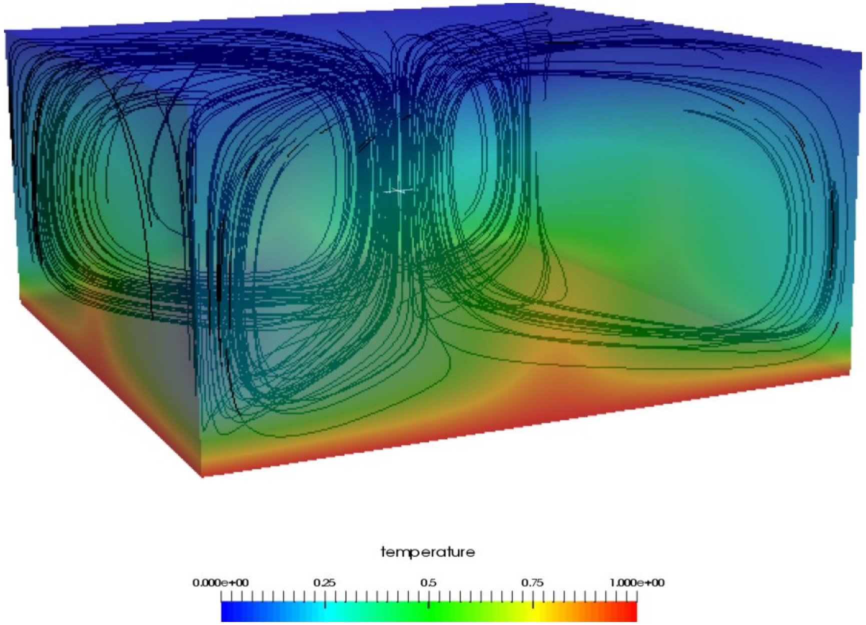
DL_MESO Code Modernization

Sergi Siso
Application Performance Engineer
IPCC@Hartree, STFC Daresbury Laboratory
United Kingdom

DL_MESO_LBE

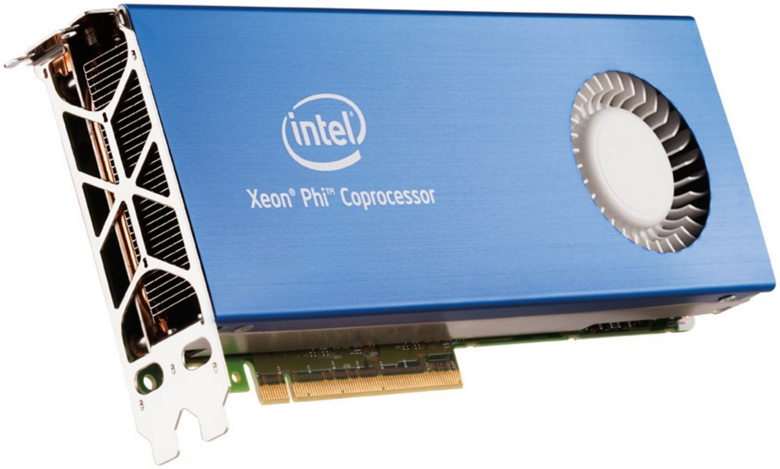
- Is a C++ general purpose mesoscopic simulation package
- Simulate multi component lattice-gas systems using the LBE
- It is used to model systems with **multiple fluids** and/or phases coupled to **solute diffusion and heat transfer**, as well as apply **geometrically complex boundaries** comparatively easily.



Differences from other LB codes

- Multi-phase and multi-component ☺
- We need the pseudo-potentials of neighbour particles to compute the collision.
  - Collision: Not local (unlike simpler LB methods ☹)
  - Streaming: Not local
- The fact that collisions are not local increases memory bandwidth intensity and makes us go at least twice through all the data-structure in each time-step. ☹

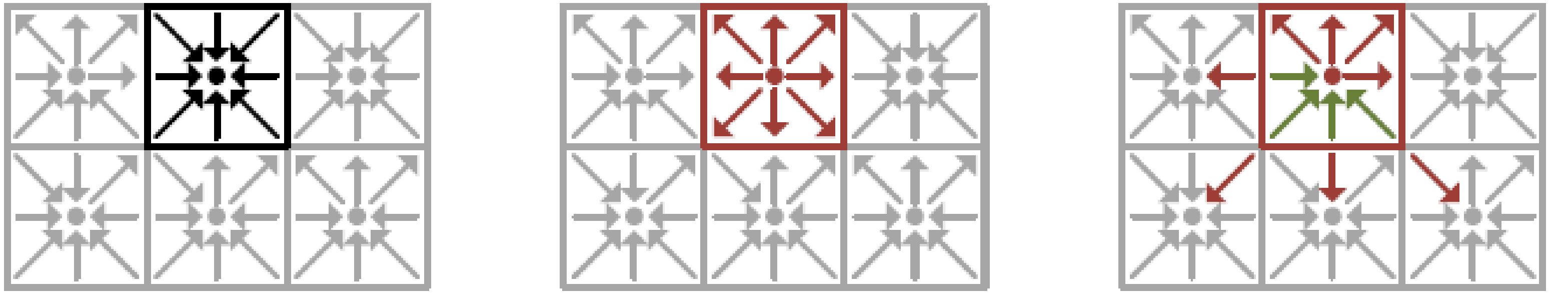
Intel Xeon Phi



Performance of the original code was disappointing in the Xeon Phi

Xeon Phi 5110P	
Cores	60
Logical cores	240
Frequency	1.053GHz
GFLOPs	2,020
SIMD width	512 Bits
Memory	8GB
Memory B/W	320GB/s

Original code: SWAP Algorithm



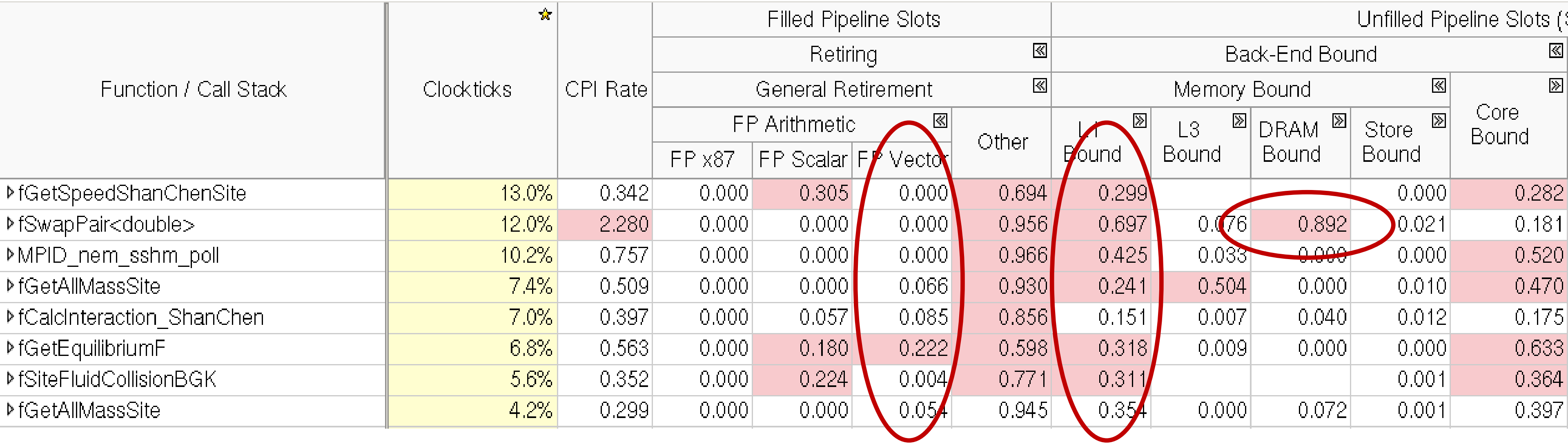
(a) Reached node to update.

(b) Collision.

(c) After Swap.

- Efficient memory usage
- Strict processing order of the lattice nodes
  - Difficult threading *
  - Difficult vectorization
  - Difficult to apply memory blocking optimizations

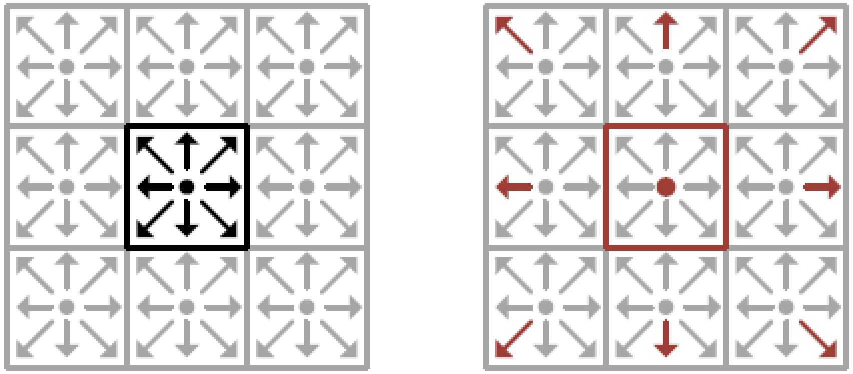
Original code: Performance



Function / Call Stack	Clockticks	CPI Rate	Filled Pipeline Slots				Unfilled Pipeline Slots (				
			Retiring				Back-End Bound				
			General Retirement				Memory Bound				Core Bound
			FP Arithmetic			Other	L1 Bound	L3 Bound	DRAM Bound	Store Bound	
			FP x87	FP Scalar	FP Vector						
fGetSpeedShanChenSite	13.0%	0.342	0.000	0.305	0.000	0.694	0.299			0.000	0.282
fSwapPair<double>	12.0%	2.280	0.000	0.000	0.000	0.956	0.697	0.076	0.892	0.021	0.181
MPID_nem_sshm_poll	10.2%	0.757	0.000	0.000	0.000	0.966	0.425	0.033	0.000	0.000	0.520
fGetAllMassSite	7.4%	0.509	0.000	0.000	0.066	0.930	0.241	0.504	0.000	0.010	0.470
fCalcInteraction_ShanChen	7.0%	0.397	0.000	0.057	0.085	0.856	0.151	0.007	0.040	0.012	0.175
fGetEquilibriumF	6.8%	0.563	0.000	0.180	0.222	0.598	0.318	0.009	0.000	0.000	0.633
fSiteFluidCollisionBGK	5.6%	0.352	0.000	0.224	0.004	0.771	0.311			0.001	0.364
fGetAllMassSite	4.2%	0.299	0.000	0.000	0.054	0.945	0.354	0.000	0.072	0.001	0.397

- Memory bandwidth was a problem (implemented passing 5 times through all data-structure)
- MPI Only was the best version (OpenMP just made cache worse!)
- We added SIMD pragmas in inner loops to solve Vector Advisor spotted issues, but small improvements code-wide.
- Xeon Phi version considerably worse (x2 slower)

New code: Two-Grid Algorithm



(a) Push: lattice *A*.   (b) Push: lattice *B*.

- Doubles the SWAP alg. memory usage
- No restrictions in lattice processing order
  - Natural parallelization
  - ‘Not so difficult’ vectorization
  - Easy to apply memory blocking optimizations

New code: New Data Layout

Original Version:

~ Array of Structures

[x] [y] [z] [fluid] [lattice]

threaded vectorized

- – Good memory locality among each grid point
- – Vectorization of some inner loops
- – Low inner loop trip count

New code: New Data Layout

Original Version:

~ Array of Structures

$$\underbrace{[x]\,[y]\,[z]}_{\text{threaded}}\,[\text{fluid}]\,\underbrace{[\text{lattice}]}_{\text{vectorized}}$$

- Good memory locality among each grid point
- Vectorization of some inner loops
- Low inner loop trip count

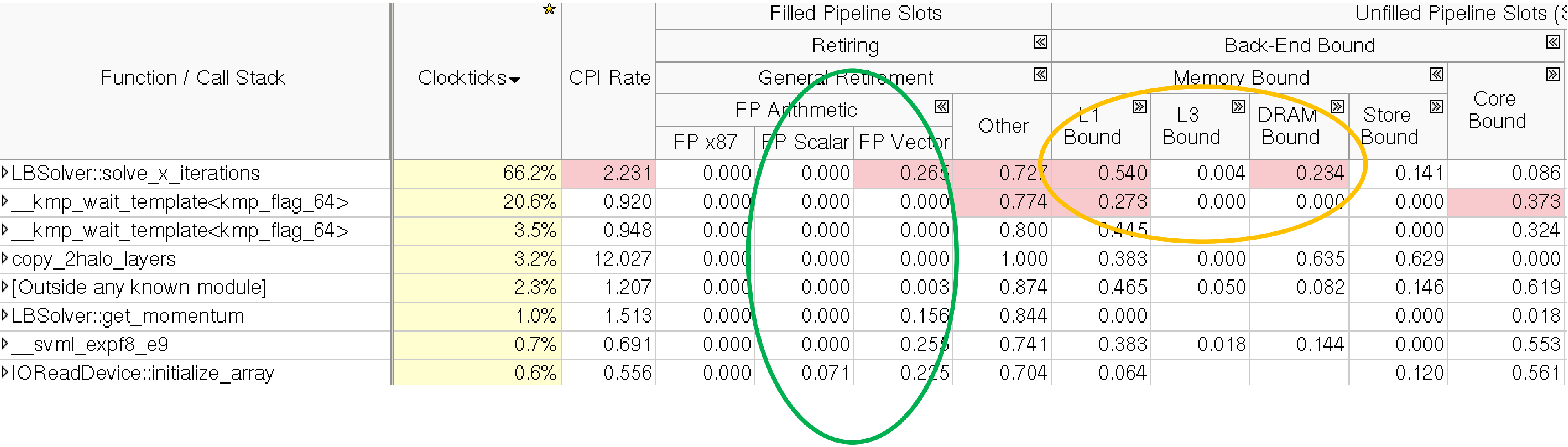
New Version:

~ Array of Structures of Array

$$\underbrace{[x]\,[y]}_{\text{threaded}}\,\underbrace{\underbrace{[\text{fluid}]\,[\text{lattice}]}_{\text{unrolled}}\,[z]}_{\text{vectorized}}$$

- Maintains some locality
- Enough threading elements
- Outer loop vectorization
- Unrolled inner loops
- Bigger trip count

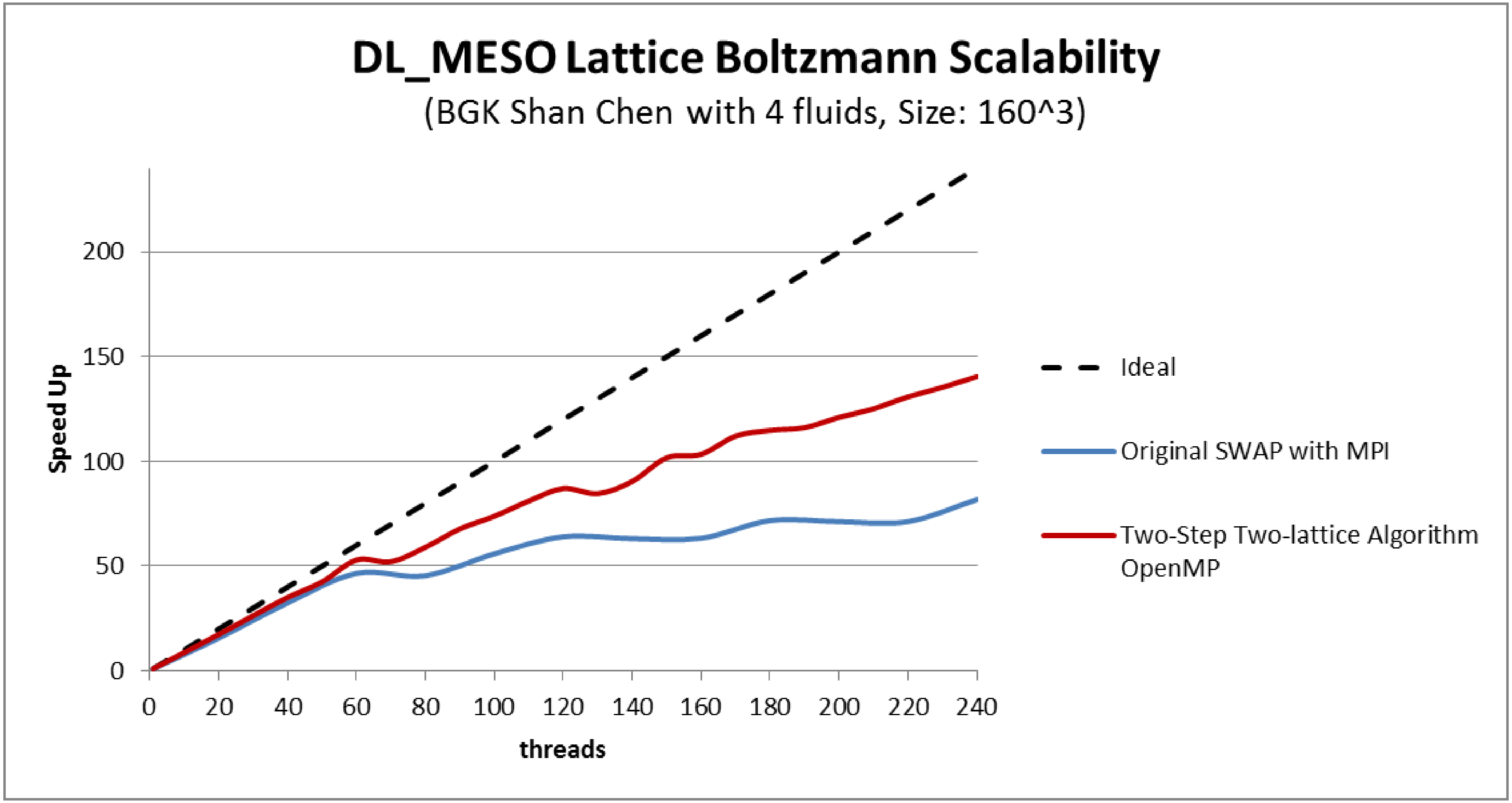
New code: Performance Analysis



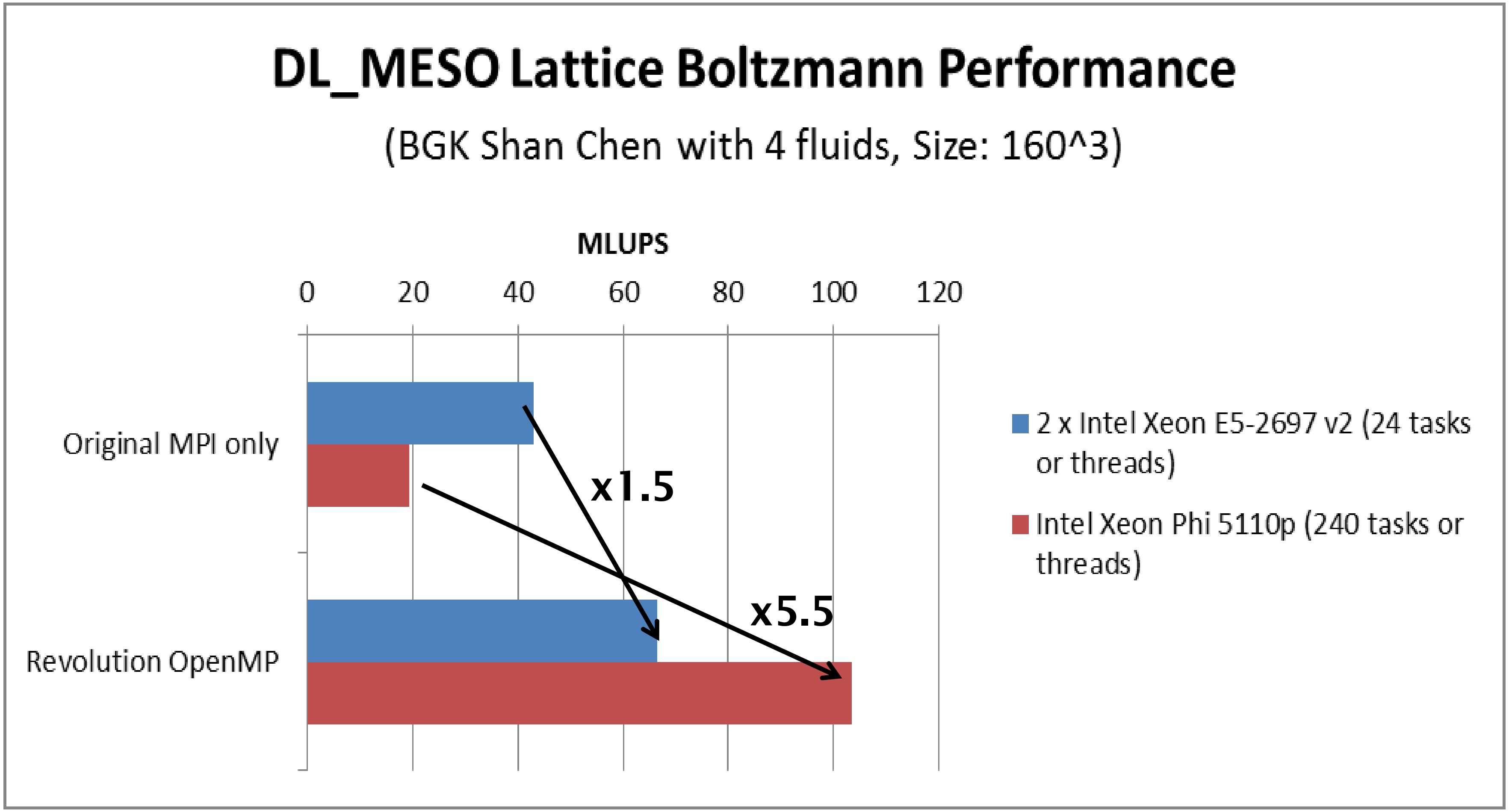
Function / Call Stack	Clockticks	CPI Rate	Filled Pipeline Slots				Unfilled Pipeline Slots (				
			Retiring				Back-End Bound				
			General Retirement				Memory Bound				Core Bound
			FP Arithmetic			Other	L1 Bound	L3 Bound	DRAM Bound	Store Bound	
			FP x87	FP Scalar	FP Vector						
LBSolver::solve_x_iterations	66.2%	2.231	0.000	0.000	0.265	0.727	0.540	0.004	0.234	0.141	0.086
__kmp_wait_template<kmp_flag_64>	20.6%	0.920	0.000	0.000	0.000	0.774	0.273	0.000	0.000	0.000	0.373
__kmp_wait_template<kmp_flag_64>	3.5%	0.948	0.000	0.000	0.000	0.800	0.445			0.000	0.324
copy_2halo_layers	3.2%	12.027	0.000	0.000	0.000	1.000	0.383	0.000	0.635	0.629	0.000
[Outside any known module]	2.3%	1.207	0.000	0.000	0.003	0.874	0.465	0.050	0.082	0.146	0.619
LBSolver::get_momentum	1.0%	1.513	0.000	0.000	0.156	0.844	0.000			0.000	0.018
__svml_expf8_e9	0.7%	0.691	0.000	0.000	0.255	0.741	0.383	0.018	0.144	0.000	0.553
IOReadDevice::initialize_array	0.6%	0.556	0.000	0.071	0.225	0.704	0.064			0.120	0.561

- Almost all FP arithmetic is done in Vector units (x5 vs –no-vec –no-simd)
- Much better cache utilization (still could be better?)

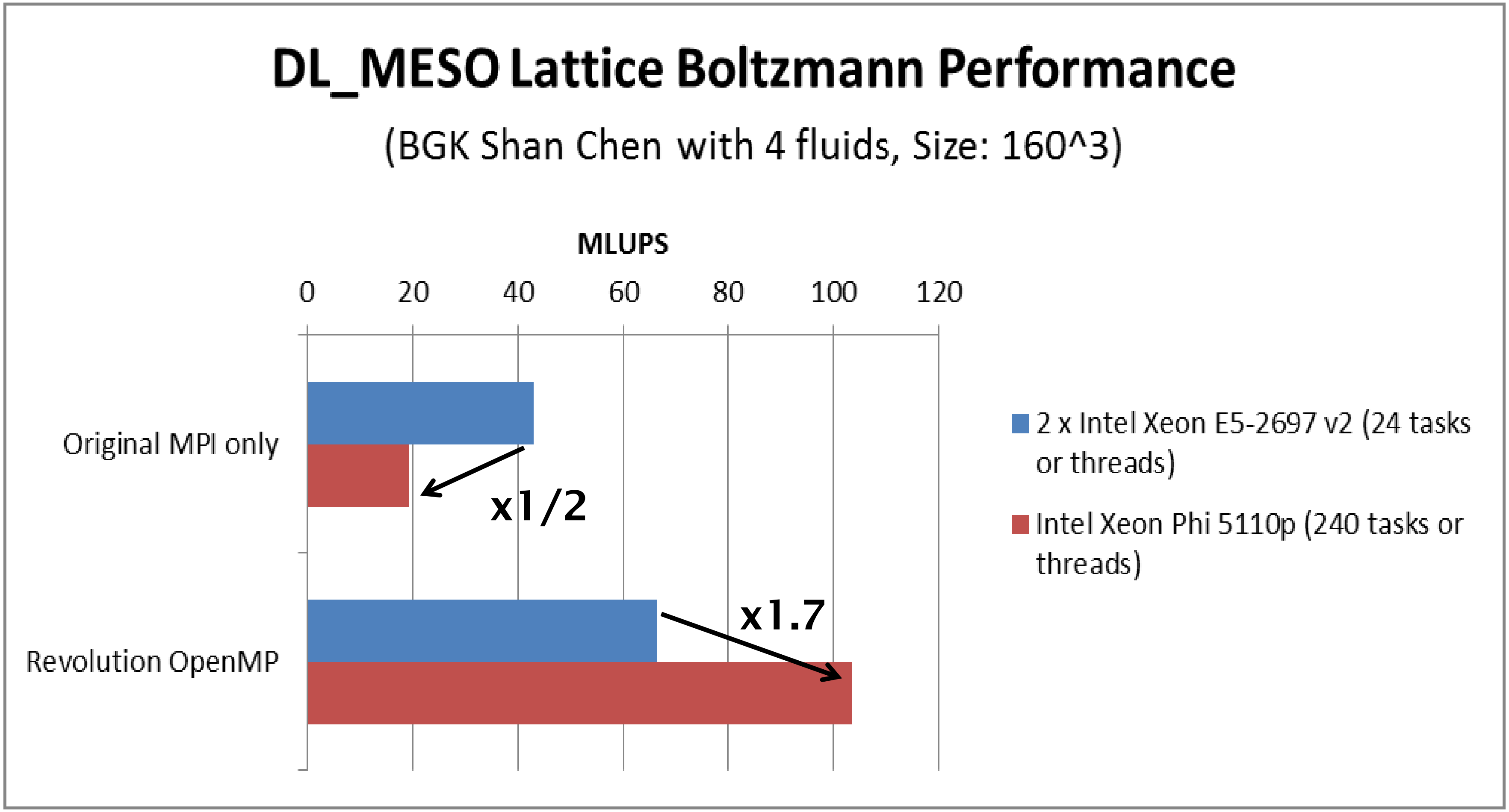
Scalability Comparison



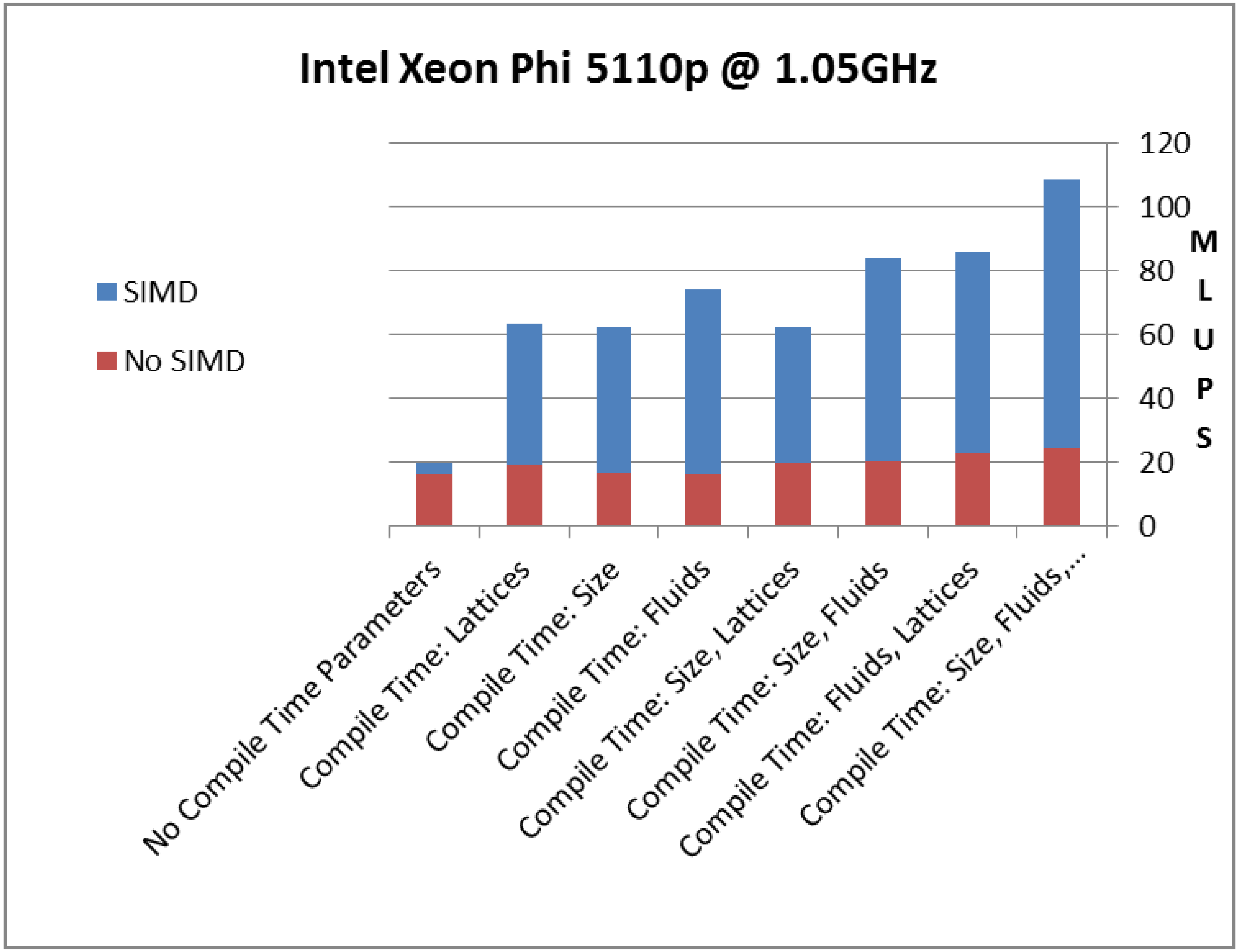
Performance Comparison



Performance Comparison



Compile-time parameters



Optimization report

```
LOOP BEGIN at src/LBSolver/LBSolver.cpp(141,9)
vectorization support: vector length 16
vectorization support: normalized vectorization overhead 0.115
SIMD LOOP WAS VECTORIZED
unmasked unaligned unit stride loads: 306
masked unaligned unit stride loads: 7
masked unaligned unit stride stores: 1
masked indexed (or gather) loads: 1
--- begin vector loop cost summary ---
scalar loop cost: 6451
vector loop cost: 633.500
estimated potential speedup: 8.500
serialized function calls: 3
type converts: 10
```

Conclusions and Insights

- Sometimes it is important to take a step back and see if the algorithm/implementation is appropriate for a highly-parallel architecture like the Intel Xeon Phi.

- **Intel VTune and Vector Advisor** were essential to spot the real code issues and tackle those.

- Not always necessary to rely on intrinsics to greatly improve the code performance, but try to **provide the compiler as much information as you can !**

Future work

- Xeon Phi port just have a subset of the original options.
- Better software engineering if #fluids and #lattices are converted to C++ templates (or delayed evaluation).
- Reintroduce MPI for inter-node communication, should be easy, serial code already implemented with halo copy functions for the periodic boundaries.
- Tested with a prototype KNL and the results look promising, port to production KNL when these are available.

Reference/Codes

- DL_MESO Webpage: http://www.scd.stfc.ac.uk/SCD/40694.aspx
- M.A. Seaton et al. "DL_MESO: highly scalable mesoscale simulations", *Mol. Sim.* (2013). doi:10.1080/08927022.2013.772297
- DL_MESO Repository: https://ccpforge.cse.rl.ac.uk/gf/project/dl_meso/
  - Xeon Phi work on MINILBE branch
- EMIT Proceedings
- Contact:
  - Sergi Siso: sergi.siso@stfc.ac.uk
  - Luke Mason: luke.mason@stfc.ac.uk
  - Michael Seaton: michael.seaton@stfc.ac.uk

Questions ?