

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

| | | 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 |

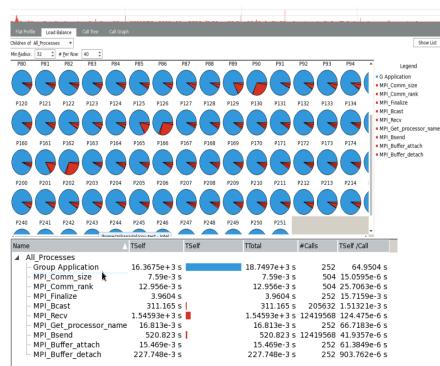
Machine Configurations

| Processor | E5-2699v3 | E5-2697v4 | Xeon Phi 7210 | Xeon Phi 7250 |
|--------------------------------|---|--------------------------------------|--------------------------------------|--------------------------------------|
| Stepping | 2 | 1(BO) | 1 (BO) | 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.01.01.0 008.021120151325 | GRRFSDP1.86B.027 1.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) | |

(inte

VLPL-S Analysis using Intel® Trace Analyzer and Collector

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



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



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

5

VLPL-S Analysis using Intel[®] VTune[™] Amplifier XE

| rouping: Function / Call Stack | | | | | packag | 44.1 33.1 22.1 | | |
|--------------------------------|----------------|-----------------------|---|------------------------|----------------------|----------------------|---|--|
| | | CPU Time v | | CPU | | 11.0 | - Tekterhethethetherenenthen | ununteenentteenenteene |
| Function / Call Stack | Effective Tin | ne by Utilization | Spin Ov Instructions | CPI Rate Fre | DRAM Bandwidth, | 55.1 46.0 | 1 6 6 9 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 | ի վեր եւ երելի հեղ երեր են, վերել |
| | Bidle Poor BOI | , | Time Time | Ratio | P packag | 36.8 | 🗉 a 🔒 a bha a bha a bha bha bha bha bha bha b | LAB LABET CALATERET CALATER |
| 4esh::SetCellNumbers | 25710.351s | | 0s 0s 264,017,000,00 | 0 137.116 1.086 | Bar | 18.4 | | an kan a kan kan kan kan kan kan kan kan |
| lesh::MoveParticles | 7610.636s | | 0s 0s 2,539,498,000,00 | | W | 9.2 | Which will show the second s | |
| | | | | | AN C | | | |
| /mlinux] | 5394.706s | | 0s 0s 256,516,000,00 | | | | | |
| ibmpi.so.12.0] | 5062.117s | | 0s 0s 452,439,000,00 | 0 15.658 1.079 | | | | |
| 1esh::ClearCurrents | 2123.497s | < ₩ | Collection Log Analysis Target 🗛 Analysis Typ | e 📓 Summary 🏼 🕹 Bottor | n-up 🗳 Caller/C | allee | 🕏 Top-down Tree 🛛 🖼 Platform 🛛 🚹 movep | art.cpp 🛛 |
| 1esh::AverageEfield | 820.145s | Sour | ce Assembly 🔢 🚟 💿 💿 🧶 😣 | Assembly grouping | g: Address | | | |
| lesh::AverageBfield | 751.256s | | | | d^_ | | | |
| omain::GetSpecie | 405.307s | S. | Source | Effective Time | Address | Sou | Assembly | Effective Tim |
| esh::Density | 322.064s | U. * | | Idle Poor Ok | | Line | | Idle Poor Ok |
| lesh::ClearDensity | | 57 | <pre>p = ccc.p Particles;</pre> | 0.1105 | 0x4096a0 | 66 | mov %rbp, %rdi | 1830.1295 |
| | 242.751s | 58 | · · · · · | | 0x4096a3 | 65 | movq 0x38(%rbp), %r15 | 10.3245 |
| lesh::MoveBfieldYee | 213.874s | 59 | if (p==NULL) | 40.243s | 0x4096a7 | | callq 0x8(%rax) | 352.384s |
| Selected 1 row(s) | | 60 | continue; | | 0x4096aa | | Block 21: | |
| | | 61 | | | 0x4096aa | | mov %eax, %r14d | 23.1545 |
| 10- 20- | 30s 40s 50s | 62 | <pre>p_PrevPart = NULL;</pre> | 12.840s | - 0x4096ad | | movq 0xd0(%rsp), %rax | 8.1095 |
| QPQ+Q-Q# 10s 20s | 30s 40s 50s | 60s 7 73.63 | while(p) | 24.537s | 0x4096b5 0x4096ba | | movssl 0x24(%rbp), %xmm0 pxor %xmm5, %xmm5 | 7.096s 11.767s |
| | | 65 | <pre>Particle *p next = p->p Next;</pre> | 10.324s | 0x4096ba | | movssl %xmm0, 0xd8(%rsp) | 4.6215 |
| | | 65 | isort = p->GetSort(); | 2213.1045 | 0x4096c7 | | movssl 0xc(%rbp), %xmm0 | 18.5935 |
| | | 67 | if (isort > 0) { | 2213.1045 | 0x4096cc | | movssl 0x10(%rbp), %xmm1 | 3.388s |
| | | 68 | int ttest = 0; | | 0x4096d1 | | movssl 0x14(%rbp), %xmm6 | 9.281s |
| | | 69 | } | | 0x4096d6 | 76 | movssl 0x18(%rbp), %xmm2 | 4.340s |
| | | 70 | l_Processed++; | 19.736s | • = 0x4096db | 77 | movssl 0x1c(%rbp), %xmm3 | 11.246s |
| | | 71 | <pre>float weight = p->f_Weight;</pre> | 11.717s | 0x4096e0 | 78 | movssl 0x20(%rbp), %xmm4 | 5.252s |
| | | 72 | <pre>float q2m = p->f_Q2m;</pre> | | 0x4096e5 | | incq 0x48(%rax) | 11.627s |
| | | 73 | float x = p->f_X; | 38.038s | 0x4096e9 | | movssl %xmm0, 0x110(%rsp) | 19.445s |
| | | 74 | float y = p->f_Y; | 15.957s | 0x4096f2 | | movssl %xmm1, 0x114(%rsp) | 12.569s |
| | | 75 | float $z = p - f_Z$; | 16.699s | 0x4096fb | | movssl %xmm6, 0x118(%rsp) | 7.417s |
| | | 76 | float px = p->f_Px; | 20.457s | 0x409704 | 76 | movssl %xmm2, 0xe0(%rsp) | 16.117s |

(inte

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 unitstride 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 |

| Function | Module | CPU Time |
|------------------------------|---------------------|-----------|
| Mesh::MoveParticles | v2d_sjtu.e.cpu.opt1 | 2374.591s |
| Mesh::ClearCurrents | v2d_sjtu.e.cpu.opt1 | 265.0719 |
| MPID_nem_mpich_blocking_recv | libmpi.so.12.0 | 155.462s |
| Cell::PCount | v2d_sjtu.e.cpu.opt1 | 105.1459 |
| Mesh::Density | v2d_sjtu.e.cpu.opt1 | 82.0109 |
| [Others] | N/A* | 603.0959 |

| | <pre>void Mesh::SetCellNumbers() {</pre> | | | | |
|------------|--|-------|------|------|-----------|
| 339 | long n=0; | | | | |
| 340 | for (n=0; n <l_sizexyz; n++)="" td="" {<=""><td>0.4%</td><td>0.0%</td><td>0.0.</td><td>42.040s</td></l_sizexyz;> | 0.4% | 0.0% | 0.0. | 42.040s |
| 341 | <pre>p_CellArray[n].l_N = n;</pre> | 0.0% | 0.0% | 0.0. | 4.640s |
| 342 | <pre>Particle *p = p_CellArray[n].p_Particles;</pre> | 0.1% | 0.0% | 0.0. | 8.591s |
| 343 | while (p) { | 50.2% | 0.0% | 0.0. | 5955.001s |
| 344 | p->l_Cell = n; | 0.3% | 0.0% | 0.0. | 41.180s |
| 345 | $p = p - p_Next;$ | 3.1% | 0.0% | 0.0. | 367.669s |
| 346 347 | } | | | | |
| 347 | } | | | | |
| 348 | } | | | | |

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.

| 53 | while(p) | | 4.983s | | 0 | s (| 0s 23, . | = | 0x409690 | 62 | movq | %rcx, | (%rsp) | 0.096s |
|----------------------|---|-------|-------------|----------|----------|-----|----------|--|--------------------------------------|---------------------------------|--|-----------------------------------|---|---|
| 54 | { | | | | | | | | 0x409694 | 62 | movq | %rbx, | 0xd0(%rsp | 0.043s |
| 55 | Particle *p_next = p->p_ | lext; | 1.000s | | 0 | s (| 0s 41,. | = | 0x409690 | : | Block | 20: | | |
| | | | 976.610s | | <u>ا</u> | s (| 0s 68. | | 0x40969c | 66 | movq | (%rbp) |), %rax | 0.313s |
| 57 | if (isort > 0) { | | | | | | | | 0x4096a0 | 66 | mov % | srbp, %r | rdi | 836.683s |
| 58 | <pre>int ttest = 0;</pre> | | | | | | | = | 0x4096a3 | 65 | movq | 0x38(% | %rbp), %r1 | 1.000s |
| 59 | } | | | | | | | | 0x4096a7 | 66 | callo | 0x8(% | %rax) | 137.076s |
| 70 | l_Processed++; | | 2.904s | | (| s (| 0s 2,2. | = | 0x4096aa | 1 | Block | 21: | | |
| | | | | | | ◀ | ŀ | | | | | | | |
| 67 | #ifdef PREFETCH | | | | | | | 0x40962 | d 65 | mova | 0x38(%) | rbp). % | r1 5.977 | s |
| | <pre>#ifdef PREFETCH</pre> | 246.7 | 715 | 0s | 05 | 34. | | 0x40962 0x40963 | | movq prefe | 0x38(%) tcht2z | 1 | _ | - |
| | | | 715 | Os Os | | 34. | | | 1 68 | prefe | | (%r15) | 246.771 | s |
| 68 | _mm_prefetch((char*)(p- | | | | | | | 0x40963 | 1 68 5 69 | prefe prefe | tcht2z | (%r15) 0x40(%r | 246.771 | s Is |
| 69 | _mm_prefetch((char*)(p- _mm_prefetch((char*)(p- | 81.8 | | | 0s | | | 0x40963 0x40963 | 1 68 5 69 a 71 | prefe prefe mov % | tcht2z tcht2z | (%r15) 0x40(% di | 246.771 r1 81.833 | s Caracteria s Is Caracteria s |
| 68 69 70 | _mm_prefetch((char*)(p- _mm_prefetch((char*)(p- #endif | 81.8 | 33s 📕 | 0s | 0s | 17. | | 0x40963 0x40963 0x40963 | 1 68 5 69 a 71 d 71 | prefe prefe mov % movq | tcht2z tcht2z rbp, %ro | (%r15) 0x40(%n di , %rax | 246.771 r1 81.833 24.506 | s |
| 68 69 70 71 | <pre>mm_prefetch((char*)(p- _mm_prefetch((char*)(p- #endif isort = p->GetSort();</pre> | 81.8 | 33s 📕 | 0s | 0s | 17. | | 0x40963 0x40963 0x40963 0x40963 | 1 68 5 69 a 71 d 71 1 71 | prefe prefe mov % movq | tcht2z tcht2z rbp, %rc (%rbp) 0x8(%n | (%r15) 0x40(%n di , %rax | 246.771 r1 81.833 24.506 2.226 | s |

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 ۲

| 📕 Idle 📕 Poor 📕 Ok 📕 Ideal 📕 Over | |
|-----------------------------------|---|
| 2017.734s | |
| 353.255s | |
| 152.350s | |
| 125.546s | |
| 0s | 12 |
| 103.088s | |
| 100.167s | |
| 54.763s | |
| | 2017.734s 353.255s 152.350s 125.546s 0s 103.088s 100.167s |

| | 🔲 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 |
|-----------------------------|---------------------|------------|
| <u>Mesh::MoveParticles</u> | v2d_sjtu.e.knl.opt4 | 10411.731s |
| [libmpi.so.12.0] | libmpi.so.12.0 | 1660.837s |
| [Outside any krigwn 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 |

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

Optimization #5: Removing Virtual Function Call

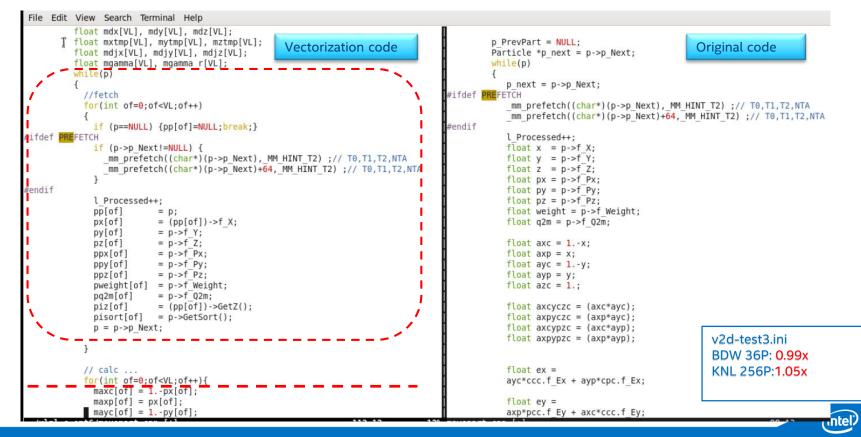
Removing Virtual Function call to reduce overhead of the redundant register spill/fill if (int i7 = n_Set7() < iAtomTyneArray[isort 0x40e1d9 137 vmovss] %xmm13 Ax9A(%rsp) 28 4665

| | 128 | 1f (1nt 1Z = p->Ge | etZ() | < 1AtomTyp | peArray | [isort | = | 0x40e1d9 | 137 | vmovssl | %xmm13, 0x90(%rsp) | | 28.466s | |
|---|-------------------|---|-------|------------------|---------|--------------------------|----------|-------------|-----|--------------------|----------------------|------|------------------|-------|
| | 129 | float field = so | qrtf(| ex*ex + ey* | *ey + e | z*ez); | | 0x40e1e2 | 137 | movq (% | rdi), %rax | | 14.002s | |
| | 130 | p->Ionize(&ccc, | fiel | d); | | | = | 0x40e1e5 | 137 | vmovssl | %xmm2, 0x98(%rsp) | | 28.967s | |
| | 131 | p_next = p->p | Next | ;//removed | by sha | n 1603 | | 0x40elee | 137 | vmovssl | %xmm9, 0xa0(%rsp) | | 13.702s | |
| | 132 | }; | | | | | - | 0x40e1f7 | 137 | vmovssl | %xmm1, 0xa8(%rsp) | | 28.797s | |
| | 133 | } | | | | | _ | 0x40e200 | 137 | vmovssl | %xmm6, 0xb0(%rsp) | | 66.824s | |
| | 134 | | | | | | | 0x40e209 | 137 | vmovssl | %xmm7, 0xb8(%rsp) | | 75.986s | |
| | 135 | <pre>float tmp=p->GetSp</pre> | pecie | ()->GetM(); | ; | 116.3895 | | 0x40e212 | 137 | vmovssl | %xmm8, 0xc0(%rsp) | | 50.086s | |
| | 136 | //mchen modified 090 | 9922 | | | | | 0x10e21b | 137 | vmovssl | %xmm10, 0xc8(%rsp) | | 44.473s | |
| | 137 | p->f_Q2m = p->GetSpe | ecie(|)->GetPola | rity()* | p->Get 1739.228s | | 0x40e224 | 137 | vmovssl | %xmm11, 0xd0(%rsp) | | 59.157s | |
| | 138 | q2m = p->f_Q2m; | | | | | | 0x40e22d | 137 | vmovssl | %xmm12, 0xd8(%rsp) | | 67.586s | |
| 127 { | 139 | | | | | | - | 0x40e236 | 137 | callq 0 | x18(%rax) | | 53.874s | |
| <pre>//if (int iZ = p->GetZ() < iAtomTypeArray[is</pre> | or ₁₄₀ | | _ | | | | | 0x40e239 | | Block 10 | θ: | | | |
| <pre>129 if (iZ < iAtomTypeArray[isort]) {</pre> | 141 | | | | | | | 0x40e239 | 137 | vmovssl | 0x488b3(%rip), %xmm3 | | 49.695s | |
| <pre>130 Thoat field = sqrtf(ex*ex + ey*ey + ez*ez)</pre> | 142 | *************************************** | **** | ******** | ****** | ****/ | | 0x40e241 | 137 | vmovssl | 0xd8(%rsp), %xmm12 | | 33.758s | |
| <pre>131 p->Ionize(&ccc, field);</pre> | 143 | 12 | | | | | | 0x40e24a | 137 | vmovssl | 0xd0(%rsp), %xmm11 | | 30.771s | |
| <pre>132 p next = p->p Next;//removed by shan 160</pre> | 144 | int mk2=15;//0,0,0,0 | ,θ,Θ, | 0,0,0,0,0,0 | 9,0,0,1 | ,1,1,1 | - | 0x40e253 | 137 | vmovssl | 0xc8(%rsp), %xmm10 | | 20.036s | |
| 133 }; | 145 | mmask16 mmk2=_mm512 | 2_int | 2mask(mk2); | ; | | = | 0x40e25c | 137 | vmovssl | 0xc0(%rsp), %xmm8 | | 32.746s | |
| 134 } | 146 | t e[16]={ex,ey,ez,0,0 | 9,0,0 | ,0,0,0,0,0 | ,0,0,0, | 0}; | | 0x40e265 | 137 | vmovssl | 0xb8(%rsp), %xmm7 | | 22.963s | |
| <pre>135 Specie *sp=p->GetSpecie();</pre> | 147 | it pd[16]={px,py,pz,1 | ,0,0, | 0,0,0,0,0,0 | 9,0,0,0 | ,0}; | | 0x40e26e | 137 | | 0xb0(%rsp), %xmm6 | | 30.541s | |
| 136 // Specie *sp=(Domain::p D->pa Species)[isort | 148 | m512 mme= mm512 loa | | | 139 | vsubss %xmm19, %xmm7, %k | 0 9 | 0x40e277 | | vmovssl 16.398s | Axa8(%rsn) %xmm1 | | 20 5785 | |
| <pre>130 // Specie sp-(bomain: p =>pa_species)[130 // 137 float tmp=sp->GetM();</pre> | - | | | 0x40e071 | 139 | vmulssl 0x4084(%r8), %x | | | | 33.578s | | | | |
| 137 Troat timp-sp->deth(); 138 //mchen modified 090922 | | | | 0x40e077 | 236 | mova Oxd8(%rsp), %rcx | , 111110 | - ∿KU, ∿KII | | | | | | |
| | 500 | | . E. | | | 1 1 1 1 1 | | | _ | 17.280s | | | | |
| <pre>139 p->f_02m = sp->f_Polarity*iZ*((float)1./tmp);</pre> | | .8005 | | 0x40e089 | 236 | movq 0xc8(%rsp), %r8 | | | | 26.511s | | | | |
| <pre>140 //p->f_02m = sp->GetPolarity()*iZ*((float)1./t</pre> | | / | | 0x40e091 | 265 | movq 0xb8(%rsp), %r11 | | | | 9.863s | | | | |
| 141 q2m = p->f_02m; | | | | 0x40e099 | 139 | vmulss %xmm22, %xmm21, % | | - | | 28.396s | | | | |
| 142 | | | | 0x40e09f | 139 | vfnmadd231ss %xmm20, %xm | 1m23, | %k0, %xm | m | 12.840s | | | | |
| 143 ************************************ | | | | 0x40e0a5 | 139 | vfmadd231ss %xmm22, %xmm | 121, | %k0, %xmm | 2 | 50.236s | | V20 | d-test3.ini | |
| 144 12 | | | | 0x40e0ab | 139 | vscalefss %xmm24, %xmm23 | 3, % | (0, %xmm26 | | 55.709s 📕 | | | | |
| 145 int mk2=15;//0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1, | | | | 0x40e0b1 | 236 | vmulssl 0x38(%r12,%rcx, | 1), | %xmm5, %k | 0 | 71.134s | | I RD | W 36P: no improv | /ment |
| 146 mmask16 mmk2= mm512 int2mask(mk2); | | | | 0x40e0b9 | 139 | vmulss %xmm26, %xmm25, % | sk0, | %xmm7 | | 10.715s | | I KN | L 256P: 1.1x | |
| 147 t e[16]={ex ev ez 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | | | | 0x40e0hf | 139 | vmovssl %xmm7 Ax28(%r1 | 5) | | | 68 1075 | | | | |
| n-1 | | | | 11 - L P - L L - | | | | | | | | | | |

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



Optimization #6: Vectorization

Optimization#6

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

Optimization#5

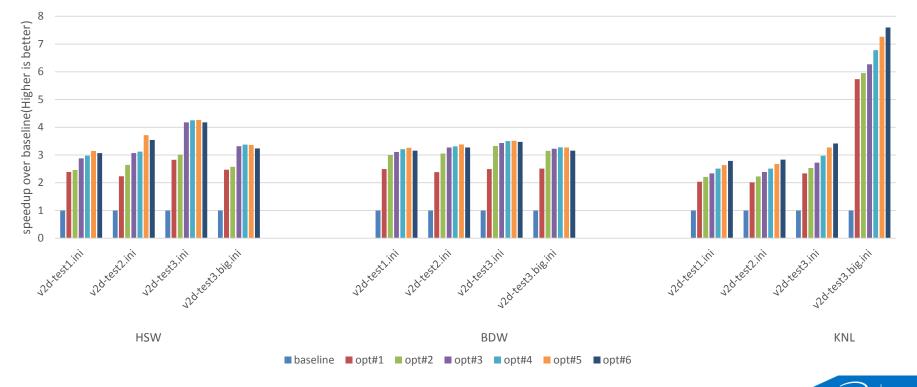
| 70 | #ifdef PREFETCH | | | | |
|----|------------------------------------|----------|----|----|------|
| 71 | _mm_prefetch((char*)(p->p_Next),_M | 494.959s | 0s | 0s | 63. |
| 72 | _mm_prefetch((char*)(p->p_Next)+64 | 85.295s | 0s | 0s | 62,. |
| 73 | #endif | | | | |

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

VLPL-S performance improvement step by step on HSW/BDW/KNL

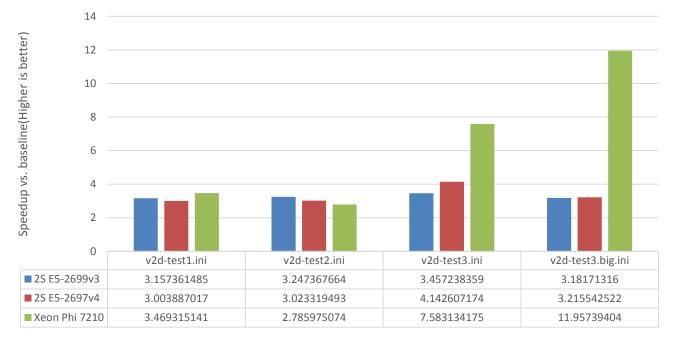


(int

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



■ 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.



What's Inside: Intel[®] Parallel Studio XE 2016

| Intel® Parallel Studio XE 2016 | Intel® Parallel Studio XE 2016 Professional Edition | Intel® Parallel Studio XE 2016 Cluster Edition |
|---|--|---|
| Intel® C++ Compiler | Intel® C++ Compiler | Intel® C++ Compiler |
| Intel® Fortran Compiler | Intel® Fortran Compiler | Intel® Fortran Compiler |
| Intel® Threading Building Blocks | Intel® Threading Building Blocks | Intel® Threading Building Blocks |
| Intel® Integrated Performance Primitives | Intel® Integrated Performance Primitives | Intel® Integrated Performance Primitives |
| Intel® Data Analytics Acceleration Library (DAAL) | Intel® Data Analytics Acceleration Library (DAAL) | Intel® Data Analytics Acceleration Library (DAAL) |
| Intel® Math Kernel Library | Intel® Math Kernel Library | Intel® Math Kernel Library |
| Intel® Cilk™ Plus | Intel® Cilk™ Plus | Intel® Cilk™ Plus |
| Intel® OpenMP* | Intel® OpenMP* | 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: | Add-on: | Add-on: |
| Rogue Wave IMSL* Library | Rogue Wave IMSL* Library | Rogue Wave IMSL* Library |

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