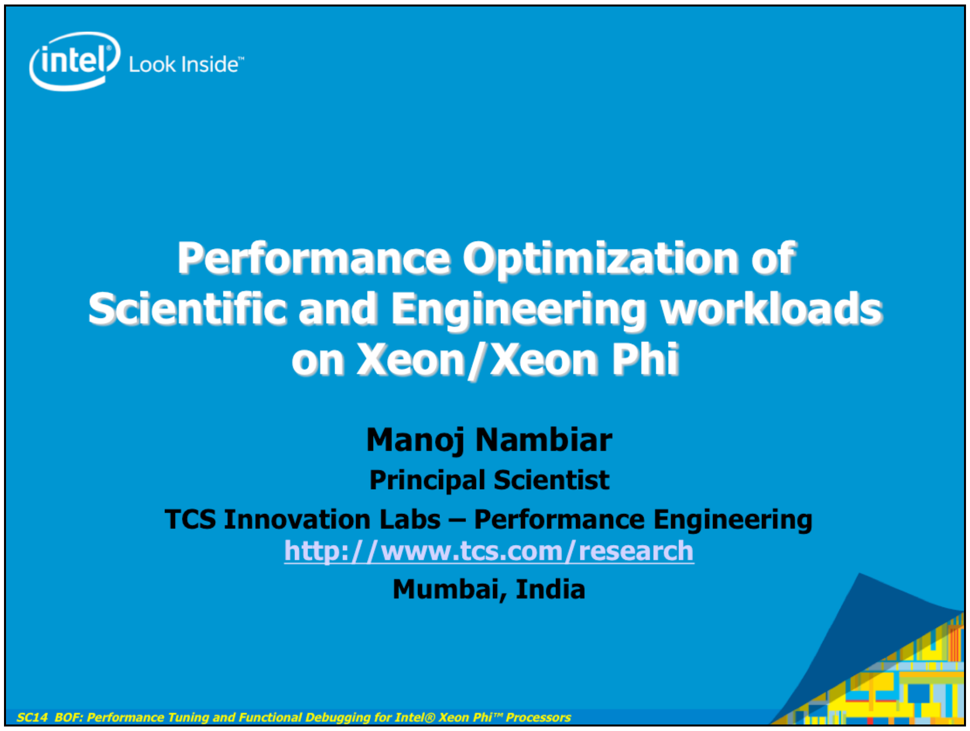
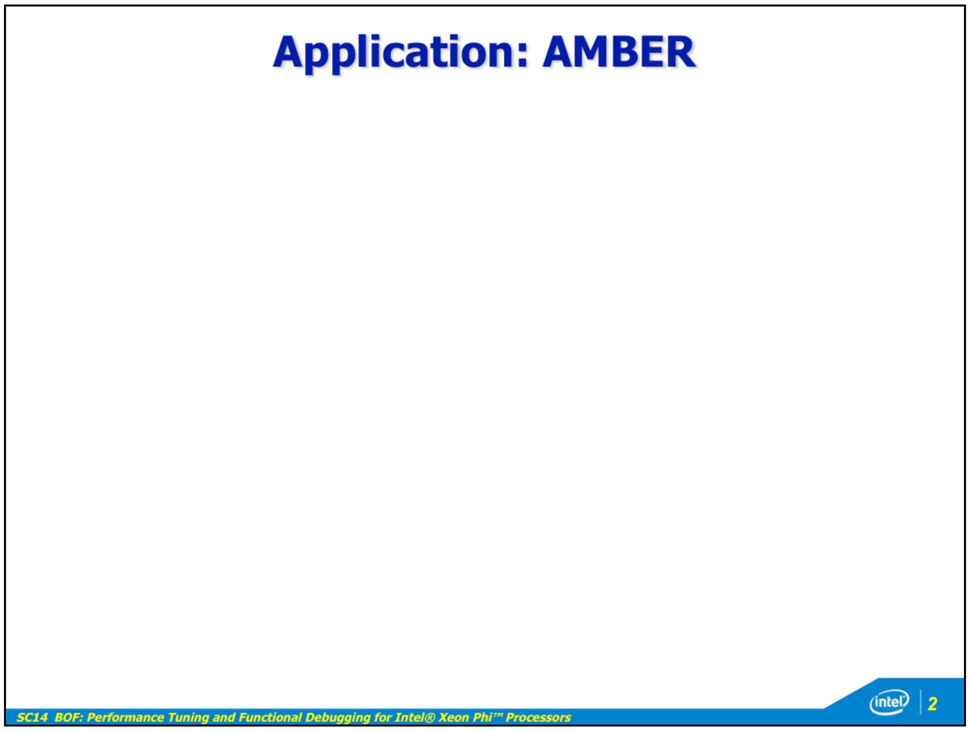


Performance Optimization of Scientific and Engineering workloads on Xeon/Xeon Phi

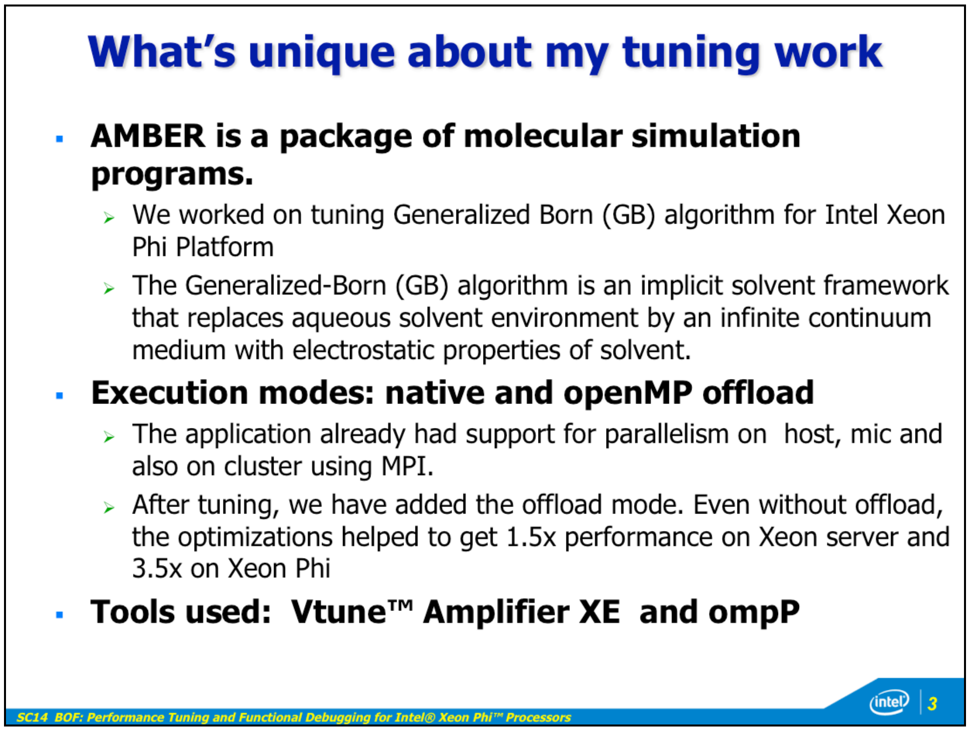
Manoj Nambiar
Principal Scientist
TCS Innovation Labs – Performance Engineering
http://www.tcs.com/research
Mumbai, India

Comments are from CJ Newburn

You have ~3 presentations here. All are very valuable. Could you prioritize one over others? We'll also discuss this



Application: AMBER



What's unique about my tuning work

- **AMBER is a package of molecular simulation programs.**
  - We worked on tuning Generalized Born (GB) algorithm for Intel Xeon Phi Platform
  - The Generalized-Born (GB) algorithm is an implicit solvent framework that replaces aqueous solvent environment by an infinite continuum medium with electrostatic properties of solvent.
- **Execution modes: native and openMP offload**
  - The application already had support for parallelism on host, mic and also on cluster using MPI.
  - After tuning, we have added the offload mode. Even without offload, the optimizations helped to get 1.5x performance on Xeon server and 3.5x on Xeon Phi
- **Tools used: Vtune™ Amplifier XE and ompP**

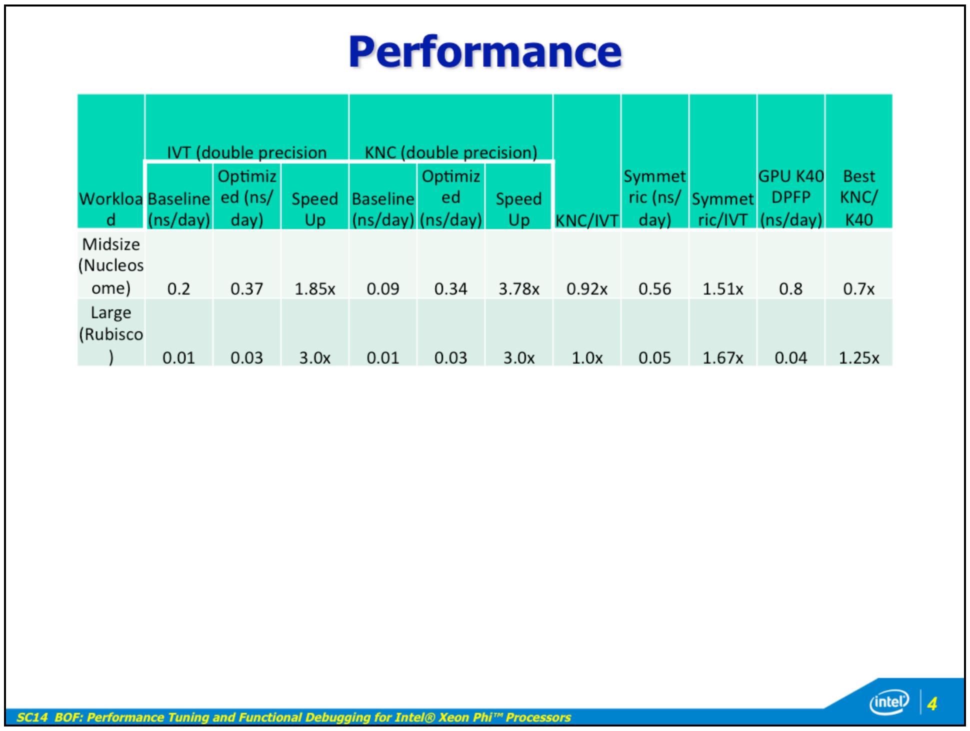
NOTES:

CJ:

If 50% and 250% better mean 1.5x and 3.5x, please use the multiplicative factor – it's less ambiguous

BM:

Changed from % to x



Performance

Workload	IVT (double precision			KNC (double precision)			KNC/IVT	Symmetric (ns/day)	Symmetric/IVT	GPU K40 DPFP (ns/day)	Best KNC/K40
	Baseline (ns/day)	Optimized (ns/day)	Speed Up	Baseline (ns/day)	Optimized (ns/day)	Speed Up					
Midsize (Nucleosome)	0.2	0.37	1.85x	0.09	0.34	3.78x	0.92x	0.56	1.51x	0.8	0.7x
Large (Rubisco)	0.01	0.03	3.0x	0.01	0.03	3.0x	1.0x	0.05	1.67x	0.04	1.25x

SC14 BOF: Performance Tuning and Functional Debugging for Intel® Xeon Phi™ Processors

intel 4

CJ:

Please show the speedup ratios of KNC/IVT (that's native, right?) [0.92x,1x], symmetric/IVT [1.51x,1.67x], best KNC/K40 [0.7x, 1.25x]

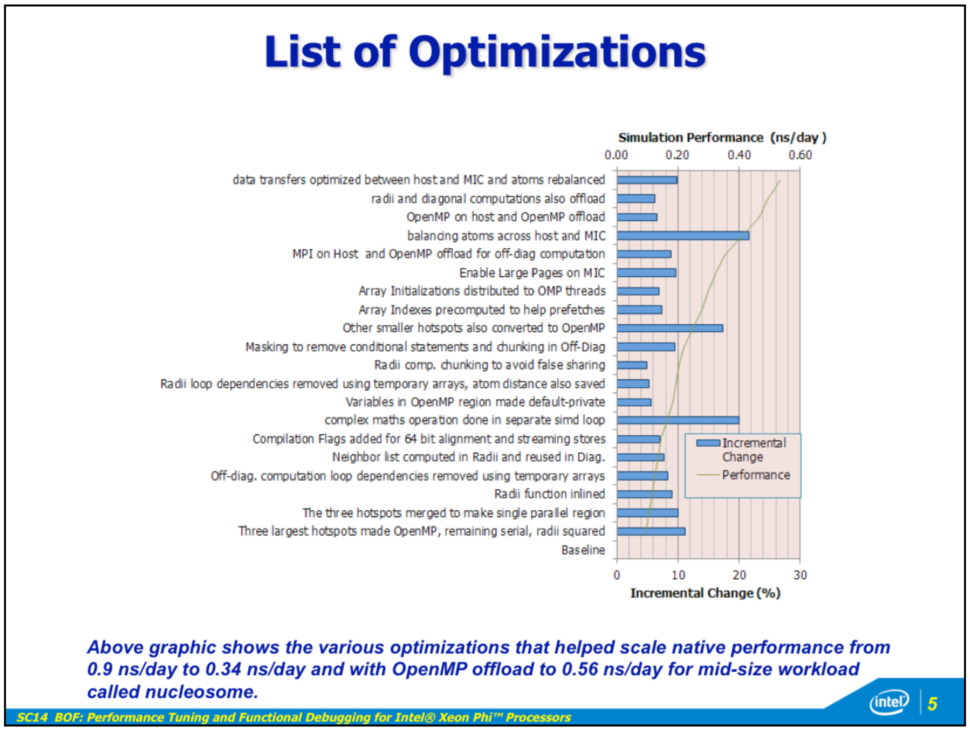
Is this DP, DP? You have to specify, since it makes a huge difference. On Rubisco, does the NV SP,FP still beat a symmetric Xeon/MIC MPI, for which I presume you still use DP,DP?

Did you have unit stride on Xeon, or memory coalescing on NV?

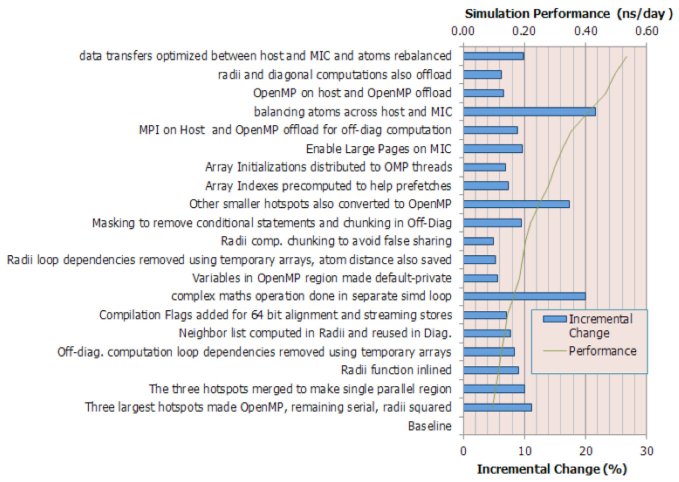
How did you get around GB's use of the compress idiom?

BM:

- Speed up ratios added
- CUDA numbers are for DPFP (Amber 14 does not have DPDP version)
- CUDA SPFP gives 4-6x times DPFP performance so we are not yet in position to compete with SPFP



List of Optimizations



Above graphic shows the various optimizations that helped scale native performance from 0.9 ns/day to 0.34 ns/day and with OpenMP offload to 0.56 ns/day for mid-size workload called nucleosome.

SC14 BOF: Performance Tuning and Functional Debugging for Intel® Xeon Phi™ Processors 5

CJ Notes:

This is a great graph!

Perhaps you could make Excel show data points which are the incremental speedups relative to the last line, or perhaps a second axis

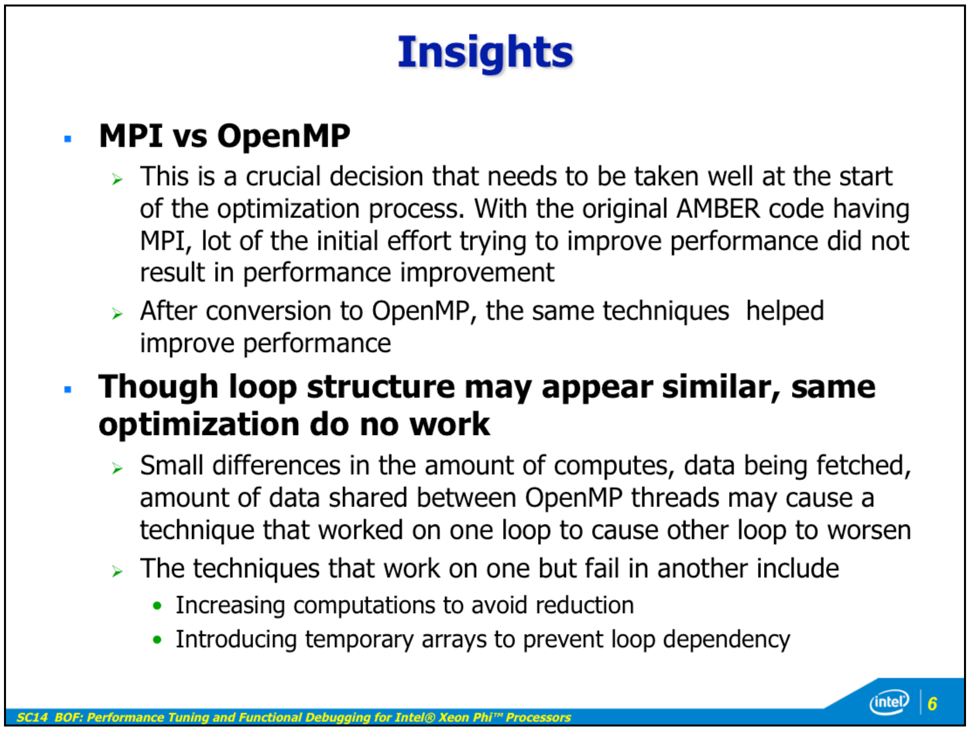
Could you provide more detail on these in the speaker notes, which we'd like to post?

I'm curious about what the trick you used was to make it vect with a temporary. Was this the variable that was used just after the loop as a function call parameter? We're working on a fix for that in the compiler; not sure whether it's in place yet. Ashraf (mohammad.ashraf.bhuiyan@intel.com), who's been working on AMBER at Intel, will know.

CJ2:

vectorization via temp not addressed yet.

Thanks for the others


Insights

- **MPI vs OpenMP**
  - This is a crucial decision that needs to be taken well at the start of the optimization process. With the original AMBER code having MPI, lot of the initial effort trying to improve performance did not result in performance improvement
  - After conversion to OpenMP, the same techniques helped improve performance
- **Though loop structure may appear similar, same optimization do no work**
  - Small differences in the amount of computes, data being fetched, amount of data shared between OpenMP threads may cause a technique that worked on one loop to cause other loop to worsen
  - The techniques that work on one but fail in another include
    - Increasing computations to avoid reduction
    - Introducing temporary arrays to prevent loop dependency

SC14 BOF: Performance Tuning and Functional Debugging for Intel® Xeon Phi™ Processors 6

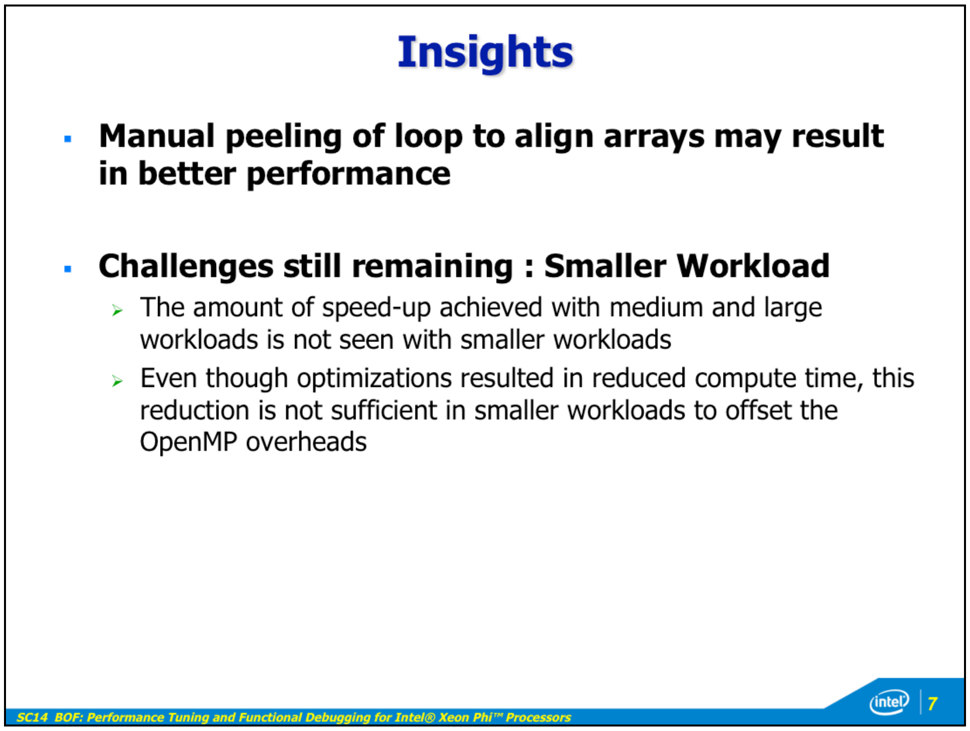
CJ

It'd be helpful to file test cases with our compiler, so we know about these and can learn from them. Sometimes, it's just a matter of code trade-offs. Other times there may be something we can do about it in the compiler.

BM

We are discussing these issues with Ashraf and Sumedh

– based on their recommendations, we will be filing some issues with compiler team



Insights

- **Manual peeling of loop to align arrays may result in better performance**

- **Challenges still remaining : Smaller Workload**
  - The amount of speed-up achieved with medium and large workloads is not seen with smaller workloads
  - Even though optimizations resulted in reduced compute time, this reduction is not sufficient in smaller workloads to offset the OpenMP overheads