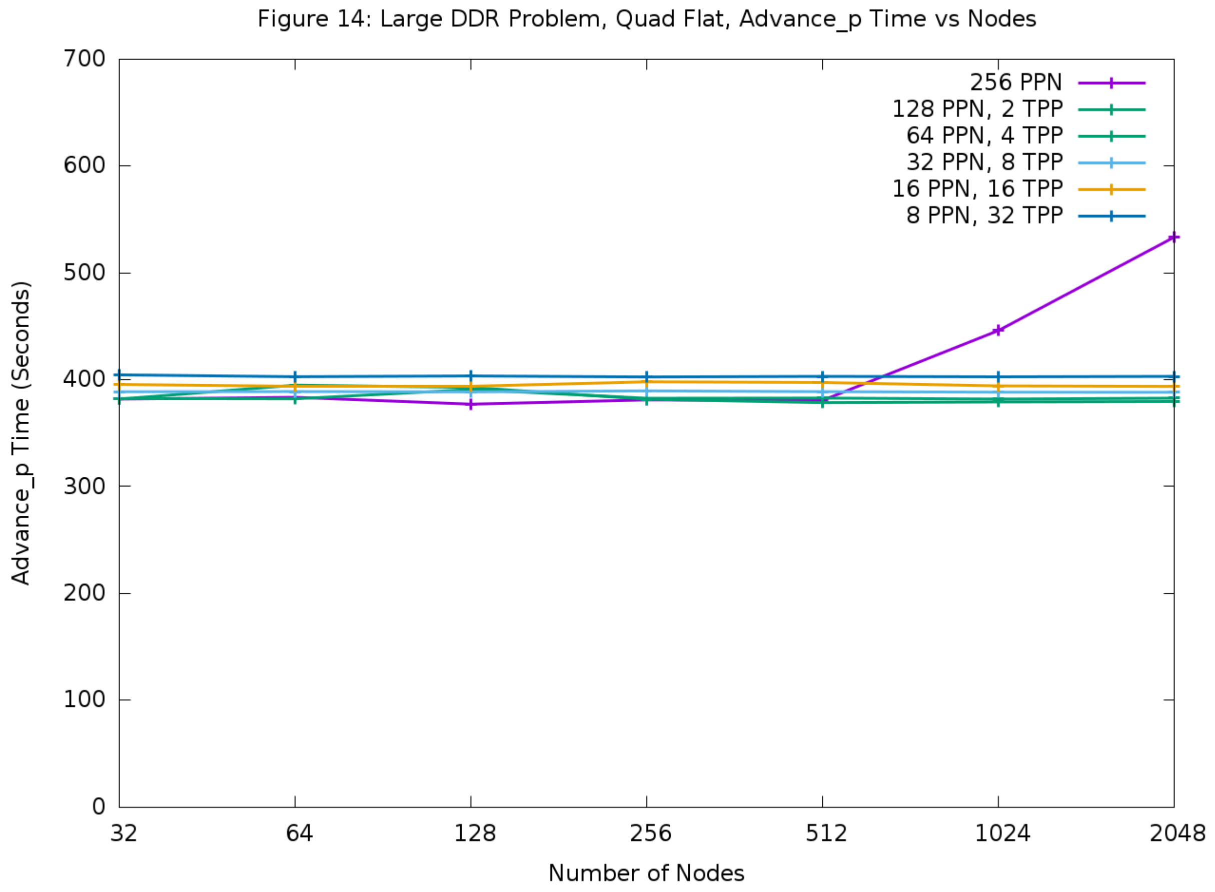
- Quad cache mode runs showed good weak scaling but larger variability in performance compared to runs in quad flat mode.
- Quad flat mode runs showed even better weak scaling than quad cache and much less variability in performance.
- The poor performance of MPI only for quad flat mode for 1024 and 2048 nodes is attributed to large memory allocations by Cray MPI getting preferentially allocated to HBM so that important VPIC data structures that reside in HBM for smaller node counts instead reside in DDR.  Cray MPI use of HBM can be suppressed by an environment variable.
- The performance of quad flat runs is ~10 percent better on average than runs using quad cache mode.
- Using more than 4 Pthreads/rank in quad flat mode results in degraded performance for these runs.  Better results may be achieved with more attention to controlling MPI rank placement and Pthread binding at higher thread counts/rank.

# Figure 14: Weak Scaled DDR Problem, Quad Cache



UNCLASSIFIED

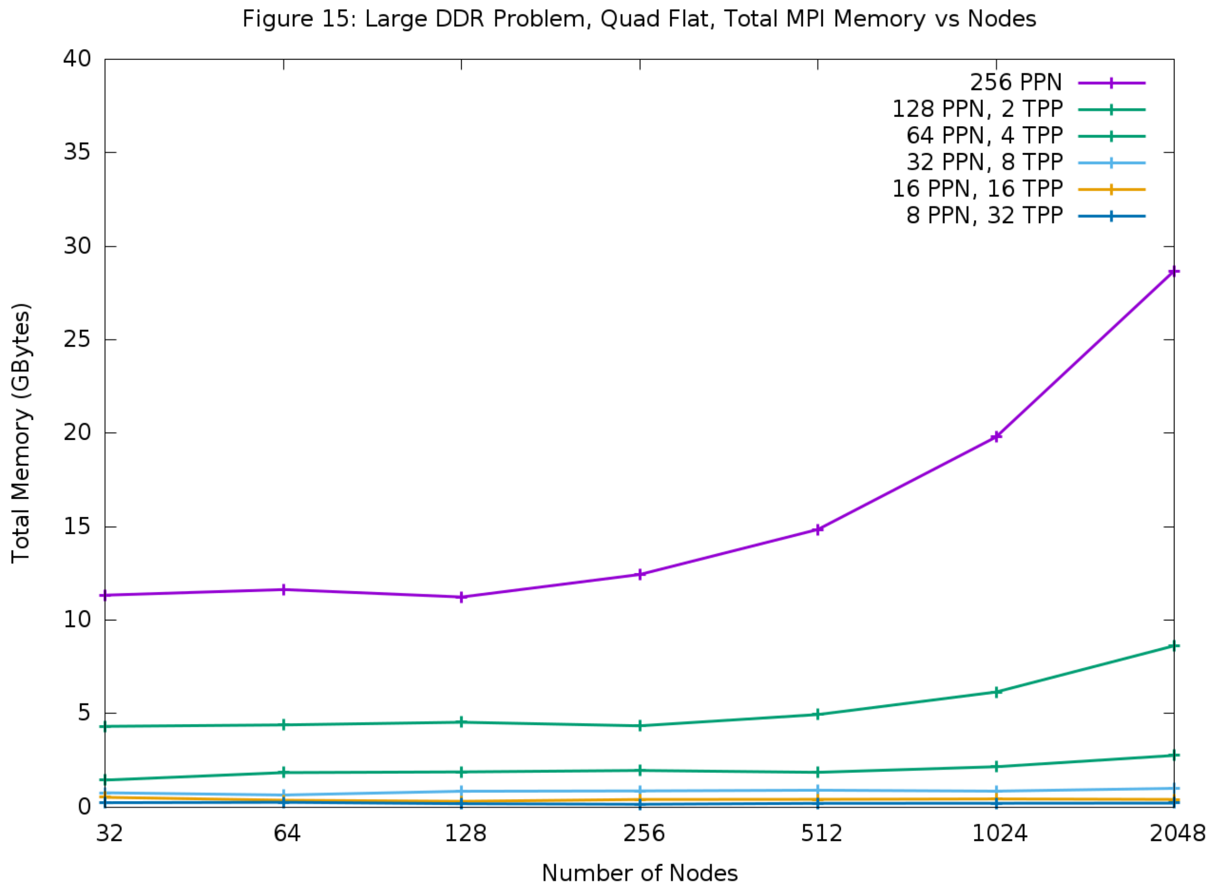
# Figure 15: Weak Scaled DDR Problem, Quad Flat



## Cray MPI Memory Usage on KNL

- Figure 16 shows total memory usage by Cray MPI for VPIC runs on KNL as a function of nodes for the quad flat runs shown in Figure 14.
- Cray MPI memory usage reported by Cray MPI using environment variable MPICH_MEMORY_REPORT=3.
- At 2048 nodes for MPI only case which uses 256 MPI ranks/node, MPI uses ~28 GB of memory which is ~30 percent of DDR memory per node on Trinity KNL nodes.
- At 2048 nodes, MPI memory usage scales nearly quadratically with total number of MPI ranks.
- VPIC gets significant performance benefit from using 4 execution threads per core versus 2.  Other applications may as well.
- VPIC can avoid this issue by using MPI+threads.  MPI only applications cannot.  To run at large scale on Trinity KNL, MPI only applications may have to sacrifice some performance to allow fitting into memory.

## Figure 15: Cray MPI Memory Usage vs Nodes



UNCLASSIFIED

## Conclusions

- Addition of wider SIMD vector support to VPIC using AVX2 implementation of V8 and AVX512 implementation of V16 has resulted in incremental improvement of VPIC single node performance.
- Performance improvements using V8 and V16 are significantly greater for the single thread and single core cases compared to a fully utilized KNL processor even for a small problem that fits entirely in HBM.
- On KNL, VPIC is memory bandwidth limited when running purely in DDR – so key to improved performance for primary use case of large DDR problem depends at least partly on more effective use of HBM.
- On KNL, VPIC is not memory bandwidth limited when running in HBM – so there seems potential for greater performance improvements that would depend on improving single thread and single core performance.
- Unless memory usage of MPI can be reduced, running VPIC optimally at scale requires using either existing OpenMP or Pthread implementation.

## Future Work

- Investigate single thread and single core performance of VPIC on KNL using Intel Vtune in collaboration with Intel engineers.
- Investigate register usage of V4, V8 and V16 SIMD implementations with Intel engineers.
- Investigate L1/L2 cache usage of V4, V8 and V16 SIMD implementations with Intel engineers.
- Investigate possibility of improved prefetch of data via software prefetching with Intel engineers.
- Investigate use of buffering of particle data in to and out of HBM to see if better HBM usage can be achieved for primary use case of large DDR problem.
- Work on performance improvements to other areas of VPIC besides particle advance including I/O and particle sort scaling.
- Investigate memory usage scaling of OpenMPI on KNL.