VLPL-S Optimization on Knights Landing

英特尔软件与服务事业部

周姗

2016.5

Agenda

- **VLPL-S性能分析**
- **VLPL-S性能优化**
- **总结**

VLPL-S Workload Descriptions

- VLPL-S is the in-house code from SJTU, paralleled with MPI and written in C++. The application is about Particle-in-Cell method for laser plasma simulation by solving the particles motion equation, current density distribution and Maxwell equations.
- Disable the result output and the wall time of computation is performance benchmark. For this app, the result output is finished by 1st rank collecting data from other ranks.
- Workload Info

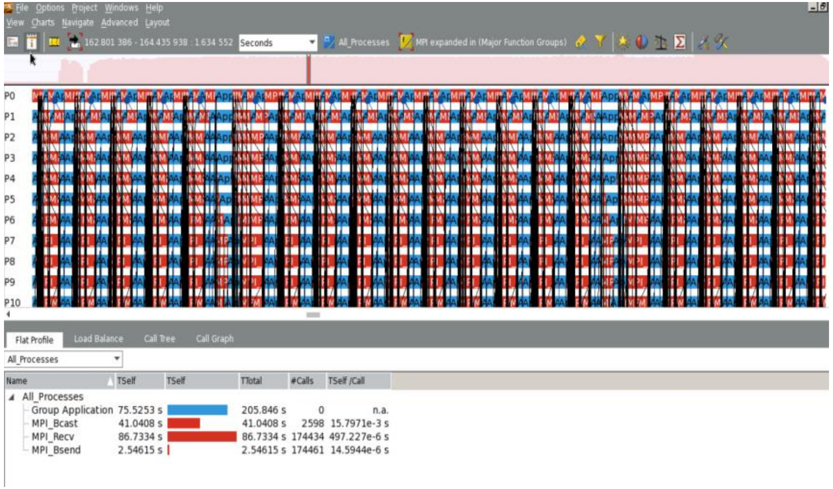
	Description	Number of Cells on X and Y	Particles per Cell	Number of Time steps	Precision
V2d-test1.ini	The total particles numbers in all cells will keep the same and the particles moves from one cell to another. It's one of typical case in engineering application.	2500x240	9	200	SP
V2d-test2.ini	The total particles number will increase as the time step goes on. For some cell, the particles number would increase to 5x. And the load balance would become better and better when the time step goes on.	2500x240	9	100	SP
V2d-test3.ini	The workload has good load balance used for benchmark. And the particles number would not change a lot in one cell.	1200x1200	16	20	SP
V2d-test3.big.ini	All of the parameter is the same with v2d-test3.ini except that the cells number on X and Y Direction is twice. And the particles per cell is 9.	2400x2400	9	20	SP

Machine Configurations

Processor	E5-2699v3	E5-2697v4	Xeon Phi 7210	Xeon Phi 7250
Stepping	2	1(B0)	1 (B0)	1 (B0)
Sockets / TDP	2S / 290W	2S / 290W	1S / 215W	1S / 215W
Frequency / Cores / Threads	2.3GHz / 36/ 72	2.3GHz / 36 / 72	1.3GHz / 64 / 256	1.4GHz / 68 / 272
DDR4 Memory	128GB 2133MHz	128GB 2400MHz	6x16GB 2133MHz	6x16GB 2400MHz
MCDRAM	N/A	N/A	16 GB Flat	16 GB Flat
Cluster/Snoop Mode	Home	Home	Quadrant	Quadrant
Turbo	Enabled	Enabled	Enabled	Enabled
BIOS	SE5C610.86B.01.01.0008.021120151325	GRRFSDP1.86B.0271.R00.1510301446	GVPRCRB1.86B.0010.R00.1603251732	GVPRCRB1.86B.0010.R00.1603251732
Operating System	CentOS release 6.7(Final)	CentOS release 6.7(Final)	RHEL 7.1 (3.10.0-327.0)	

VLPL-S Analysis using Intel® Trace Analyzer and Collector

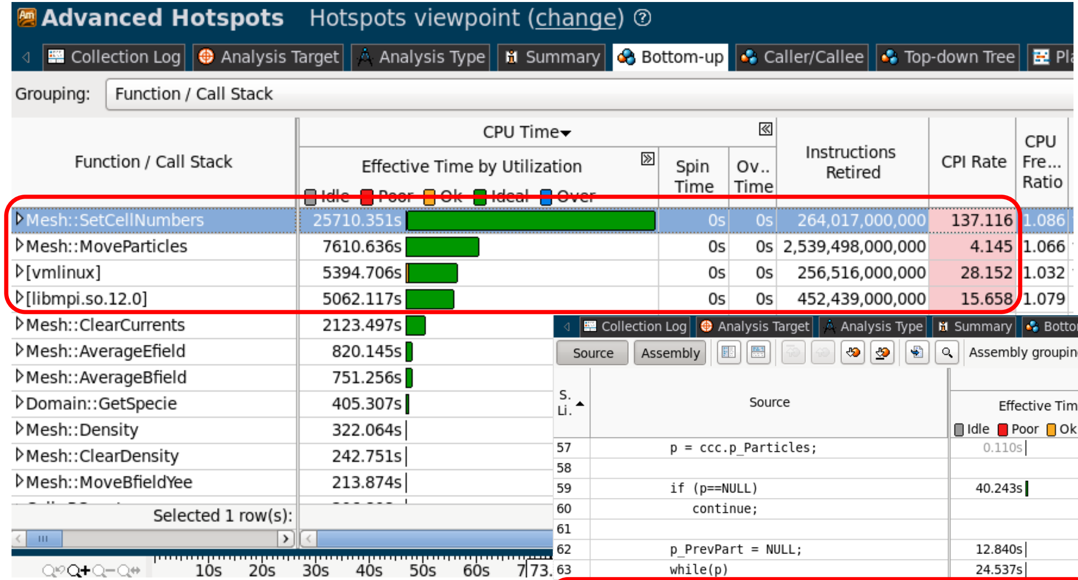
Using workload v2d-test3.ini and v2d-test1.ini

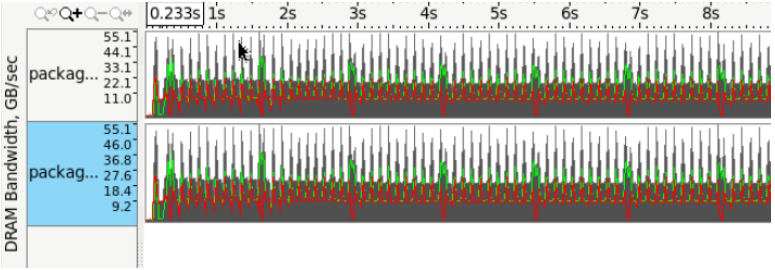


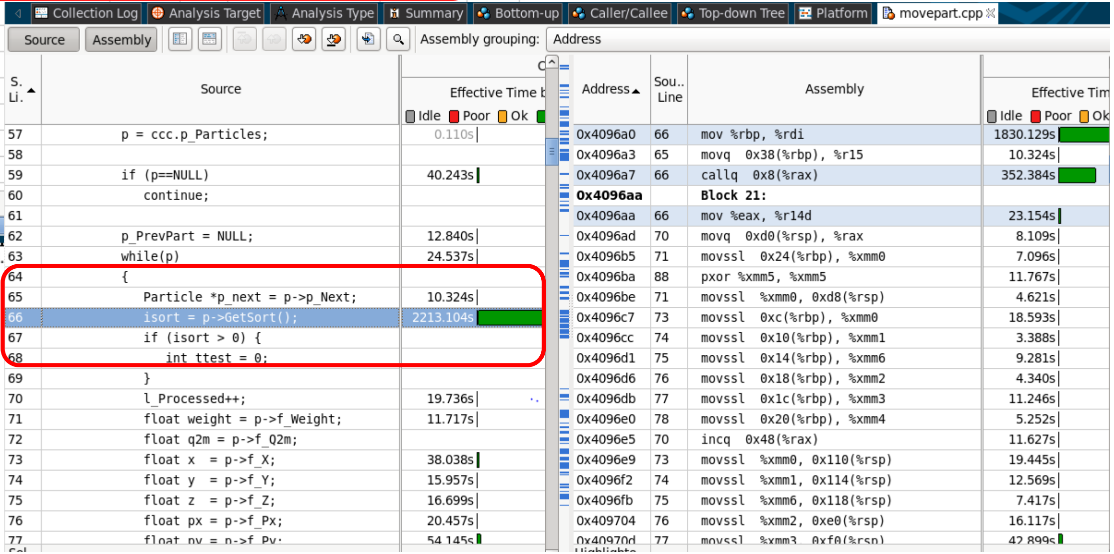
For the workload v2d-test1.ini, the overhead is MPI communication. So the scalability is bad.

For the workload v2d-test3.ini, the load balance is not bad. And MPI overhead is not high compared to the real computation time.

VLPL-S Analysis using Intel® VTune™ Amplifier XE







Agenda

- VLPL-S性能分析
- VLPL-S性能优化
- 总结

Optimizations Policies for VLPL-S

Optimization been tried

- Remove the unnecessary computation and memory access
- Improve cache hit rate by prefetch
- Avoid unnecessary precision conversion of constant and function call
- Inter-procedural Optimization
- Improve function call efficiency by removing the virtual function call
- Vectorization
- Make good usage of MCDRAM on KNL

Optimization being tried

- Data restructure, make sure all of list in the unit-stride array
- Multi-Thread Support, such as OpenMP, TBB, etc.
- Modify MPI communication mode to unblocked
- Parallel I/O if needed

Optimization #1: Remove the unnecessary operation from algorithm

- Avoiding unnecessary memory access by removing the function SetCellNumbers

Function	Module	CPU Time
Mesh::SetCellNumbers	v2d_sjtu.e.cpu.org	6419.121s
Mesh::MoveParticles	v2d_sjtu.e.cpu.org	3188.829s
PMPI_Recv	libmpi.so.12	1041.520s
Mesh::ClearCurrents	v2d_sjtu.e.cpu.org	355.378s
Cell::PCount	v2d_sjtu.e.cpu.org	143.645s
[Others]	N/A*	714.528s

*N/A is applied to non-summable metrics.

Function	Module	CPU Time
Mesh::MoveParticles	v2d_sjtu.e.cpu.opt1	2374.591s
Mesh::ClearCurrents	v2d_sjtu.e.cpu.opt1	265.071s
MPID_nem_mpich_blocking_recv	libmpi.so.12.0	155.462s
Cell::PCount	v2d_sjtu.e.cpu.opt1	105.145s
Mesh::Density	v2d_sjtu.e.cpu.opt1	82.010s
[Others]	N/A*	603.095s

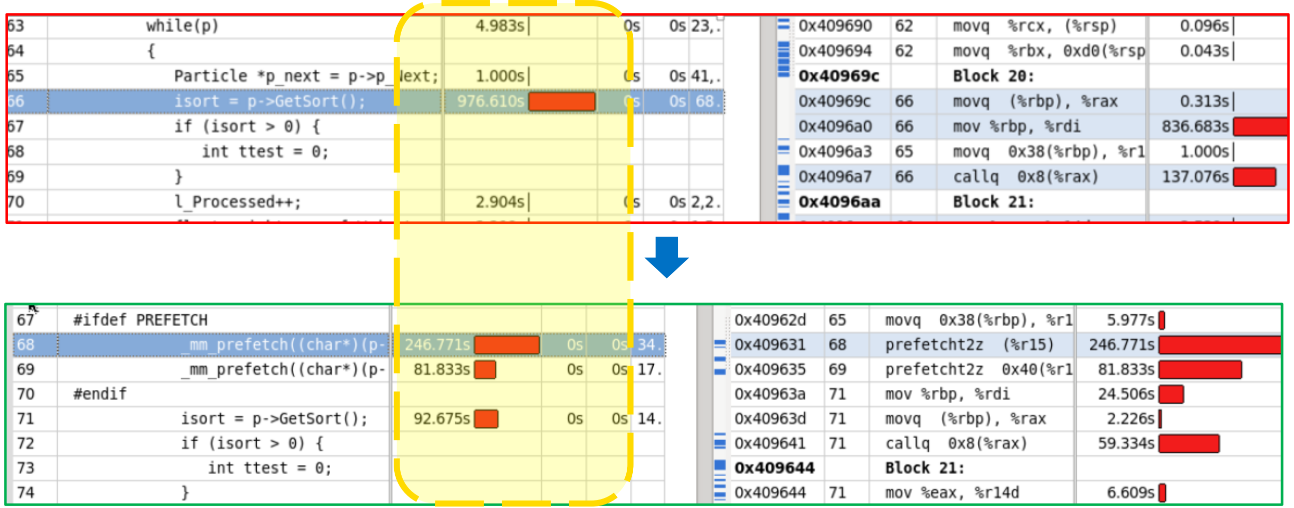
*N/A is applied to non-summable metrics.

void Mesh::SetCellNumbers() {				
long n=0;				
for (n=0; n<l_sizeXYZ; n++) {	0.4%	0.0%	0.0.	42.040s
p_CellArray[n].l_N = n;	0.0%	0.0%	0.0.	4.640s
Particle *p = p_CellArray[n].p_Particles;	0.1%	0.0%	0.0.	8.591s
while (p) {	50.2%	0.0%	0.0.	5955.001s
p->l_Cell = n;	0.3%	0.0%	0.0.	41.180s
p = p->p_Next;	3.1%	0.0%	0.0.	367.669s
}				
}				
}				

v2d-test3.ini
BDW 36P: 2.49x
KNL 256P: 2.33x

Optimization #2: Improving cache hit rate by prefetch

- The list structure is not friendly to improving memory access. Cache miss occurs. Using _mm_prefetch to optimize the code.



v2d-test3.ini
BDW 36P: 1.33x
KNL 256P: 1.08x

Optimization #3: Using the right precision and avoiding precision conversion

- Using the right precision function
- Avoiding constant precision conversion

	Idle / Poor / Ok / Ideal / Over	Co
Mesh::MoveParticles	2017.734s	
Mesh::ClearCurrents	353.255s	
Cell::PCount	152.350s	
Mesh::Density	125.546s	
MPID_nem_mpich_blocki	0s	12
Mesh::AverageBfield	103.088s	
Mesh::AverageEfield	100.167s	
Mesh::MoveBfieldYee	54.763s	

	Idle / Poor / Ok / Ideal / Over	Co
Mesh::MoveParticles	1834.868s	
Mesh::ClearCurrents	355.908s	
Cell::PCount	148.600s	
Mesh::Density	123.755s	
MPID_nem_mpich_blocki	0s	114
Mesh::AverageBfield	98.813s	
Mesh::AverageEfield	96.075s	
Mesh::MoveBfieldYee	54.596s	

For example:

```
gamma = 1./sqrt(1. + px*px + py*py + pz*pz);
```

```
gamma = (float)1./sqrtf((float)1. + px*px + py*py + pz*pz);
```

v2d-test3.ini
BDW 36P: 1.03x
KNL 256P: 1.07x

Optimization #4: Inter-procedural Optimization

- Use the compiler option "-ipo"

Function	Module	CPU Time
Mesh::MoveParticles	v2d_sjtu.e.knl.opt3	10294.500s
[libmpi.so.12.0]	libmpi.so.12.0	1850.735s
[Outside any known module]	[Unknown]	1660.055s
Domain::GetSpecie	v2d_sjtu.e.knl.opt3	894.516s
Mesh::ClearCurrents	v2d_sjtu.e.knl.opt3	724.534s
[Others]	N/A*	4437.190s

*N/A is applied to non-summable metrics.

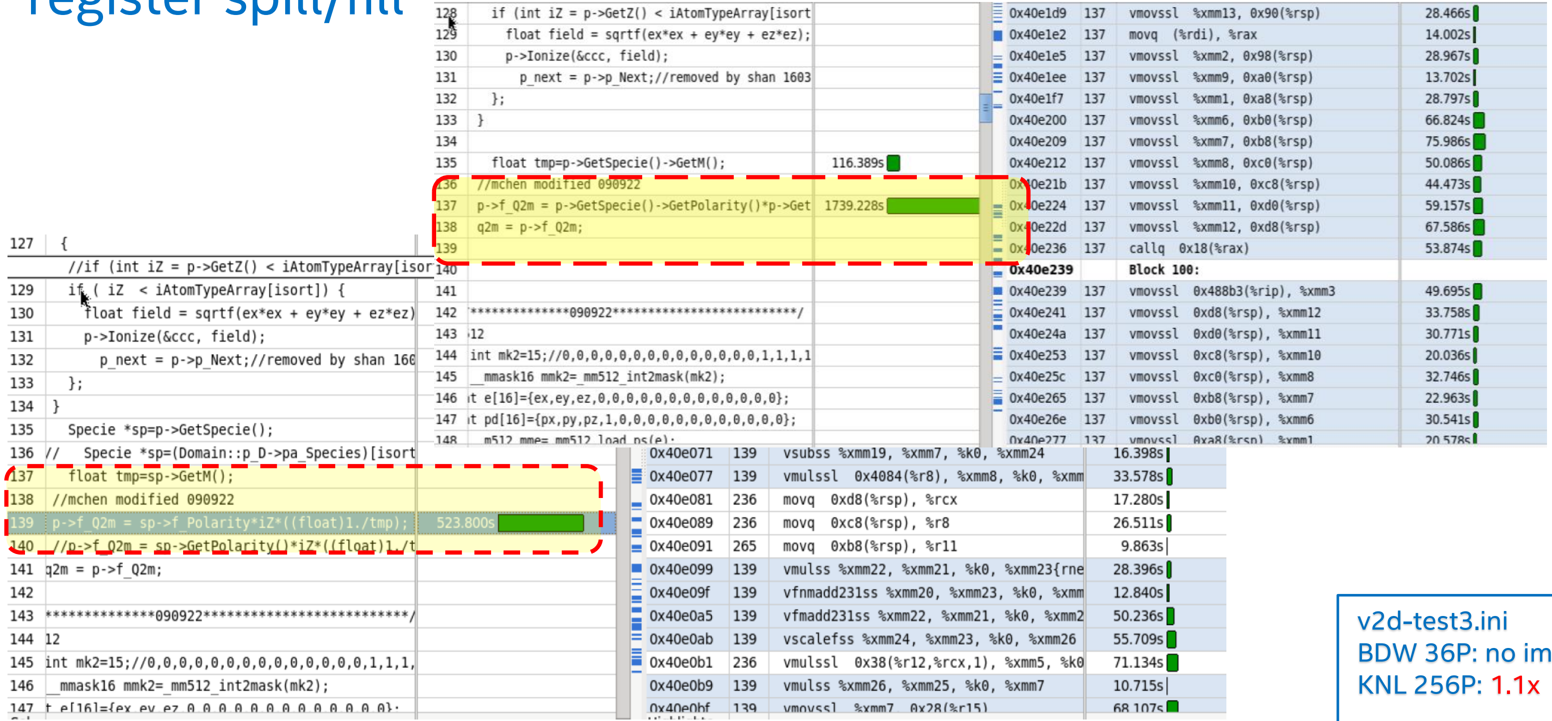
Function	Module	CPU Time
Mesh::MoveParticles	v2d_sjtu.e.knl.opt4	10411.731s
[libmpi.so.12.0]	libmpi.so.12.0	1660.837s
[Outside any known module]	[Unknown]	1562.960s
Mesh::ClearCurrents	v2d_sjtu.e.knl.opt4	733.294s
Mesh::Density	v2d_sjtu.e.knl.opt4	441.139s
[Others]	N/A*	3039.401s

*N/A is applied to non-summable metrics.

v2d-test3.ini
BDW 36P: 1.02x
KNL 256P: 1.09x

Optimization #5: Removing Virtual Function Call

- Removing Virtual Function call to reduce overhead of the redundant register spill/fill



v2d-test3.ini
BDW 36P: no improvment
KNL 256P: 1.1x

For BDW, there is no obvious improvement.

Optimization #6: Vectorization

- Achieve vectorization by fetching the data from list and coping them to arrays. Because the data structure is list and memory access is random which is not good for vectorization

File Edit View Search Terminal Help

Vectorization code

```
        float mdx[VL], mdy[VL], mdz[VL];
        float mxtmp[VL], mytmp[VL], mztmp[VL];
        float mdjx[VL], mdjy[VL], mdjz[VL];
        float mgamma[VL], mgamma_r[VL];
        while(p)
        {
          //fetch
          for(int of=0;of<VL;of++)
          {
            if (p==NULL) {pp[of]=NULL;break;}
#ifdef PREFETCH
            if (p->p_Next!=NULL) {
              _mm_prefetch((char*)(p->p_Next), _MM_HINT_T2) ;// T0,T1,T2,NTA
              _mm_prefetch((char*)(p->p_Next)+64, _MM_HINT_T2) ;// T0,T1,T2,NTA
            }
#endif
            l_Processed++;
            pp[of]       = p;
            px[of]       = (pp[of])->f_X;
            py[of]       = p->f_Y;
            pz[of]       = p->f_Z;
            ppx[of]      = p->f_Px;
            ppy[of]      = p->f_Py;
            ppz[of]      = p->f_Pz;
            pweight[of]  = p->f_Weight;
            pq2m[of]     = p->f_Q2m;
            piz[of]      = (pp[of])->GetZ();
            pisort[of]   = p->GetSort();
            p = p->p_Next;
          }

          // calc ...
          for(int of=0;of<VL;of++){
            maxc[of] = 1.-px[of];
            maxp[of] = px[of];
            mayc[of] = 1.-py[of];
```

Original code

```
          p_PrevPart = NULL;
          Particle *p_next = p->p_Next;
          while(p)
          {
            p_next = p->p_Next;
#ifdef PREFETCH
            _mm_prefetch((char*)(p->p_Next), _MM_HINT_T2) ;// T0,T1,T2,NTA
            _mm_prefetch((char*)(p->p_Next)+64, _MM_HINT_T2) ;// T0,T1,T2,NTA
#endif
            l_Processed++;
            float x  = p->f_X;
            float y  = p->f_Y;
            float z  = p->f_Z;
            float px = p->f_Px;
            float py = p->f_Py;
            float pz = p->f_Pz;
            float weight = p->f_Weight;
            float q2m = p->f_Q2m;

            float axc = 1.-x;
            float axp = x;
            float ayc = 1.-y;
            float ayp = y;
            float azc = 1.;

            float axcyczc = (axc*ayc);
            float axpyczc = (axp*ayc);
            float axcypzc = (axc*ayp);
            float axpypzc = (axp*ayp);


            float ex =
            ayc*ccc.f_Ex + ayp*cpc.f_Ex;

            float ey =
            axp*pcc.f_Ey + axc*ccc.f_Ey;
```

v2d-test3.ini
BDW 36P: 0.99x
KNL 256P:1.05x

Optimization #6: Vectorization

Optimization#6

83	for(int of=0;of<VL;of++)	5.726s	0s	0s	22,.
84	{				
85	if (p==NULL) {pp[of]=NULL;break;	1.038s	0s	0s	1,7.
86	#ifdef PREFETCH				
87	if (p->p_Next!=NULL) {	898.851s	0s	0s	56.
88	_mm_prefetch((char*)(p->p_Next	9.002s	0s	0s	14,.
89	_mm_prefetch((char*)(p->p_Next	92.392s	0s	0s	13.
90	}				
91	#endif				

Optimization#5

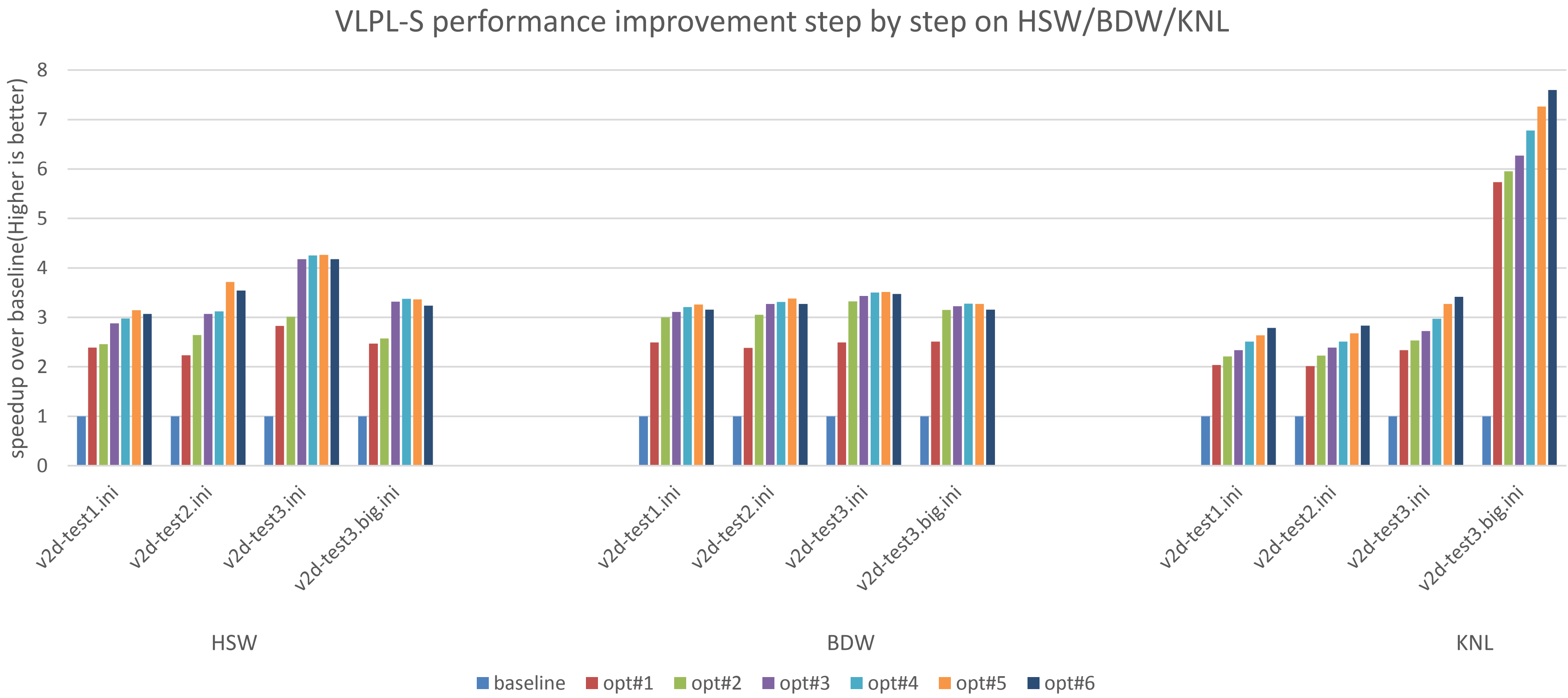
70	#ifdef PREFETCH				
71	_mm_prefetch((char*)(p->p_Next),_M	494.959s	0s	0s	63.
72	_mm_prefetch((char*)(p->p_Next)+64	85.295s	0s	0s	62,.
73	#endif				

The performance on HSW/BDW became **worse** than optimization #5.Because for optimization 6, the prefetch latency hasn't been hidden well after memory copy and vectorization can't bring enough benefit on BDW.

v2d-test3.ini
BDW 36P: 0.99x
KNL 256P:1.05x

Speedup over baseline step by step

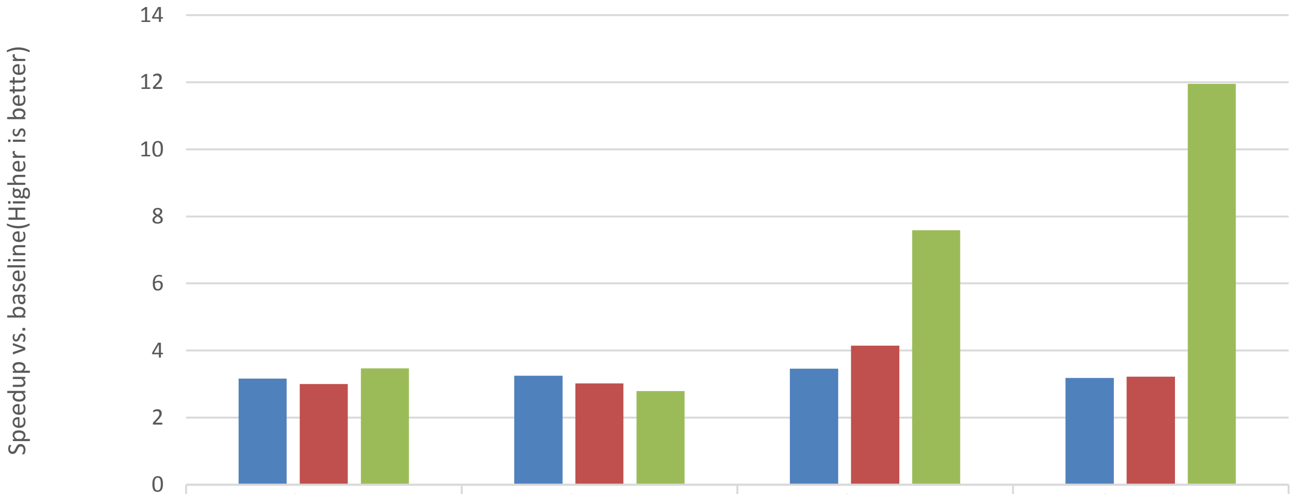
KNL baseline is the performance of Quadrant/Flat(MCDRAM only) 256 P



Speedup from Optimizations

Baseline is the performance of original code and on KNL ,it's the same binary as Xeon, no option of "-xMIC-AVX512" and just DDR only+Quadrant

VLPL-S Performance on Xeon and Xeon Phi



	v2d-test1.ini	v2d-test2.ini	v2d-test3.ini	v2d-test3.big.ini
2S E5-2699v3	3.157361485	3.247367664	3.457238359	3.18171316
2S E5-2697v4	3.003887017	3.023319493	4.142607174	3.215542522
Xeon Phi 7210	3.469315141	2.785975074	7.583134175	11.95739404

2S E5-2699v3 2S E5-2697v4 Xeon Phi 7210

Summary

- Most of optimization policies work on both of Xeon and Xeon Phi
- AVX512 will bring benefit on KNL if the code is vectorized
- High Scalability, high memory access efficiency and vectorization are very important
- MCDRAM is very helpful to performance on KNL for the application of high memory bandwidth
- Intel® Parallel Studio XE plays an important role in the performance analysis and optimization.

intel®

What’s Inside: Intel® Parallel Studio XE 2016

Intel® Parallel Studio XE 2016 Composer Edition 	Intel® Parallel Studio XE 2016 Professional Edition 	Intel® Parallel Studio XE 2016 Cluster Edition 
Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Data Analytics Acceleration Library (DAAL) Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Data Analytics Acceleration Library (DAAL) Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Data Analytics Acceleration Library (DAAL) Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*
	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE Intel® MPI Library Intel® Trace Analyzer and Collector
Bundle or Add-on: Rogue Wave IMSL* Library	Add-on: Rogue Wave IMSL* Library	Add-on: Rogue Wave IMSL* Library

Additional configurations including, floating and academic, are available at: http://intel.ly/perf-tools