Code modernization a practical approach

HPC-TNT-1.2 fall 2016





Annie Cuyt 🗌 Stefan Becuwe 🗌 Franky Backeljauw 🗌 Kurt Lust 🗌 Engelbert Tijskens





· A practical approach

- \cdot To build efficient applications
- \cdot To understand performance issues

To maximize code flexibility To minimize coding efforts





- Some computer architecture concepts related to performance and machine limits
 - · Levels of parallelism
 - Memory on modern CPUs

- \rightarrow peak performance
- \rightarrow bandwidth, latency

- Example1 [toy problem]
 - · Atomic system interacting through a Lennard-Jones potential
 - Monte Carlo setting
- Example 2
 - · Atomic system interacting through Lennard-Jones potential
 - Molecular dynamics setting
- $\cdot~$ Optimizing code \Leftrightarrow optimizing data access
 - Spatial sorting using space filling curve
- · Choosing a programming language



A little terminology

- A (compute) node in a cluster is basically a PC without all the peripheral devices
- Nodes are connected through a fast network: interconnect
- Every node has several **socket**s, each of which contains a processor
- Every processor contains many **core**s
- Each core can simultaneously execute one task: thread
 - · Simultaneous multi-threading : more than one thread per core
 - \cdot Usually switched off on clusters
- · Thread executes a sequential stream of instructions







































1. Several nodes can cooperate on a task

- · **Distributed** memory parallellization
- Nodes communicate information via interconnect
 - \cdot Typically using MPI: Message Passing Interface
- 2. Several cores can cooperate on a task
 - Shared memory parallellization
 - · Cores communicate information via memory
 - · Often using OpenMP, but also MPI, ...



3. A single core may exploit **pipelining** and **vectorisation** to execute instructions in parallel





- Instructions break down in micro-instructions
- Each micro-instruction uses a distinct part of the hardware
 - 1. Instruction fetch (IF)
 - 2. Instruction Decode (ID)
 - 3. Execution (EX)
 - 4. Memory Read/Write (MEM)
 - 5. Result Writeback (WB)























Load 2 operands in register Execute 1 add instruction Store 1 result



Register 1
$$A_i$$
 A_j A_k A_l +vectorRegister 2 B_i B_j B_k B_l =vectorRegister 3 C_i C_j C_k C_l

Load 2x4 operands in register Execute 1 vadd instruction Store 1x4 results



Register 1
$$A_i$$
 A_j A_k A_l +vectorRegister 2 B_i B_j B_k B_l =vectorRegister 3 C_i C_j C_k C_l

Load 2x4 operands in register Execute 1 vadd instruction Store 1x4 results

Single Instruction Multiple Data (SIMD)





Potentially 4x faster [*if the loads and stores can be executed fast enough*]

Load 2x4 operands in register Execute 1 vadd instruction Store 1x4 results

Single Instruction Multiple Data (SIMD)



- Fused instructions: Fused Multiply-Add executes y = a*x+b (in vector mode in one cycle
- · Vector register width on Hopper is 256 bits
 - · 8 single precision numbers
 - · 4 double precision numbers



Levels of parallellism

- 3 levels of parallellism:
 - \cdot Intra-core: pipelining and SIMD
 - Multi-core: shared memory

· Multi-node: distributed memory



Levels of parallellism

- 3 levels of parallellism:
 - \cdot Intra-core: pipelining and SIMD
 - Multi-core: shared memory

Multi-node: distributed memory

- Who does the work?
 - \cdot Compiler
 - but it appreciates/needs your help
 - · You and the compiler
 - OpenMP = directives
 - · Relatively simple
 - · You only
 - MPI
 - · Harder



1. Monte Carlo setting:

- compute the energy of configurations
- ensemble averages
- no (individual) forces, accelerations or velocities
- no time integration

2. Molecular Dynamics setting:

- \cdot compute the time evolution of a collection of atoms
- · Individual forces, accelerations, velocities
- [time integration]



- Compute the potential energy of a system of *N* atoms: $\sum_{i < j} V_{LJ}(r_{ij})$
- · Doing this for all atoms is a $O(N^2)$ approach
 - we do it only for a single atom just for the purpose of illustrating the behavior of the hardware (toy problem)
 - $\cdot \sum_{j} V_{LJ}(r_{ij})$
- · Techniques to reduce to O(N) will be discussed in MD setting
- We will only consider **single core** performance
 - \cdot That is the first thing to optimize anyway
 - · Already complex enough for a single lecture
 - · SIMD and pipelining are the only level of parallelism



• Lennard-Jones potential (neglecting constants)





Lennard-Jones potential (sequential code)

```
double VLJ0( double r ) {
    return 1./pow(r,12) - 1./pow(r,6);
}
                                                            // 18.0 x slower
double VLJ1( double r )_{
    return std::pow(r,-12) - std::pow(r,-6);
                                                            // 14.9 x slower
}
double VLJ2( double r ) {
    double tmp = std::pow(r,-6);
    return tmp*(tmp-1.0);
                                                             // 7.8 x slower
}
double VLJ3( double r ) {
    double tmp = 1.0/(r*r*r*r*r*r);
    return tmp*(tmp-1.0);
                                                            // 1.01 x slower
}
double VLJ( Real t r ) {
    double rr = 1./r;
    rr *= rr;
    double rr6 = rr*rr*rr;
    return rr6*(rr6-1);
}
                                                               // 1 x slower
```



```
double VLJ( Real_t r ) {
    double rr = 1./r;
    rr *= rr;
    double rr6 = rr*rr*rr;
    return rr6*(rr6-1);
} // 1 x slower
double VLJ( Real_t r2 ) {
    double rr = 1./r2;
    // rr *= rr;
    double rr6 = rr*rr*rr;
    return rr6*(rr6-1);
} // avoid one sqrt per function call (to compute the distance r)
```



Cost of instructions

- cheap instructions
- Rather expensive
- Expensive
- Very expensive

+,-,* ~1 cycle / ~10-20 cycles sqrt ~35 cycles trigonometric/ logarithmic/ exponential functions ~100-200 cycles

- \cdot Things get better if pipelining can be exploited
- Relative cost remains



- \cdot x0, y0, z0 : coordinates of our central atom
- x1[1:m], y1[1:m], z1[1:m] : coordinates of m neighbouring atoms
- Let m=512, 1012, ..., 2²⁹~0.5*10⁹
- \cdot surround by outer loop iterating 2²⁹/m times
 - every m-case executes 2^{29} evaluations of $V_{LJ}(r^2)$
- · Variations
 - \cdot Loop over all m neighbouring atoms contiguously
 - Structure of arrays (SoA) : x x x ... y y y ... z z z ...
 - Array of structures (AoS) : x y z ... x y z ... x y z ...
 - Pick atoms in the arrays x1, y1, z1 with a random permutation



enddo

1	х				у				z			
2		x				У				Z		
3			x				У				Z	



MC - Contiguous - AoS

enddo

1	х	у	Z									
2				x	У	Z						
3							x	У	z			



MC - Random access

```
integer :: m ! Number of neighbour atoms
real(wp) :: x0,y0,z0, p(3*m)
```

integer :: j(m) ! random permutation of [1:m]

```
! random access
do ik=1,k
    do im=1,m
        r2 = (p(j(im))-x0)**2 +(p(j(im)+m)-y0)**2 +(p(j(im)+2*m)-z0)**2
        v = v + lj_pot2(r2)
        enddo
```

enddo

1		x			у			Z	
2	x			У			Z		
3			x			У			Z




















- Which factors influence performance of a code?
- Machine limits



- Maximum # floating point operations per second
- \cdot For a single core the peak performance =
 - \cdot 2*8 instructions per cycle in SP
 - \cdot 2*4 instructions per cycle in DP
 - $\cdot\,$ The 2 comes from the fused multiply and add
 - The 8, resp. 4 come from the vector register width
- Peak performance per node
 - · (1 cycle = 1/clock_frequency)
 - Assuming 1 hardware thread per core:
 - · (#cores=20) * 2*8(SP) * (f=2.8Ghz) = 896 Gflop/s
 - · (#cores=20) * 2*4(DP) * (f=2.8Ghz) = 448 Gflop/s



- \cdot Peak performance is not the only limiting factor...
- $\cdot\,$ It is not the most common limiting factor
- · Instructions operate on data, ...
- · Data resides in memory
- Accessing data takes time (and energy)
 - Data has to be moved from memory to cpu register before it can be processed
- Peak performance has increased much faster than the speed at which data can be moved between memory and cpu



Memory bandwidth

- The maximum number of bytes that can be moved per second between main memory and the cores
- · Hopper
 - · 92-110 GB/s (varies depending on read:write ratio)

Memory latency

- The number of cycles (or the time) needed to fetch a single item from main memory
- Hopper
 - $\cdot \sim 180$ cycles (within socket)
 - ~350 cycles (across sockets)



Performance

- \cdot Code is **compute bound** if
 - The cpu can execute its compute instructions without having to wait for data
 - · The limit is the theoretical peak performance
 - [Used to be the common case not any more]
- $\cdot\,$ Code is **memory bound** if
 - · A considerable amount of cycles is spent waiting for data
 - · Too much data requested:
 - **Bandwidth saturation** = machine limit
 - Too distant data requested:
 - · If data is not in the cache: latency penalty
 - · Latency problem = machine limit
 - [most common situation]



















































- Optimizing code was about organizing compute instructions
 - · Pretty straightforward: less compute cycles is less cputime
 - · Algorithmic complexity was important guideline
- Optimizing code is optimizing data access
 - \cdot To keep the processor busy doing useful stuff
 - Algorithmic complexity is no longer a guarantee for optimal performance
 - $\cdot\,$ E.g. linear search (as in a map) often faster than binary or other search algorithms, also sorting
 - For large N low order complexity wins, but hardware caching takes an early lead
- · Understanding how memory works is necessary
- Experimenting and measuring is necessary



On-chip (E5-2680 v2)	ALU		memory size speed
	Registers: ~1kB per core	0 cycles	
	L1 Cache: 32 kB per core	~1 cycles	
	L2 Cache: 256 kB per core	~10 cycles	
	L3 Cache: 25 MB per socket	~50 cycles	
Off-chip	DRAM: 64-256 GB per node	~200 cycles	



- Memory is not fetched on a per item basis
- But in chunks called cache lines
 - typically 64 Bytes long
 - \cdot 16 single precision items
 - \cdot 8 double precision items



- **Linear search** of array A[i], i=1..n
 - A[1] is not in cache, wait time before cache line is loaded, dram latency (~200 cycles) and before item A[1] can be examined
 - Once the cache line is loaded, A[2..16] are also in L1 cache and are ready to be examined without delay
 - Hardware recognizes your loop over the array and keeps loading next (or previous) cache lines into the L1 cache, so that the delay is vanishing
 - Depending on how much work it takes to examine each item, as soon as item A[16], the next cache line A[17:32] may have been loaded already or not
 - \cdot In any case, the wait time is now less than the dram latency (~200 cycles)
 - · The limitation becomes memory bandwidth of the machine



• **Binary search** of array A[i], i=1..n

- A[n/2] is not in cache, wait time before cache line is loaded, dram latency
- \cdot Next item needed is A[n/4] or A[3n/4], which is not in the cache, dram latency hits you again
- In fact, the dram latency keeps on hitting you until the search range is reduced to one or two cache lines,
- You do only one examination/dram latency, as opposed to 16/dram latency in linear search.



- · Instructions are also data stored in memory
- Branching instructions can cause cache misses too!
 Instruction cache misses
- · Avoid unpredictable branches in loops



- code::dive conference 2014 Scott Meyers: Cpu Caches and Why You Care
- https://www.youtube.com/watch?v=WDIkqP4JbkE















Experiment











Flops per second

do ik=1,k $r^{2} = (p(im)-x0)^{**2}$ $+(p(m+im)-y0)^{**2}$ $+(p(2^{*}m+im)-z0)^{**2}$ $r = lj_pot2(r)$ $r^{2i} = 1.0d0/r^{2}$ $r^{6i} = r^{2i^{*}r^{2}}}}}}}}}}}}}}}}}}}}}}}$

· ! Contiguous access, SoA: p=[xxx...yyy...zzz...]

- 14 flops * 2^{29} iterations in 1.2 s = 6.26 10⁹ flops/s
- peak performance:

1*1*4*2.8 GHz = 11.2 Gcycles/s = 11.2 Gflops/s

• We are running at 55.9 % of peak performance

















•	do ik=1,k			
•	do im=1,m			
•	$r2 = (p(im)-x0)^{**}2$			
	+(p(m+im)-y0)**2			
	+(p(2*m+im)-z0)**2	•	3-, 2+, 3* · 3 DP	
	! $r = lj_pot2(r)$			
•	r2i = 1.0d0/r2		1/	
•	rr6i = r2i*r2i*r2i;		2*	
•	lj_pot2 = 4.0d0*rr6*(rr6-1.0d0);	•	2*, 1-	
•	enddo	•		-
•	enddo		14 flops · 24 B	

- 24 B $*2^{29}$ iterations in 1.2 s = 10.7 GB/s
- · Bandwidth measured by Intel mlc:
 - 109 GB/s for 10 threads (all reads)

! Contiguous access. SoA: p=[xxx...vvv...777...]

- \cdot 10.9 GB/s for 1 thread
- \cdot We are running at maximal bandwidth
 - Bandwidth saturation
















































- \cdot For simple cases
 - \cdot a back of the envelope calculation like this
 - and an understanding of how memory works
 can guide you to more efficient code
- For real cases we need something more sophisticated



- · Intel Advisor xe
 - \cdot Vectorization and threading
- · Intel VTune Analyzer xe
 - · Data access and cpu utilization
- Intel Inspector
 - · Thread performance analysis (OpenMP, Intel TBB)
- Intel Cluster Inspector
 - · MPI process performance analysis



- Intel Advisor xe
 - Vectorization and threading
- Intel VTune Analyzer xe
 - Data access and cpu utilization
- Intel Inspector
- Thread performance analysis (OpenMP, Intel TBB)
 Intel Cluster Fiospattater session
 - MPI process performance analysis



- https://software.intel.com/en-us/get-started-with-advisor
- https://software.intel.com/en-us/get-started-with-vtune











00	X Create a Project	
Project name:	test	
Location:	werpen/201/vsc20170/intel/advixe/projects	Browse
	Create Project	Cancel



	X test - Project Properties							
Analysis Target Binary/Sym	nbol Search Source Search							
✓ ≧ Survey Analysis Type	Survey Launch Application							
Survey Hotspots	Specify and configure the application executable (targe more details.	et) to analyze. Press F1 for		• • •	X Select File			
🦞 Suitability Analysi	No application executable (target) file specific	ad		📝 🖣 🗁 vsc2017	0 data neon ppmd01 bin			
🗢 🗁 Refinement Analysi:								
Dependencies An	Application:	▼ Browse		<u>P</u> laces	Name	▼ Size	Modified ^	
🖇 Memory Access F	Application parameters:	▼ Modify		🔍 Search	🛅 Advisor		Friday	
				Recently Used	🛅 ppmd01.dSYM		09/05/2016	
	Use application directory as working directory			🗁 ppmd01	🗋 md.optrpt	81.3 KB	09:36	
	Working directory:	 Browse 		🚞 vsc20170	📄 ppmd01	792.0 KE	11:31	
				🔯 File System	ppmd01.optrpt	21.7 KB	11:31	
	User-defined environment variables:				prec.optrpt	1.2 KB	11:31	
		Modiry			util_random.optrpt	8.2 KB	11:31	
	Child application:		lcome		🗋 wprec.optrpt	1.2 KB	11:31	
	Analyze loops that reside in non-executed code pat	hs	orization (
	Advanced		int project:					
	Colloction data limit MR: 100		v Project					
			Ducie at					
	Resume collection after ms 10		<u>in Project</u>					
		OK Cancel	<u>en Result</u>				Ŧ	
		Rec	ent Projects	- <u>h</u>			All files (*) 🔻	
						💥 <u>C</u> ance	Den <u>Den</u>	
			L.					



	X test - Project Properties
Analysis Target Binary/Sym	abol Search Source Search
 ✓	Survey Launch Application Specify and configure the application executable (target) to analyze. Press F1 for more details.
✓ ≧ Refinement Analysi	Application: data/neon/ppmd01/bin/ppmd01
	Application parameters: Modify
	☑ Use application directory as working directory
	Working directory: /user/antwerpen/201/vsc20170 Browse
	User-defined environment variables: Modify
	Child application:
	Analyze loops that reside in non-executed code paths
	OK



•••	X /user/antwerpen/201/vsc20170/intel/advixe/projects/test - Intel Advisor
<u>F</u> ile View Help	
և 🖆 📾 🎭 i 🖉 🔤 🕉 T	» 🏇 🎋 🖞 🛯 🗃 🗍 😨
Welcome e000 🐹	▼
Vectorization Threading Workflow Workflow	Where should I add vectorization and/ INTEL ADVISOR XE 2016 Summary Survey Report Refinement Reports Annotation Report
OFF Batch mode	🔥 No Data
1. Survey Target	To collect data about your application's performance, compile your application with Release build settings and run <u>Survey</u> analysis.
Collect	
Collect 🖿 📘	
Mark Loops for Deeper An Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis. There are no marked loops	
2.1 Check Dependencies	
Collect	
In the second	
2.2 Check Memory Access P	
Collect	
• Nothing to analyze	



							-			
ation	Threading	Where should I add vector	izat	ion and/or t	hreading	parallelisn	n? 💿			
DW	Workflow	Elapsed time: 2162.89s Ovectorize	d	Not Vectorized 🖉	FILTER: All	Module 🗾 All S	Sources 💽 Lo	op 🗾 All Threads		` _ •
atch mod	e	🌪 Summary 🔮 Survey Report 🔌 Refinen	nent R	eports 💧 Annotat	ion Report					
atch mou		😞 \land Higher instruction set arc	hite	cture (ISA) av	/ailable					
y Target	8	Your application was compiled using an	ISA lov	ver than the ISA ava	ailable on this ma	achine. Consider	recompiling your ap	plication: on this machine		
		using the highest ISA available - use th	e xHo	st option, or on the	original machine	for a higher ISA	- use the x option.			
	<u>P_</u>									
		Function Call Sites and Loops		Vector Issues	Self Time s	Total Time	Type	Why No Vectorization?	Vectoriz	ed Lo
Trip Cou	ints	runction can sites and Loops		Vector issues	Sell fille +	local nine	Type	wity No vectorization:	Vect	Effici
		⊕© [loop in fill_uniform_integer at util_rando		💡 1 Data type c	15.657s	15.657s 📩	Vectorized (Body)		SSE2	~100
t 🖿 🧾		⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Inefficient m	1.210s	1.210s	Vectorized (Body)		SSE2	~100
		⊕ [loop in fill_uniform_real at util_random.f9			1.000s	1.000s	Vectorized (Body)		SSE2	~100
ops for D	Deeper An	호 [loop in experiment1 at ppmd01.f90:73]		💡 2 Assumed de	0.981s	0.981s	Scalar	vector dependence		
ps in the S	Survey	±♂ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized us	0.976s	0.976s	Vectorized (Body)			~100
Depender	ncies and/or	⊕o [loop in fill_uniform_real at util_random.f9	· 🗆	💡 1 Serialized us	0.359s(0.359s (Vectorized (Body)		SSE	~100
iccess Pat	Cerris	🗉 🖱 [loop in ppmd01 at ppmd01.f90:26]			0.000s (20.183s	Scalar	loop with function cal		



ation		🖉 Where should I add vectori	zat	ion and/or t	hreading	parallelisn	n? 🗖			
DW				Not Vectorized 🖉	FILTER: All	Module 🗾 All	Sources 💽 Lo	op 🗾 All Threads		
t ch m e de		P Summary Survey Report 🔅 Refineme		eports 💧 Annotai	tion Report					
y Target		A Higher instruction set arch Your application was compiled using an IS using the highest ISA available - use the	A lov A lov	cture (ISA) av ver than the ISA ava st option, or on the	vailable ailable on this m original machine	achine. Consider e for a higher ISA	recompiling your ap - use the x option.	pplication: on this machine	2	
		Function Call Sites and Loops		Vector Issues	Self Time 🗸	Total Time	Type	Why No Vectorization?	Vectoria	zed Lo
Trip Coun	nts		-						Vect	Effici
		⊕o [loop in fill_uniform_integer at util_rando		💡 l Data type c	15.657s	15.657s	Vectorized (Body)		SSE2	
t 🖿 📐		⊕o [loop in fill_uniform_real at util_random.f9]		🔋 1 Inefficient m	1.210s	1.210s	Vectorized (Body)		SSE2	
		⊕o [loop in fill_uniform_real at util_random.f9			1.000s	1.000s	Vectorized (Body)		SSE2	
ops for De	eeper An	☑♂ [loop in experiment1 at ppmd01.f90:73]		2 Assumed de	0.981s	0.981s	Scalar	vector dependence		
		⊕ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized us	0.976s	0.976st	Vectorized (Body)			
		⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized us	0.359s (Vectorized (Body)		SSE	
		⊡ර් [loop in ppmd01 at ppmd01.f90:26]			0.000s (20.183s	Scalar	loop with function cal		

 \cdot Add compiler option –xHost













	Vectorization	Threading	🚇 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
Ш	Workflow	Workflow	Elapsed time: 220.37s Ucctorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
	OFF Batch mod	0	🍄 Summary 🛭 📽 Survey Report 🔌 Refineme	ent Rep	orts 💧 Annotation Report													\leq
	on a butter not		Duration Call Citize and Leaves		Verster lesses	Call Times	Tabel Times	7	Miles Ma Marchanian bing 2	Vectoria	zed Loops		()	Instruction Set Analysis		5	J 🖉	J Destina
Ш	1. Survey Target	•	Function Call Sites and Loops	0 ¥	vector issues	sell lime	iotal fime	iype	why no vectorization?	Vect	Efficiency	Gain	VL.	. Traits	Data Types	Number of Vector Registers	Advanced	Location
Ш		-	⊕o [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Ш		<u>b_</u>			1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In.	5		util_random.f90:5
			∃O [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
	nd Trip Cou	ints	⊕o [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
						0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
	Collect		⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
	/		±0 [loop in experiment1 at ppmd01.f			0.100st	0.100s(Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Ш	Mark Loops for D	Deeper An	∃oop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
1	Select loops in the S	Survey Report	🗉 🖉 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99
ш	Access Patterns an:	alveis																



• Run Intel Advisor again

Vectorization	Threading	🚇 Where should I add vectori:	zatio	on and/or threading paralle	lism? 💿											INTE	L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
Off Batch m	ode	🌪 Summary 👹 Survey Report 🔌 Refineme	nt Rep	orts 💧 Annotation Report													
Daten III	Jue					an . 1 an	-		Vectori	zed Loops		D	Instruction Set Analysis		Di		B
1. Survey Targe	et	Function Call Sites and Loops	0 -	vector issues	Self lime	iotal lime	iype	why no vectorization?	Vect	Efficiency	Gain	VL .	. Traits	Data Types	Number of Vector Registers	Advanced	Location
		±♂ [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect		⊕o [loop in fill_uniform_integer at util_rando		🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		±o [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip C	ounts	⊡oop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	_	⊕o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	<u>L</u>	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±0 [loop in experiment1 at ppmd01.f			0.100st	0.100s(Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for	r Deeper An	∃oop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
 Select loops in the for Dependencies 	e Survey Report and/or Memory	🗟 🖉 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	I Scalar	inner loop was alread		_			Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

Function Call Sites and Loops	å -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
Iloop in experiment1 at ppmd01.f.			4.034s 📩	4.034s 📩	Vectorized (B	
⊕ [loop in fill_uniform_integer at util_rando .	🗆	🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
⊕o [loop in fill_uniform_real at util_random.f9	🗆	I Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
⊞o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕o [loop in fill_uniform_real at util_random.f9	🗆	💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)	
⊕o [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9	🗆	💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)	
🖻 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread



• Run Intel Advisor again

Vectorization	Threading	🛯 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
orr Batch mod	0	🌪 Summary 🚭 Survey Report 🔅 Refineme	nt Rep	orts 💧 Annotation Report													
Batchillou	ie .					w . 1 w	-		Vectori	zed Loops		(b)	Instruction Set Analysis		D		· · · · · · · · · · · · · · · · ·
1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	lotal lime	iype	why no vectorization?	Vect	Efficiency	Gain	. VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
		±© [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect Di m		⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		±© [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Cou	ints	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	1	±© [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	1	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±© [loop in experiment1 at ppmd01.f			0.100si	0.100s (Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for D	Deeper An	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the S for Dependencies ar	Survey Report nd/or Memory	🖻 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

Function Call Sites and Loops	å -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
⊕ <mark>©</mark> [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕ <mark>⊘</mark> [loop in fill_uniform_real at util_random.f9		9 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
±© [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s(Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s(Vectorized (Body)	
🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread



• Run Intel Advisor again

		Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												FL ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Vectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	- Loops	📕 All Th	reads	•								Q
Off Batch me	ode ⁰	🍨 Summary 🛭 📽 Survey Report 🚯 Refineme	nt Rep	orts 💧 Annotation Report													
on Daten int	Jue						_		Vectori	zed Loops		D	Instruction Set Analysis		2	1	20
1. Survey Targe	et [°]	Function Call Sites and Loops	ð -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?	Vect	Efficiency	Gain	. VL .	. Traits	Data Types	Number of Vector Registers	Advanced	Location
	_	±© [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect		±© [loop in fill_uniform_integer at util_rando		🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Co	ounts	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	_	± 🖸 [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sI	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	<u></u>	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±0 [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for	Deeper An	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the for Dependencies	e Survey Report and/or Memory	🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	s inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

Random access

Function Call Sites and Loops	å -	Vector sues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
① [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access patterns	0.170s	0.170st	Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9			0.156s	0.156st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕ of [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
🛛 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



• Run Intel Advisor again

		Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												FL ADVISOR XE 2016			
Workflow	Workflow	Elapsed time: 220.37s Vectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	- Loops	📕 All Th	reads	•							٩				
Off Batch me	ode ⁰	🍨 Summary 🛭 📽 Survey Report 🚯 Refineme	nmary 🙀 Survey Report 🍺 Refinement Reports 💧 Annotation Report																	
on Daten int	Jue						_		Vectori	zed Loops		D	Instruction Set Analysis		2	1	20			
1. Survey Targe	et [°]	Function Call Sites and Loops	ð -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?	Vect	Efficiency	Gain	. VL .	. Traits	Data Types	Number of Vector Registers	Advanced	Location			
	_	±© [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134			
Collect		±© [loop in fill_uniform_integer at util_rando		🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5			
		±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3			
nd Trip Co	ounts	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3			
	_	± 🖸 [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sI	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117			
Collect	<u></u>	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1			
		±0 [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1			
Mark Loops for	Deeper An	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1			
Select loops in the for Dependencies	e Survey Report and/or Memory	🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	s inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99			

Random access

	-					
Function Call Sites and Loops	å -	Vector sues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	□ 4.034s 💳	Vectorized (B	
⊞ o [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access patterns	0.170s	0.170st	Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9			0.156s	0.156st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕ of [loop in fill_uniform_real at util_random.f9]		I Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊕ of [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread



• Run Intel Advisor again

Vectorization	Threading	Where should I add vectorize	zatio	on and/or threading paralle	lism? 💿												EL ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Th	reads	•								٩
OFF Batch mod	۵ ⁰	🗣 Summary 🙀 Survey Report 🔌 Refinement Reports 👌 Annotation Report															
on Bacchinou							_		Vectori	zed Loops		D	Instruction Set Analysis		2	6	9
1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	lotal lime	iype	why no vectorization?	Vect	Efficiency	Gain .	. VL .	. Traits	Data Types	Number of Vector Registers	Advanced	Location
	-				4.034s	4.034s	Vectorized (Body		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect P _{il}	<u>e</u>	±© [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		⊕o [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Cou	ints	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	1	⊞O [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect		⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
V		± ^o [loop in experiment1 at ppmd01.f			0.100st	0.100s (Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for D	Deeper An	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the S for Dependencies ar	Survey Report nd/or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	c inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

List	of	hot	spots	
------	----	-----	-------	--

Random access

Function Call Sites and Loops	å -	Vector sues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
① [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
⊕ [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access patterns	0.170s	0.170st	Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9			0.156s	0.156st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕ of [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
🛛 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



Vectorization	Threading	🚇 Where should I add vectori	zatio	on and/or threading paralle	elism? 💿												EL ADVISOR XE 2016			
Workflow	Workflow	Elapsed time: 220.37s Ucctorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Th	reads	•							٩				
off Batch m	odo ⁰	🌳 Summary 🛭 🛠 Survey Report 🔅 Refineme	nmary 🕸 Survey Report 🔌 Refinement Reports 👌 Annotation Report																	
OFF Bacching	oue						_		Vectori	zed Loops		D	Instruction Set Analysis		2	1 6	ð 19			
1. Survey Targe	et	Function Call Sites and Loops	ð -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?	Vect	Efficiency	Gain .	VL .	Traits	Data Types	Number of Vector Registers	Advanced	Location			
					4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134			
Collect				🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5			
		⊕o [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3			
nd Trip C	ounts	⊞o [loop in fill_uniform_real at util_random.f9			0.156s	0.156sl	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3			
	_	⊞O [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117			
Collect	2	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1			
		∃ [©] [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1			
Mark Loops for	r Deeper An	∃O [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1			
Select loops in the for Dependencies	e Survey Report and/or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99			

\cdot List of hot s	ро	Random access	Contiguo	us-AoS		
Function Call Sites and Loops	å -	Vector sues	Self Time	Total Time	Туре	Why No Vectorization?
+©[loop in experiment1 at ppmd01.f			4.034s	□ 4.034s	Vectorized (B	
±♂[loop in fill_uniform_integer at util_rando		🔋 1 Data type concersions present	0.254s	0.254s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access patterns	0.170sl	0.170s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9			0.156st	0.156s	Vectorized (Body)	
±♂ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9]		9 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
+ 🖱 [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		9 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
অত [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread



	Vectorization	Threading	🚇 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿											INT	EL ADVISOR XE 2016		
	Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Th	reads	•							Q			
	orr Datch mod	10 ⁰	🌳 Summary 🛭 🛠 Survey Report 🔅 🔅 Refineme	imary 缓 Survey Report 🇿 Refinement Reports 💧 Annotation Report																
16	orr Bacch mod	ie .					m + 1 m ²	-		Vectori	zed Loops		D	Instruction Set Analysis		D	J	2		
	1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	iotal lime	iype	why no vectorization?	Vect	Efficiency	Gain	. VL .	Traits	Data Types	Number of Vector Registers	Advanced	Location		
			±♂ [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134		
	Collect 🖣 🖿 🛌		⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5		
1			⊕ [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3		
	nd Trip Cou	ints	⊕ [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3		
		1	⊕o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s I	Vectorized (Body		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117		
	Collect	1	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1		
			±© [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1		
	Mark Loops for D	Deeper An	⊕ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1		
	Select loops in the S for Dependencies ar	Survey Report nd/or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread		_			Unpacks	Float64	16		ppmd01.f90:99		
	Assess Dakkama and	- lucia																		

\cdot List of hot s	ро	Random access	Contiguou	us-AoS	Contiguous-So	A
Function Call Sites and Loops	å -	Vector sues	Self Time	Time	Туре	Why No Vectorization?
±@[loop in experiment1 at ppmd01.f			A _45	4.034s	Vectorized (B	
±© [loop in fill_uniform_integer at util_rando		🔋 1 Data type concersions present	0.254s	0.254s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access atterns	0.170st	0.170st	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156st	Vectorized (Body)	
±♂[loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
±o [loop in fill_uniform_real at util_random.f9		🔋 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
+ 🖱 [loop in experiment1 at ppmd01.f90:100]		-	0.100s	0.100s (Vectorized (Body)	
∃o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
ුර [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



		m When a chardel to del us start			1												
Vectorization	Threading	where should I add vectori	zatio	on and/or threading paralle	iism?												L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s O Vectorized	O Not	Vectorized FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
OFF Batch more	de ⁰	🗣 Summary 📽 Survey Report 🚯 Refinement Reports 💩 Annotation Report															
Duten mot							-		Vectori	zed Loops		5	Instruction Set Analysis		12	1	J 199
1. Survey Target		Function Call Sites and Loops	ð •	vector issues	Self lime	iotai lime	iype	why no vectorization?	Vect	Efficiency	Gain	VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
	-	±© [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect Di		±© [loop in fill_uniform_integer at util_rando		🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	. 5		util_random.f90:5
		± © [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Cou	unts	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	-	±© [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	J	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±© [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for I	Deeper An	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the for Dependencies a	Survey Report and/or Memory	🖻 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread		-			Unpacks	Float64	16		ppmd01.f90:99
Assess Dakkama an	- humin																

\cdot List of hot s	ро	ts Random access	Contiguou	ıs-AoS	Contiguous-So	A
Function Call Sites and Loops	å -	Vector sues	Self Time	Time	Туре	Why No Vectorization?
+©[loop in experiment1 at ppmd01.f			45	4.034s	Vectorized (B	
±© [loop in fill_uniform_integer at util_rando		🔋 1 Data type concersions present	0.254s	0.254s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access atterns	0.170s	0.170s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
±♂ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
± 🖱 [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
ාර [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



	Vectorization	Threading	🖴 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
	Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
	orr Datch mad		🍄 Summary 🔮 Survey Report 🔌 Refineme	nt Rep	orts 💧 Annotation Report													
	Batch mod	16					m	~		Vector	ized Loops		D	Instruction Set Analysis		D		
	1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	iotal lime	іуре	why no vectorization?	Vect	Efficiency	Gain	. VL .	Traits	Data Types	Number of Vector Registers	Advanced	Location
			⊕o [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
			⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	. 5		util_random.f90:5
10			⊕ [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
	nd Trip Cou	unts	⊕ [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
		1	⊕o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
		1	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
			±© [loop in experiment1 at ppmd01.f			0.100st	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
	Mark Loops for D	Deeper An	⊕ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
	Select loops in the S for Dependencies a	Survey Report nd/or Memory	ාර [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

\cdot List of hot s	ро	Random access	Contiguou	ıs-AoS	Contiguous-So	A
Function Call Sites and Loops	å -	Vector sues	Self Time	Time	Туре	Why No Vectorization?
±©[loop in experiment1 at ppmd01.f			A _45	4.034s	Vectorized (B	
±♂ [loop in fill_uniform_integer at util_rando		🔋 1 Data type concersions present	0.254s	0.254s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory accessed terns	0.170s	0.170st	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
±♂ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9]		9 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
+ 🖱 [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		I Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
অত [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



• Run Intel Advisor again

	Vectorization	Threading	🖴 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												EL ADVISOR XE 2016	
	Workflow	Workflow	Elapsed time: 220.37s O Vectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩	
	OFF Batch mod	10 ⁰	P Summary 📽 Survey Report 🤌 Refinement Reports 💧 Annotation Report																
16	on a bacch mou					e 10 m	m	~		Vectori	zed Loops		D	Instruction Set Analysis		B	3	3	
	1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	iotai lime	іуре	why No vectorization?	Vect	Efficiency	Gain .	. VL .	Traits	Data Types	Number of Vector Registers	Advanced	Location	
			± 🖱 [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134	
	Collect		±© [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	. 5		util_random.f90:5	
100			± o [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3	
	nd Trip Cou	ints	± o [loop in fill_uniform_real at util_random.f9			0.156s	0.156sl	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3	
		1	± o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117	
		1	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1	
			±© [loop in experiment1 at ppmd01.f			0.100s	0.100s (Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1	
	Mark Loops for D	Deeper An	± o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1	
	Select loops in the S	Survey Report	🗉 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000st 4.287s Scalar sinner loop was alread Un						Unpacks Float64 16 ppmd01.f90:99							
	or Dependencies al	nu/or memory																	

\cdot List of hot s	ро	Random access	Contiguou	is-AoS	Contiguous-So	A
Function Call Sites and Loops	å -	Vector sues	Self Time	Time	Туре	Why No Vectorization?
±©[loop in experiment1 at ppmd01.f			A _45	4.034s	Vectorized (B	
±© [loop in fill_uniform_integer at util_rando		🔋 1 Data type concersions present	0.254s	0.254s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access atterns	0.170s	0.170st	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156st	Vectorized (Body)	
±♂ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152st	Vectorized (Body)	
±♂ [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s(Vectorized (Body)	
অত [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread

Loop over m



• Run Intel Advisor again

Ĩ	Vectorization	Threading	🖴 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
-11	Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Th	eads	•								٩
	Off Ratch mod	1e ⁰	🍨 Summary 🛭 🛠 Survey Report 🏼 🍅 Refineme	nt Rep	orts 💧 Annotation Report													
	on accimica						w . 1 w	-		Vectori	zed Loops		D	Instruction Set Analysis		19	I I	· · · · · · · · · · · · · · · · ·
	1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	lotal lime	iype	why no vectorization?	Vect	Efficiency	Gain	. VL .	Traits	Data Types	Number of Vector Registers	Advanced	Location
- 11			⊞O [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
11	Collect				🛊 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
- 11			⊕o [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
	nd Trip Cou	ints	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156sl	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
		1	⊞O [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
		1	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
			±© [loop in experiment1 at ppmd01.f			0.100si	0.100s[Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
11	Mark Loops for D	Deeper An	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
11	Select loops in the 5 for Dependencies a	Survey Report nd/or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

Function Call Sites and Loops	Vectoriz	zed Loops		≫	Instruction Set Analysis	>>	Advanced	. ∣	
Function Call Sites and Loops	Vect	Efficiency	Gain	VL	Traits	Data Types	Number of Vector Registers	Advanced	
∎© [loop in experiment1 at ppmd01.f	AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		p
⊕ <mark>©</mark> [loop in fill_uniform_integer at util_rando	AVX		4.96x	4	Shifts; Type Conversions	Float64; In	5		ut
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9	AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	ut
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9	AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	ut
±♂ [loop in experiment1 at ppmd01.f90:117]	AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		р
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9	AVX	~100%	4.89x	4	Extracts	Float64	2		ut
±♂ [loop in experiment1 at ppmd01.f90:100]	AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		p
⊕ [loop in fill_uniform_real at util_random.f9]	AVX	~100%	5.48x	4		Float64	1		ut
🛛 🖱 [loop in ppmd01 at ppmd01.f90:99]					Unpacks	Float64	16		р



• Run Intel Advisor again

Vectorization	Threading	🖴 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	reads	•								٩
orr Batch me	ude ⁰	🌳 Summary 🛭 🛠 Survey Report 🔅 🔅 Refineme	nt Rep	orts 💧 Annotation Report													
Batchino	Jue					w	-		Vector	zed Loops		5	Instruction Set Analysis		2		B 19
1. Survey Targe	et 🛛	Function Call Sites and Loops	0 -	vector issues	Self Time	iotai lime	iype	why No vectorization?	Vect	Efficiency	Gain	VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
	_	±♂ [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect		⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
· · · ·		±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Co	ounts	⊕o [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	-				0.153s	0.153s	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect		⊕o [loop in fill_uniform_real at util_random.f9		I Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
_		±© [loop in experiment1 at ppmd01.f			0.100st	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for	Deeper An	⊕ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the for Dependencies	e Survey Report and/or Memory	🗟 🖉 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	I Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

Function Call Sites and Loops	\$ -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
⊕ <mark>©</mark> [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
🗄 🖱 [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170s	0.170st	Vectorized (Body)	
⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
🗄 🖱 [loop in fill_uniform_real at util_random.f9		9 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
±© [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊕♂ [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	🖬 inner loop was alread


· Run Intel Advisor again

Vectorization	Threading	🖴 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿											INTE	L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ucctorized	O Not	Vectorized 🔄 FILTER: All Module	All Sources	 Loops 	All Th	reads	•								٩
orr Batch mod	ها ا	🍄 Summary 🔮 Survey Report 🔌 Refineme	nt Rep	orts 💧 Annotation Report													
or Bacchillou	le			A Marshan Innun		w . 1 w			Vectorized Loops			Instruction Set Analysis		2		· · · · · · · · · · · · · · · · ·	
1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	lotal lime	іуре	why no vectorization?	Vect	Efficiency	Gain	VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
	-	±♂ [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect 🖣 🖿	le_	⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		⊕ [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Cou	ints	⊕ [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	1				0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	1	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±☉ [loop in experiment1 at ppmd01.f			0.100si	0.100s	Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for D	Deeper An	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the S for Dependencies ar	Survey Report nd/or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread		_			Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

Function Call Sites and Loops	å -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
⊕♂ [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
① [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170st	0.170st	Vectorized (Body)	
① [loop in fill_uniform_real at util_random.f9			0.156st	0.156st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152st	0.152st	Vectorized (Body)	
⊕ [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
⊞o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
말 ੴ [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



• Run Intel Advisor again

Vectorization	Threading	🛯 Where should I add vectori	zatio	on and/or threading paralle	lism? 💿												L ADVISOR XE 2016
Workflow	Workflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
orr Batch mod	0	🌪 Summary 🚭 Survey Report 🔅 Refineme	nt Rep	orts 💧 Annotation Report													
Batchillou	ie .					w . 1 w		Ve	Vectori	Vectorized Loops			Instruction Set Analysis		D	19 19 19 19 19 19 19 19 19 19 19 19 19 1	· · · · · · · · · · · · · · · · ·
1. Survey Target		Function Call Sites and Loops	0 -	vector issues	Self lime	lotal lime	iype	why no vectorization?	Vect	Efficiency	Gain	. VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
		±© [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect Di m		⊕o [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		±© [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd Trip Cou	ints	±© [loop in fill_uniform_real at util_random.f9			0.156s	0.156sl	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
	1	±© [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153sl	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect	1	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s I	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
		±© [loop in experiment1 at ppmd01.f			0.100si	0.100s (Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loops for D	Deeper An	±© [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s(Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loops in the S for Dependencies ar	Survey Report nd/or Memory	🖻 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

High [loop in experiment1 at ppmd01.f Image: Conversions present 4.034s Vectorized (B Image: Conversions present 0.254s I 0.254s I Vectorized (Body) Image: Conversions present 0.254s I 0.254s I Vectorized (Body) Image: Conversions present 0.254s I 0.254s I Vectorized (Body) Image: Conversions present 0.254s I 0.254s I Vectorized (Body) Image: Conversions present 0.170s I 0.170s I Vectorized (Body) Image: Conversions present 0.156s I 0.170s I Vectorized (Body) Image: Conversions present 0.156s I 0.150s I Vectorized (Body) Image: Conversion present 0.153s I 0.150s I Vectorized (Body) Image: Conversion present 0.152s I 0.152s I Vectorized (Body) Image: Conversion present 0.100s I 0.100s I 0.100s I 0.100s I Image: Conversion present 0.100s I 0.100s I 0.100s I 0.100s I 0.007s I Image: Conversion present 0.000s I 0.000s I 0.254s I Vectorized (Body) Image: Conversion present Image: Conversion present<	Function Call Sites and Loops	å -	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
Image: Constraint of the second of the se	∎© [loop in experiment1 at ppmd01.f			4.034s	4.034s	Vectorized (B	
Image: Construction of the system of the	⊕ <mark>©</mark> [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
Image: Constraint of the system of the sy	⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9		I Inefficient memory access patterns	0.170s	0.170st	Vectorized (Body)	
Image: Constraint and the symmetry of the symme	① [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
Image: Construction of the system of the	⊕ [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)	
① [loop in experiment1 at ppmd01.f90:100] □ 0.100s I 0.100s I Vectorized (Body) ③ [loop in fill_uniform_real at util_random.f9 □ 9.1 Serialized user function call(s) present 0.067s I 0.067s I Vectorized (Body) ⑤ [loop in ppmd01 at ppmd01.f90:99] □ 0.000s I 4.287s I Scalar Inner loop was alread	⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9		9 1 Serialized user function call(s) present	0.152s	0.152st	Vectorized (Body)	
Image: Description of the symbol of the s	±© [loop in experiment1 at ppmd01.f90:100]			0.100s	0.100s (Vectorized (Body)	
Scalar inner loop was alread.	⊕ <mark>©</mark> [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
	🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



• Run Intel Advisor again

Vectoriz	ation T	breading	Where should I add vectori:	zatio	on and/or threading paralle	lism? 💿											INTE	L ADVISOR XE 2016
Workflo	w V	Norkflow	Elapsed time: 220.37s Ovectorized	O Not	Vectorized 5 FILTER: All Module	All Sources	 Loops 	All Thr	eads	•								٩
	tch mode		🌪 Summary 🛭 🛠 Survey Report 🌖 Refineme	nt Rep	orts 💧 Annotation Report													
OIT DE	iccii mode	_				e 10 m	w	-		Vectorized Loops		Instruction Set Analysis				3	Mored 8	3 19
1. Survey	y Target		Function Call Sites and Loops	0 -	vector issues	Self Time	iotai lime	iype	why No vectorization?	Vect	Efficiency	Gain	VL	Traits	Data Types	Number of Vector Registers	Advanced	Location
		-	± of [loop in experiment1 at ppmd01.f90:134]			4.034s	4.034s	Vectorized (Body)		AVX	~76%	3.04x	4	Divisions; Inserts	Float64	10		ppmd01.f90:134
Collect	Collect 🕅 🖿 📐		± ♂ [loop in fill_uniform_integer at util_rando		1 Data type conversions present	0.254s	0.254s	Vectorized (Body)		AVX	~100%	4.96x	4	Shifts; Type Conversions	Float64; In	5		util_random.f90:5
		0	± of [loop in fill_uniform_real at util_random.f9		1 Inefficient memory access patterns	0.170s	0.170s	Vectorized (Body)		AVX	~100%	4.30x	4		Float64	10	Unrolled by 4	util_random.f90:3
nd	Trip Count	s	± o [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)		AVX	~100%	4.37x	4		Float64	10	Unrolled by 4	util_random.f90:3
			± o [loop in experiment1 at ppmd01.f90:117]			0.153s	0.153s	Vectorized (Body)		AVX	~97%	3.89x	4	Divisions; Inserts	Float64	10		ppmd01.f90:117
Collect			± of [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)		AVX	~100%	4.89x	4	Extracts	Float64	2		util_random.f90:1
	V		±© [loop in experiment1 at ppmd01.f			0.100s1	0.100s(Vectorized (B		AVX	~97%	3.88x	4	Divisions; Inserts	Float64	10		ppmd01.f90:1
Mark Loo	ops for Dee	eper An	± o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s(0.067s (Vectorized (Body)		AVX	~100%	5.48x	4		Float64	1		util_random.f90:1
Select loop for Depend	ps in the Surv dencies and/o	vey Report or Memory	ුල් [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread					Unpacks	Float64	16		ppmd01.f90:99

• List of hot spots

High [loop in experiment1 at ppmd01.f Image: Conversions present 4.034s Vectorized (B Image: Conversions present 0.254s] 0.254s] Vectorized (Body) Image: Conversions present 0.254s] 0.254s] Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion present 0.170s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion present 0.170s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion present 0.156s1 0.170s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion present 0.153s1 0.153s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion present 0.152s1 0.152s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion call(s) present 0.100s1 0.100s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion call(s) present 0.067s1 Vectorized (Body) Image: Conversion fill_uniform_real at util_random.f9 Image: Conversion call(s) present 0.06	Function Call Sites and Loops	đ	Vector Issues	Self Time	Total Time	Туре	Why No Vectorization?
Image: Construction of the system of the	📲 🕲 [loop in experiment1 at ppmd01.f	~		4.034s	4.034s	Vectorized (B	
Image: Constraint of the sympton of	🗄 👩 [loop in fill_uniform_integer at util_rando		🔋 1 Data type conversions present	0.254s	0.254s	Vectorized (Body)	
Image: Constraint of the system of the sy	🗄 🖱 [loop in fill_uniform_real at util_random.f9		🔋 1 Inefficient memory access patterns	0.170s	0.170sl	Vectorized (Body)	
Image: Constraint of the system of the sy	🗄 🖱 [loop in fill_uniform_real at util_random.f9			0.156s	0.156s	Vectorized (Body)	
Image: Constraint of the system of the sy	⊕ [loop in experiment1 at ppmd01.f90:117]	~		0.153s	0.153s	Vectorized (Body)	
① [loop in experiment1 at ppmd01.f90:100] ② ① [loop in fill_uniform_real at util_random.f9 ③ ③ [loop in ppmd01 at ppmd01.f90:99] ③ ③ [loop in ppmd01 at ppmd01.f90:99] ③	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.152s	0.152s	Vectorized (Body)	
Image: Second	⊕o [loop in experiment1 at ppmd01.f90:100]	~		0.100s	0.100s (Vectorized (Body)	
Image: Second star in the second star i	⊕o [loop in fill_uniform_real at util_random.f9		💡 1 Serialized user function call(s) present	0.067s (0.067s (Vectorized (Body)	
	🗵 🖱 [loop in ppmd01 at ppmd01.f90:99]			0.000s (4.287s	Scalar	inner loop was alread



<u>F</u> ile View Help			
🕼 i 🖆 🖬 🕞 i 🖉) 📾 (🛠 To		
Welcome e000 💥			
Vectorization Workflow	Threading Workflow	Check memory access patterns in your application Elapsed time: 224.49s Vectorized Not Vectorized FILTER: All Module ppmd01.f90 Loops All Threads	
OFF Batch mod	e	🥐 Summary 😂 Survey Report 🤌 Refinement Reports 🔥 Annotation Report	
1. Survey Target		Site Location Loop-Carried Dependencies Strides Distribution Access Pattern Site Name O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Mixed strides loop_site_15	
Collect 🕅 🖿	<u>}_</u>	O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Mixed strides loop_site_18 O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 0% / 50% Mixed strides loop_site_20	
1.1 Find Trip Cou ▶ Collect 🖿 🛄	nts ^e	 50%:percentage of memory instructions with unit stride or stride 0 accesses Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration 	
Mark Loops for D Select loops in the S for Dependencies an Access Patterns ana 3 loops are marked	eeper An urvey Report Id/or Memory Ilysis.	Memory Access Patterns Report Dependencies Report Recommendations ID Stride Stride	Typ
2.1 Check Depend	lencies	№ P3 I ▶ P4 I ▶ P4 I ▶ P12 • 43790546; -32607650; -30850839; -29707033; -21902228; -14150377; -1085	Unit Para Vari
2.2 Check Memory Access P.		 gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture - scatter (irregular) accesses, detected for v(p)scatter* instructions on AVX2 Instruction Set Architecture 	



<u>F</u> ile View Help		
da i 🍅 🖬 📚 i 🕻	🕘 📾 l 🕉 To	🏚 🔏 🖞 🖆 🗇
Welcome e000 💥		
Vectorization Workflow	Threading Workflow	Check memory access patterns in your application Image: Elapsed time: 224.49s Vectorized Image: State of the sta
OFF Batch mod	le ^e	Summary & Survey Report Refinement Reports Annotation Report
1. Survey Target		O[loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Nixed strides loop_site_15
Collect	Þ	O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Mixed strides loop_site_18 O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 0% / 50% Mixed strides loop_site_20
1.1 Find Trip Counts		Unit stride (stride 1) = Instructions with unit stride or stride 0 accesses Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
▶ Collect ►		 Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration 0%: percentage of memory instructions with fixed or constant non-unit stride accesses
Select loops in the s for Dependencies a Access Patterns an	Survey Report nd/or Memory alysis. d	Memory Access Patterns Report Dependencies Report
2.1 Check Depen	dencies	ID ID ID Stride With each iteration Type IP 3 I ID
Collect	l	 Prediction P12 -43790546; -32607650; -30850839; -29707033; -21902228; -14150377; -10835 P12 -43790546; -32607650; -30850839; -29707033; -21902228; -14150377; -10835 P12 P14 P
2.2 Check Memor	y Access P	 gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture - scatter (irregular) accesses, detected for v(p)scatter* instructions on AVX2 Instruction Set Architecture



<u>F</u> ile View Help		
🏨 i 🏠 🖬 🌫 i 🐸	🖬 🛛 🕉 To	
Welcome e000 🕱		
Vectorization	Threading	Check memory access patterns in your application in
Workflow	Workflow	Elapsed time: 224.49s 🚺 Vectorized 💽 Not Vectorized 🔄 FILTER: All Module 💽 ppmd01.f90 💽 Loops 💽 All Threads
OFF Batch mode	0	🌳 Summary 📽 Survey Report 🤌 Refinement Reports 👌 Annotation Report
0		Site Location 🔺 Loop-Carried Dependencies Strides Distribution Access Pattern Site Name
1. Survey Target		ق [loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Nixed strides loop_site_15
h Collect h E		O [loop in experiment1 at ppmd01.f90:1 No information available 50% / 50% / 0% Nixed strides loop_site_18
	<u>b_</u>]	Operation of the second descent of the second descent of the second descent des
1.1 Find Trip Cour Collect Collect Collect Collect Collect Coops for December 2015	nts eeper An urvey Report	 50%:percentage of memory instructions with unit stride or stride 0 accesses Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration 0%: percentage of memory instructions with fixed or constant non-unit stride accesses Constant stride (stride N) = Instruction accesses memory that consistently changes
for Dependencies and Access Patterns anal 3 loops are marked	d/or Memory lysis. l	Memory Access Patterns Report Dependencies Report
2.1 Check Depende	encies	Image: Strice Image: Strice<
2.2 Check Memory	Access P	 gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture - scatter (irregular) accesses, detected for v(p)scatter* instructions on AVX2 Instruction Set Architecture



- Advisor tells us
 - · Vectorization is ok
 - Strided memory access in the random access loop is a problem
- · Let's run a **memory access** analysis in VTune



```
$ssh -X vsc20170@login.hpc.uantwerpen.be
Last login: Thu Sep 8 16:38:25 2016 from 143.169.185.55
```

```
Welcome to Hopper!
```

```
vsc20170@ln02 ~$ module load VTune
vsc20170@ln02 ~$ module list
Currently Loaded Modulefiles:
1) GCCcore/5.4.0 4) ifort/2016.3.210-GCC-5.4.0-2.26 7) VTune/2016_update3
2) binutils/2.26-GCCcore-5.4.0 5) iccifort/2016.3.210-GCC-5.4.0-2.26
3) icc/2016.3.210-GCC-5.4.0-2.26 6) Advisor/2016_update4
vsc20170@ln02 ~$
```

vsc20170@ln02 ~\$ amplxe-gui &



















Project Navigator	X /user/antwerpen/201/vsc	20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier	
/user/antwerpen/201/vsc20170/intel/amplxe/	Choose Target and Analys	sis Type	Intel VTune Amplifier XE 2016
▼ /m ppmd01	Analysis Target		
🚠 rooons	A A A A	Memory Access Copy	Start
 Foolhs Foolhs Foolhs Foolha Fool	 ▲ ▲ ▲ ▲ ▲ Algorithm Analysis ▲ Basic Hotspots ▲ Advanced Hotspots ▲ Advanced Hotspots ▲ Concurrency ▲ Locks and Waits ▲ HPC Performance Characterizatio ♥ Microarchitecture Analysis ▲ General Exploration ▲ Memory Access ▲ TSX Exploration ▲ TSX Exploration ▲ TSX Hotspots ▲ SGX Hotspots ♥ Platform Analysis ▲ CPU/GPU Concurrency ▲ GPU Hotspots (preview) ▲ Disk Input and Output (preview) ♥ Custom Analysis 	Memory Access Copy Measure a set of metrics to identify memory access related issues (for example, specific for NUMA architectures). This analysis type is based on the hardware event-based sampling collection. Learn more (F1) CPU sampling interval, ms: 1 Analyze memory objects 1 Minimal memory object size to track, in bytes: 1024 Evaluate max DRAM bandwidth Analyze OpenMP regions O Details Openalis	Start Start Paused Choose Target
	x		Command Line



New analysis

	V 🛛 /user/antwerpen/201/vsc	20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier	
Project Navigator	🖞 🥼 🖆 😥 🕨 🗗 🕼 🕼 🕼 Welcor	me r008macc New Am X	=
/user/antwerpen/201/vsc201/0/intei/ampixe/	Choose Target and Analys	sis Type	Intel VTune Amplifier XE 2016
ppmd01	🔄 🖶 Analysis Target 🕺 Å. Analysis Type		
Am rooths		Memory Access	Chant
🔚 r002hs	×⊕ ×0 × ×	Measure a set of metrics to identify memory access related issues (for	Start
An r003hs	Å Basic Hotspots	example, specific for NUMA architectures). This analysis type is based on the	Start Paused
満 r004ah	Å Advanced Hotspots	hardware event-based sampling collection. Learn more (F1)	Start Paused
Am roosan Am roosan	A Concurrency	CPU sampling interval, ms:	Choose Target
🚠 r007macc	A Locks and Waits	☑ Analyze memory objects	
🚡 r008macc	A. HPC Performance Characterizatio	Minimal memory object size to track, in bytes: 1024	
	Å General Exploration	☑ Evaluate max DRAM bandwidth	
	A Memory Access	Analyze OpenMP regions	
	Å TSX Exploration	. O Dataile	
	A TSX Hotspots	O Decails	
	Platform Analysis		
	Å CPU/GPU Concurrency		
	A GPU Hotspots (preview)		
	Å Disk Input and Output (preview)		
	🗁 Custom Analysis		
	4		Command Line



Project Navigator Image: Characteristic State Image: Characteristic State Image: Characteristic State I	X /user/antwerpen/201/vsc	20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier		
▼ ● ppmd01 ™ r000hs ™ r001hs ™ r002hs ™ r003hs ™ r004ah ™ r006hs ™ r007macc ™ r008macc	🗈 😥 í 🕨 😼 í 🕐 🛛 Welco	me r008macc New Am X		≡
Image: Point of the second	oose Target and Analy	sis Type	Intel VTune Ar	nplifier XE 2016
Image: Toolhs Image: Amage:	Analysis Target Å Analysis Type			Þ
¹ r002hs マンの3hs マンの4h ボックの4h ボックの5h ボックの5h ボックの5h ボックの6hs ボックの7macc ズン	Ar Ar	Memory Access	Copy Start	
	Algorithm Analysis Algorithm Analysis Advanced Hotspots Advanced Hotspots Concurrency Locks and Waits HPC Performance Characterizatio Microarchitecture Analysis General Exploration Memory Access TSX Exploration TSX Hotspots SGX Hotspots Platform Analysis CPU/GPU Concurrency GPU Hotspots (preview) Disk Input and Output (preview) Custom Analysis	Memory Access Measure a set of metrics to identify memory access related issues (for example, specific for NUMA architectures). This analysis type is based hardware event-based sampling collection. Learn more (F1) CPU sampling interval, ms: CPU sampling interval, ms: Analyze memory objects Analyze OpenMP regions Analyze memory access	Copy Start or on the Start Pa	aused



Project Navigator	X /user/antwerpen/201/vsc: Im Im	20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier me r008macc New Am X	=
/// /user/antwerpen/201/vsc20170/intel/amplxe/	Choose Target and Analys	sis Type	Intel VTune Amplifier XE 2016
k r000hs	🔄 🖶 Analysis Target Å Analysis Type		Þ
滿 r001hs 滿 r002hs	Ap Ap Ap Ax	Memory Access	Copy Start
k r003hs r004ah	♥ 🌈 Algorithm Analysis .Å. Basic Hotspots .Å. Advanced Hotspots	Measure a set of metrics to identify memory access related issues (for example, specific for NUMA architectures). This analysis type is based or hardware event-based sampling collection. <u>Learn more</u> (F1)	art Paused
noosan no	Å. Concurrency Å. Locks and Waits	CPU sampling interval, ms: 1 Analyze memory objects	Chd Target
🚡 r008macc	A HPC Performance Characterization	Minimal memory object size to track, in bytes: 1024	
	A General Exploration A Memory Access	Analyze OpenMP regions	Hit start
	Å. TSX Hotspots Å. SGX Hotspots	O Details	
	♥ 📄 Platform Analysis Å- CPU/GPU Concurrency		
	쇼 GPU Hotspots (preview) 쵸 Disk Input and Output (preview)		
	🍃 Custom Analysis		
			Command Line



	X /user/antwerpen/201/vsc20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier	
Project Navigator (1), /user/antwerpen/201/vsc20170/intel/amplxe/	11 🖆 😥 🕨 😰 🕧 😕 Welcome r008macc New Am X	
✓ Ø ppmd01 ₩ r000hs ₩ r001hs ₩ r002hs	 Collecting data Analysis Target Analysis Type ≅ Collection Log Collecting Memory Access data 	Resume
滿 r003hs 滿 r005ah 滿 r005ah 滿 r006hs 滿 r007macc 滿 r008macc	Tue 20 Sep 2016 02:50:47 PM CEST The Intel VTune Amplifier XE 2016 is collecting data. Closing this window will cancel the analysis and terminate the profiled application in case the Launch Application target type is selected. ▲ To enable hardware event-base sampling, VTune Amplifier has disabled the NMI watchdog timer. The watchdog timer will be re-enabled after collection completes. Per-node peak bandwidth measurement is enabled for this collection. Collection time m	 Pause Stop X Cancel Mark Timeline
	Application Output	
		Elapsed time: 00:00:13



000	🗙 /user/antv	verpen/201/vsc20170/ir	ntel/amplxe/project	s/ppmd01 - Intel VTune A	mplifier	
Project Navigator	dha i 🏫 💿 i 🕨 🖼 🕪 🚘	Welcome	r008macc	r009ma X		=
(l) /user/antwerpen/201/vsc20170/intel/amplxe/		Memory Usage	viewpoint	(change) @	Intel V/Tune Amplifier XE	2016
🔻 💯 ppmd01	Access	Memory Usage	e viewponit			2010
🚠 r000hs	🔄 🖶 Analysis Target 🗛 Ana	lysis Type 🛛 🚟 Collect	tion Log 🔋 Sun	nmary 🥵 Bottom-up	E Platform	Þ
🚠 r001hs		. 44 120-				<u>^</u>
🚠 r002hs		: 44.1305	40.007			
🚠 r003hs	<u>CPU Time</u> :		42.827s			
踊 r004ah	The metric value is hi	gh. This can indicate t	hat the significan	t fraction of execution (ipeline slots could be stalled due to demand memory load and stores	
踊 r005ah	Use Memory Access a	nalysis to have the m	etric breakdown l	by memory hierarchy, m	emory bandwidth information, correlation by memory objects.	
踊 r006hs	L1 Bound ⁽²⁾ :		0.033			
📠 r007macc	L2 Bound [®] :		0.000			
👼 r008macc	L3 Bound [®] :		0.000			
🖬 r009macc			0.470			
	This metric shows	how often CPU was s	talled on the mai	n memory (DRAM). Cach	ing typically improves the latency and increases performance.	
	Memory Bandy	vidth [®] :	0.712			
	This metric rep This metric do	presents a fraction of	cycles during wh	ich an application could	be stalled due to approaching bandwidth limits of the main memory (DF an Uncore counters for that). Consider improving data locality in NUMA n	RAM).
	socket system	s.		in eaus/cores/sockets (s	ee oncore counters for that, consider improving data locality in normal	nuici-
	Memory Lat	ency ®:	0.252			
	This metric rep	presents a fraction of	cycles during wh	ich an application could	be stalled due to the latency of the main memory (DRAM). This metric	does
	not aggregate	requests from other	threads/cores/so	ckets (see Uncore cour	ters for that). Consider optimizing data layout or using Software Prefet	ches
	(through the c	ompiler).				
	Remote / L	ocal DRAM Ratio [©] :	0.000			
	Local DRAM	<u>1</u> [™] :	1.000			
	performan	ce.	ads from the loc	al memory exceeds the	e threshold. Consider caching data to improve the latency and increase	2 the
	Bemote DB	AM ⁽²⁾ .	0 000			
	Remote Ca	che [®] :	0.000			
	Loads:		44,152,066,228			
	Stores:		10,512,015,768			
	③ <u>LLC Miss Count</u> ^② :	3	L,600,048,000			
	Average Latency (cyc	les) [©] :	43			
	<u>Iotal Ihread Count</u> : Poused Time ®:		4			
	raused nine		05			
	o - ••					
	⊘ Top Memory O	bjects				
	This section lists the mos	t actively used memo	ry objects in your	application.		
	Memory Object	Loads	Stores	LLC Miss Count		
	[Unknown]	19,448,	,029,172 [Unkr	nown] 1,600,048,000		
	5 m 1 3					/



000	🔀 /user/antw	verpen/201/vsc20170/int	el/amplxe/project	s/ppmd01 - Intel VT	Fune Amplifier
Project Navigator 🛛 🗶	🕼 🖆 👦 🕨 🗗 😂	Welcome	r008macc	r009ma 🗙	
user/antwerpen/201/vsc20170/intel/amplxe/	Memory Access	Memory Usage	viewpoint	(change) @	Intel VTune Amplifier XE 201
🔻 💯 ppmd01	A Memory Access	Memory Usage	viewpoint	(<u>enange</u>) ©	
🏧 r000hs	🔄 \varTheta Analysis Target 🗛 Ana	lysis Type 🛛 🚟 Collecti	ion Log 🔋 🗂 Sum	ımary 🏼 😪 Bottor	n-up 🗄 Platform
🚠 r001hs		. 44 130-			
踊 r002hs		: 44.1305	10.007	[Our program is mamony bound
🚠 r003hs	<u>CPU Time</u> ···		42.82/5		
踊 r004ah	The metric value is his	gh. This can indicate th	hat the significan	t fraction of execu	ution pipeline slots could be stalled due to demand memory load and stores.
踊 r005ah	Use Memory Access a	nalysis to have the me	etric breakdown b	y memory hierarc	hy, memory bandwidth information, correlation by memory objects.
🊠 r006hs	L1 Bound [®] :		0.033		
🚠 r007macc	L2 Bound [®] :		0.000		
🚠 r008macc	L3 Bound [®] :		0.000		
🚠 r009macc	DRAM Bound This matrix shows	how often CPU was st	0.470		Caching tunically improves the latency and increases performance
	This metric shows	in @		Themory (DRAM).	. Caching typically improves the latency and increases performance.
	Memory Bandy This metric rer	<u>vidth</u> ~: presents a fraction of c	0.712 vcles during whi	ich an application	could be stalled due to approaching bandwidth limits of the main memory (DRAM)
	This metric do	es not aggregate reque	ests from other t	threads/cores/sock	ets (see Uncore counters for that). Consider improving data locality in NUMA multi-
	socket system	S.			
	⊖ <u>Memory Lat</u>	ency [®] :	0.252		
	This metric rep	presents a fraction of c	ycles during whi	ich an application	could be stalled due to the latency of the main memory (DRAM). This metric does
	(through the co	ompiler).	nreads/cores/so	ckets (see Oncore	counters for that). Consider optimizing data layout or using software Prefetches
	Bemote / I	ocal DBAM Batio	0 000		
	Local DRAM	1 [®] :	1.000		
	The number	er of CPU stalls on loa	ds from the loca	al memory excee	ds the threshold. Consider caching data to improve the latency and increase the
	performan	ce.			
	Remote DR	<u>AM</u> [@] :	0.000		
	<u>Remote Ca</u>	<u>che</u> ":	0.000		
	Stores:		44,152,066,228		
	Stores LLC Miss Count [®] :	1,	,600,048,000		
	Average Latency (cyc	es) [®] :	43		
	<u>Total Thread Count</u> :		4		
	Paused Time [®] :		0s		
	O Top Memory O	bjects			
	This section lists the mos	t actively used memory	y objects in your	application.	
	Memory Object	Loads	Stores	LLC Miss C	ount [®]
	[Unknown]	19,448,0)29,172 [Unkn	1,600,04	18,000
					· ·

= |



000	🔀 /user/antw	erpen/201/vsc20170/intel/ampl	ke/projects/ppm	d01 - Intel VTune Amp	ifier
Project Navigator 🔀	🕼 🖆 👦 🕨 🗗 😂	Welcome r008m	acc r009	ma X	
user/antwerpen/201/vsc20170/intel/amplxe/	Memory Access	Memory Usage view	noint (cha	nge) @	Intel VTune Amplifier XE 201
👻 💯 ppmd01		Memory Usage view	ponne (<u>enia</u>	<u>nge</u> / ©	
🏧 r000hs	🔄 \ominus Analysis Target 🗛 Anal	ysis Type 🛛 🚟 Collection Log	🗂 Summary	🚱 Bottom-up 🛛 🖽	Platform
🚠 r001hs		. 44 120-			
🚠 r002hs	S Elapsed lime	: 44.130s			
🚠 r003hs	<u>CPU Time</u> : Memory Round [®]		42.82/s		
🚠 r004ah	The metric value is his	h. This can indicate that the	significant fract	tion of execution pipe	line slots could be stalled due to demand memory load and stores.
🚠 r005ah	Use Memory Access a	nalysis to have the metric bre	akdown by mer	mory hierarchy, mem	ory bandwidth information, correlation by memory objects.
👼 r006hs	L1 Bound ⁽²⁾ :		0.033		
👼 r007macc	L2 Bound [®] :		0.000		
👼 r008macc	L3 Bound [®] :		0.000		Due to complete cache misses
👼 r009macc	⊙ <u>DRAM Bound</u> [⊕] :		0.470		
	Memory Bandw This metric rep This metric do socket system Memory Late This metric rep not aggregate (through the co Remote /Lo Local DRAM The numbe performance Remote DR Remote Ca Loads: Stores: ULC Mirs Count	idth ⁽⁰⁾ : resents a fraction of cycles d ss not aggregate requests from s. ency ⁽⁰⁾ : requests from other threads, pmpiler). occal DRAM Ratio ⁽⁰⁾ : ⁽⁰⁾ : e: AM ⁽⁰⁾ : che ⁽⁰⁾ : (2): 44,152, 10,512,	0.712 uring which an m other thread 0.252 uring which an (cores/sockets) 0.000 1.000 n the local mer 0.0000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.000000 0.00000000	application could be s/cores/sockets (see application could be (see Uncore counters mory exceeds the th	stalled due to approaching bandwidth limits of the main memory (DRAM). Uncore counters for that). Consider improving data locality in NUMA multi- stalled due to the latency of the main memory (DRAM). This metric does i for that). Consider optimizing data layout or using Software Prefetches reshold. Consider caching data to improve the latency and increase the
	Average Latency (cycl	es) [@] :	43		
	Total Thread Count:		4		
	Paused Time [®] :		0s		
	Top Memory Ol This section lists the most	bjects actively used memory object	s in your applic	ation.	
	Memory Object	Loads	Stores	LLC Miss Count	
	[Unknown]	19,448,029,172	[Unknown]	1,600,048,000	
The second se	fac. 13				

= |



000	🗙 /user/antwerpen/201/v	sc20170/intel/amplxe/project	s/ppmd01 - Intel VTune	a Amplifier
Project Navigator 🔀	💵 🖆 👦 🕨 🗗 🖓 🖤 🚔 🔞 Wei	come r008macc	r009ma 🗙	
🕼 /user/antwerpen/201/vsc20170/intel/amplxe/	Memory Access Memory	Usage viewpoint ((change) @	Intel VTune Amplifier XE 2016
👻 💯 ppmd01	Memory Access Memory	Usage viewpoliti (<u>cnange</u> / ©	
🚠 r000hs	🔄 \varTheta Analysis Target 🔥 Analysis Type	🞬 Collection Log 🛚 🛱 Sum	ımary 🥵 Bottom-u	p
🚠 r001hs	G Elancod Time [®] , 44.17	200		
🚠 r002hs	© Elapsed Time : 44.13	10.007-		
🚡 r003hs	Memory Bound [®]	42.8275		
🚠 r004ah	The metric value is high. This can	indicate that the significant	t fraction of executior	n pipeline slots could be stalled due to demand memory load and stores.
🗟 r005ah	Use Memory Access analysis to h	ave the metric breakdown b	y memory hierarchy,	memory bandwidth information, correlation by memory objects.
🚠 r006hs	L1 Bound ⁽²⁾ :	0.033		
🚡 r007macc	L2 Bound ⁽³⁾ :	0.000		
🚡 r008macc	L3 Bound [©] :	0.000		
🚡 r009macc	DRAM Bound ": This matric shows how often (0.470		
		cro was scalled on the main	B	andwidth saturates fast because we move an
	<u>Memory Bandwidtn</u> : This metric represents a f	v./12	ich an anni	entire cache line for almost every data item
	This metric does not aggre	egate requests from other t	hreads/cores/sockets	
	socket systems.			
	⊘ Memory Latency [®] :	0.252		
	This metric represents a f	raction of cycles during whi	ich an application cou	Id be stalled due to the latency of the main memory (DRAM). This metric does
	(through the compiler).	om other threads/cores/soo	ckets (see Uncore cou	unters for that). Consider optimizing data layout or using Software Prefetches
	Remote (Local DRAM F	Patio ⁽²⁾ . 0.000		
	Local DRAM [®] :	1.000		
	The number of CPU st	talls on loads from the loca	al memory exceeds t	the threshold. Consider caching data to improve the latency and increase the
	performance.			
	Remote DRAM ⁽²⁾ :	0.000		
	Remote Cache [®] :	0.000		
	Loads:	44,152,066,228		
	Stores: () LLC Miss Count [®] :	1.600.048.000		
	Average Latency (cycles) [®] :	43		
	Total Thread Count:	4		
	Paused Time [®] :	0s		
	○ Top Memory Objects			
	This section lists the most actively us	ed memory objects in your	application.	
	Memory Object	Loads Stores	LLC Miss Coun	t®
	[Unknown]	19.448.029.172 [Unkn	own] 1.600.048.0	00
		15,440,025,172 [OIKI	1,000,040,0	-



000	X /user/antwerpen/201/vsc2	0170/intel/amplxe/projects/ppr	nd01 - Intel VTune Amplifier	
Project Navigator	🖍 🖆 💿 🕨 🔊 🐠 🚔 🔞 Welcom	roog	maX	
user/antwerpen/201/vsc20170/intel/amplxe/		sage viewpoint (ch	ange) @	Intel V/Tune Amplifier XE 2016
👻 💯 ppmd01	Memory Access Memory 0	sage viewponit (<u>cha</u>		
踊 r000hs	🔄 🕀 Analysis Target 🔥 Analysis Type 📟	Collection Log 🔋 🕄 Summary	y 🚭 Bottom-up 🔣 Platform	
🚠 r001hs	○ Elaward Time [®] : 44 130	-		
🚠 r002hs	Elapsed Time : 44.130s	j 40.007-		
🚠 r003hs	Memory Bound [®]	42.8275		
🚠 r004ah	The metric value is high. This can inc	licate that the significant frag	ction of execution pipeline slots coul	d be stalled due to demand memory load and stores.
🚠 r005ah	Use Memory Access analysis to have	the metric breakdown by me	emory hierarchy, memory bandwidtl	h information, correlation by memory objects.
踊 r006hs	L1 Bound [®] :	0.033		
🚠 r007macc	L2 Bound [®] :	0.000		
🚠 r008macc	L3 Bound [®] :	0.000		
🚠 r009macc	ORAM Bound :: This matrix shows how often CBU	0.470	many (DRAM). Cashing tunically impr	aves the latency and increases performance
	This metric shows now orten CPO	was scalled on the main mer	(DRAM): Caching typically impr	oves the latency and increases performance.
	Memory Bandwidth ": This matric represents a fract	0.712	a application could be stalled due to	approaching handwidth limits of the main memory (DPAM)
	This metric does not aggregat	te requests from other threa	ds/cores/sockets (see Uncore count	ers for that). Consider improving data locality in NUMA multi-
	socket systems.			
		0.252		
	This metric represents a fract	tion of cycles during which ar	n application could be stalled due to	the latency of the main memory (DRAM). This metric does
	(through the compiler)	other threads/cores/sockets	(see Uncore counters for that). Co	onsider optimizing data layout or using Software Prefetches
	Remete (Legal DRAM Bati	⁽²⁾ . 0.000		
	Local DRAM [®]	1 000		
	The number of CPU stalls	on loads from the local me	emory exceeds the threshold. Cons	ider caching data to improve the latency and increase the
	performance.		-	
	Remote DRAM [®] :	0.000		
	<u>Remote Cache</u> [®] :	0.000		
	Loads:	44,152,066,228	#	complete cache misses
	Stores:	10,512,015,768		· · · · · · · · · · · · · · · · · · ·
	Average Latency (cycles) [®] :	43		
	Total Thread Count:	4		
	Paused Time [®] :	0s		
	○ Top Memory Objects			
	This section lists the most actively used i	memory objects in your appli	cation.	
	Memory Object	oads Stores	LLC Miss Count ®	
		19 448 029 172 [Upkpowp]	1 600 048 000	
		.9,440,029,172 [UNKNOWN]	1,000,048,000	



000	🔀 /user/antwerpen/201/vsc20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier	
Project Navigator	🕼 🖆 👦 🖡 😥 🕼 🗃 🕐	=
user/antwerpen/201/vsc20170/intel/amplxe/	Memory Access Memory Usage viewpoint (change) @	016
👻 💯 ppmd01		010
🛅 r000hs	🔄 😌 Analysis Target 🛛 🗛 Analysis Type 🔤 Collection Log 📓 Summary 😪 Bottom-up 📴 Platform	⊳
🛅 r001hs	Selanced Time (), 14 130c	-
📠 r002hs		
🔚 r003hs	© Memory Bound [®] : 58.0%	
👼 r004ah	The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores.	
🛅 r005ah	Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.	
🔚 r006hs	L1 Bound [®] : 0.033	
🚡 r007macc	L2 Bound [®] : 0.000	
🚡 r008macc		
🚡 r009macc	DRAM Bound : 0.470 This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance	
	Manager Developing to the international states of the main memory (Divin). Calcing typically improves the latency and incleases performance.	
	This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (DBA)	a)
	This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA mu	iti-
	This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric do	es
	not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetch (through the compiler).	es _
	Remote / Local DRAM Ratio [©] : 0.000	
	Local DRAM [®] : 1.000	
	The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase t performance.	ne
	Remote DRAM [®] : 0.000	
	Remote Cache [®] : 0.000	
	Loads: 44,152,066,228	
	Stores: 10,512,015,768	
	Average Latency (cycles) [®]	
	Total Thread Count: 4 AVERAGE # OF CYCLES WE HAVE TO WAIT FOR	
	Paused Time [®] : 0s a data item (should be ~1!)	
	⊘ Top Memory Objects	
	This section lists the most actively used memory objects in your application.	
	Memory Object Loads Stores LLC Miss Count [®]	
	[Unknown] 19,448,029,172 [Unknown] 1,600,048,000	4
And a state of the		



	🔀 /user/antwerpen/201	l/vsc20170/intel/amplxe/projects/ppmd01 - Intel VTune Amplifier	
Project Navigator	🏨 i 🖆 🎲 i 🕨 😰 🜗 😅 i 🕐 🗰 Welco	me r011ge New Am X	=
Project Navigator (g) (), /user/antwerpen/201/vsc20170/intel/amplxe/ (), /user/antwerpen/201/vsc20170/i	Image: Choose Target and Analysis Choose Target and Analysis Choose Target Analysis Type Analysis Target Analysis Algorithm Analysis Advanced Hotspots Aconcurrency	me r011ge New Am X sis Type General Exploration Copy Analyze general issues affecting the performance of your application. This analysis type is based on the hardware event-based sampling collection. Learn more (F1) Collect stacks	≡ Intel VTune Amplifier XE 2016 Start Start Paused
 ☐ r006hs ☐ r007macc ☐ r008macc ☐ r009macc ☐ r010ge ☐ r011ge 	 A Locks and Waits A Locks and Waits A HPC Performance Characterization ✓ Microarchitecture Analysis A General Exploration A Memory Access A TSX Exploration A TSX Exploration A TSX Hotspots A SGX Hotspots ✓ Platform Analysis A CPU/GPU Concurrency A GPU Hotspots (preview) A Disk Input and Output (preview) ✓ Custom Analysis 	 Analyze memory bandwidth Evaluate max DRAM bandwidth Analyze OpenMP regions Analyze user tasks, events, and counters O Details 	
	4		🗈 Command Line



000		V /user/antwerpen/201/vsc	c20170/intel/amplxe/projects/ppmd01 - Intel VTupe Amplifier
Project Navigator			
🕼 /user/antwerpen/201/vsc20170/intel/amplxe/			
▼ / ppmd01	General Exploration General E	exploration viewpoint (<u>change</u>)	
📅 r000hs	🖪 \ominus Analysis Target 🛝 Analysis Type 🔛 Collect	ion Log 📓 Summary 🥵 Bottom-up 🚱	General Exploration
🔚 r001hs			This view uses <u>hardware event-based metrics</u> to show code
🔚 r002hs	🕑 Elapsed Time 🕆 35.740s 🍺		bottlenecks. Hover over a metric name in the <u>grid</u> for the metric
k r003hs	<u>Clockticks</u> :	126,124,189,186	description.
🔤 r004ah	Instructions Retired:	48,346,072,519	Use this view to:
📠 r005ah	The CPI may be too high. This could be cau	used by issues such as memory stalls, instru	- Identify code regions (modules, functions, and so on) with the
🔚 r006hs	Explore the other hardware-related metrics	to identify what is causing high CPI.	highest execution time.
📠 r007macc	MUX Reliability [®] :	0.996	- <u>Analyze detected hardware issues</u> highlighted by pink cells and get tuning recommendations.
🔚 r008macc	Front-End Bound [®] :	0.7%	
k r009macc	Bad Speculation [®] :	0.0%	Press F1 for help on each window.
🔤 r010ge	⊗ Back-End Bound [™] :	87.3%	Show the analysis description when result opens
An r011ge	describe a portion of the pipeline where the	out-of-order scheduler dispatches ready uOp	is into their respective execution units, and, once completed, these
	uOps get retired according to program orde	r. Stalls due to data-cache misses or stalls du	ue to the overloaded divider unit are examples of back-end bound
	issues.		
		66.2%	
	The metric value is high. This can indica	te that the significant fraction of execution pi	ipeline slots could be stalled due to demand memory load and stores.
	O L1 Round [®]	o odo	ernory bandwider information, correlation by memory objects.
	This metric shows how often machin	e was stalled without missing the L1 data ca	ache. The L1 cache typically has the shortest latency. However, in certain
	cases like loads blocked on older sto	res, a load might suffer a high latency even	though it is being satisfied by the L1. Note that this metric value may be
	highlighted due to DTLB Overhead or	Cycles of 1 Port Utilized issues.	
	DTLB Overhead [®] :	0.441	
	A significant proportion of cycles reducing working-set size to redu same page. Try using larger page	is being spent handling first-level data TLB ice DTLB overhead. Additionally, consider using sizes for large amounts of frequently-used d	misses. As with ordinary data caching, focus on improving data locality and ing profile-guided optimization (PGO) to collocate frequently-used data on the lata.
	Loads Blocked by Store Forwardir	[®] : 0.000	
	Lock Latency [®] :	0.000	
	Split Loads	0.000	
	<u>4K Aliasing</u> [©] :	0.002	
	L2 Bound ©:	0.000	
	⊘ DRAM Bound [®] :	0.576	
	This metric shows how often CPU wa	as stalled on the main memory (DRAM). Cachi	ing typically improves the latency and increases performance.
	Memory Bandwidth ⁽²⁾ :	0.787	
	This metric represents a fraction	of cycles during which an application could l	be stalled due to approaching bandwidth limits of the main memory (DRAM).
	This metric does not aggregate r	equests from other threads/cores/sockets (so	ee Uncore counters for that). Consider improving data locality in NUMA multi-
	Memory Latency : This metric represents a fraction	of cycles during which an application could be	e stalled due to the latency of the main memory (DRAM). This metric does not
	aggregate requests from other (through the compiler)	threads/cores/sockets (see Uncore counter	rs for that). Consider optimizing data layout or using Software Prefetches
	Local DRAM [®] .	1.000	
	The number of CPU stalls o	n loads from the local memory exceeds th	he threshold. Consider caching data to improve the latency and increase the
	performance.	0.000	
	Remote DRAM	0.000	



roject Navigator	🛛 🖉 🖡 🗇 🕞 🗋 🖉 🗇 🛄 👘 😪 🖓 🔤 👘 🖉	
/user/antwerpen/201/vsc20170/intel/a	mplxe/	
✓ Ø ppmd01 № r000hs № r001hs № r002hs	General Exploration	
 ○ r003hs ○ r003hs ○ r003hs ○ r005ah ○ r005ah ○ r007macc ○ r008macc ○ r009macc ○ r010ge ○ r011ge 	This view uses <u>hardware event-based metrics</u> to show code egions that experienced potentially significant architectural oottlenecks. Hover over a metric name in the <u>grid</u> for the metric description.	
- - h	Jse this view to: Identify code regions (modules, functions, and so on) with the highest execution time. Analyze detected hardware issues highlighted by pink cells and get tuning recommendations.	n e 1 and 1 the
F	Press F1 for help on each window.	
	Show the analysis description when result opens	
	This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (D This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA socket systems. Memory Latency : 0.190 This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric do aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefe (through the compiler). Local DRAM ^(D) : 1.000 The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and inco performance.	RAM). multi- es not etches rease the



✓ Elapsed Time²: 35.740s

Clockticks:	126,124,189,186
Instructions Retired:	48,346,072,519
CPI Rate [®] :	2.609

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

MUX Reliability [@] :	0.996
Front-End Bound [®] :	0.7%
Bad Speculation [®] :	0.0%
Back-End Bound [®] :	87.3%

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

66.2%

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

⊘ ⊘

0.040

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB Overhead ⁽²⁾:

0.441

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

DRAM Bound [®] :	0.576
L3 Bound [®] :	0.000
L2 Bound [®] :	0.000
<u>4K Aliasing</u>	0.002
Split Loads [©] :	0.000
Lock Latency [©] :	0.000
Loads Blocked by Store Forwarding	0.000

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance.

Memory Bandwidth

0.787

This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA multisocket systems.

⊘ Memory Latency[®]:

0.190

This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetches (through the compiler).

Local DRAM[®]:

1.000

The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase the performance.

Remote Cache ⁽²⁾ :	0.000
≥ <u>Store Bound</u> [®] :	0.134



Elapsed Time : 35.740)s	
<u>CIOCKTICKS</u> :	126,124,189,186	CPI = cycles per instruction
<u>CPI Rate</u> [®] :	2.609	Peak performance corresponds to 4 instructions per
The CPI may be too high. This coul Explore the other hardware-related	d be caused by issues such as memory stalls, instrumetrics to identify what is causing high CPI.	cycle in DP vectorized code. Hence CPI should be
MUX Reliability ^② :	0.996	between 0.25 and 0.5
Front-End Bound [®] :	0.7%	
Bad Speculation [®] :	0.0%	
Back-End Bound Section Se	87.3%	
dentity slots where no uOps are d describe a portion of the pipeline wl uOps get retired according to progr issues.	anvered due to a lack of required resources for acce here the out-of-order scheduler dispatches ready uOp am order. Stalls due to data-cache misses or stalls o	pting more uops in the back-end of the pipeline. Back-end metrics os into their respective execution units, and, once completed, these Jue to the overloaded divider unit are examples of back-end bound
	66.2%	
The metric value is high. This ca Use Memory Access analysis to	an indicate that the significant fraction of execution p have the metric breakdown by memory hierarchy, n	pipeline slots could be stalled due to demand memory load and stores. nemory bandwidth information, correlation by memory objects.
⊙ <u>L1 Bound</u> ^⑦ :	0.040	
This metric shows how often cases like loads blocked on highlighted due to DTLB Ove	n machine was stalled without missing the L1 data c older stores, a load might suffer a high latency even rhead or Cycles of 1 Port Utilized issues.	cache. The L1 cache typically has the shortest latency. However, in certain In though it is being satisfied by the L1. Note that this metric value may be
DTLB Overhead ^⑦ :	0.441	
A significant proportion (reducing working-set size same page. Try using lan	of cycles is being spent handling first-level data TLB e to reduce DTLB overhead. Additionally, consider us ger page sizes for large amounts of frequently-used	3 misses. As with ordinary data caching, focus on improving data locality and sing profile-guided optimization (PGO) to collocate frequently-used data on the data.
Loads Blocked by Store F	orwarding [®] : 0.000	
Lock Latency ⁽²⁾ :	0.000	
Split Loads ⁽²⁾ :	0.000	
<u>4K Aliasing</u> [©] :	0.002	
L2 Bound ⁽²⁾ :	0.000	
D L3 Bound ¹⁰ :	0.000	
	0.576	en e
This metric shows how ofter	CPU was stalled on the main memory (DRAM). Cach	ning typically improves the latency and increases performance.
<u>Memory Bandwidth</u> [@] :	0.787	
This metric represents a This metric does not ago socket systems.	fraction of cycles during which an application could gregate requests from other threads/cores/sockets (s	be stalled due to approaching bandwidth limits of the main memory (DRAM). see Uncore counters for that). Consider improving data locality in NUMA multi-
	0.190	
aggregate requests for (through the compiler).	n other threads/cores/sockets (see Uncore counte	rs for that). Consider optimizing data layout or using Software Prefetches
Local DRAM [®] :	1.000	
The number of CPU performance.	stalls on loads from the local memory exceeds t	the threshold. Consider caching data to improve the latency and increase the
Bomoto DBAM [®]	0.000	

Remote DRAM [®] :	0.000
<u>Remote Cache</u> ⁽²⁾ :	0.000
Store Bound [®] :	0.134



⊘ Elapsed Time[®]: 35.740s

<u>Clockticks</u> :	126,124,189,186
Instructions Retired:	48,346,072,519
CPI Rate [®] :	2.609

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

MUX Reliability ⁽²⁾ :	0.996
Front-End Bound [®] :	0.7%
Bad Speculation [®] :	0.0%
Back-End Bound [®] :	87.3%

Identify slots where no uOps are delivered due to a lack of required resources for acce describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOp uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound

uops get retired according to program order. Stalls due to data-cache misses or stalls due to the ovenoaded divider unit are examples of back-end bot issues.

Memory Bound[®]:

66.2%

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

⊘ ⊘

0.040

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB Overhead[®]:

0.441

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

_2 Bound [®] :	0.000
<u>_2 Bound</u> [®] :	0.000
2 Bound ®	0.000

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance.

<u>Memory Bandwidth</u> [©]:

0.787

This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA multisocket systems.

Memory Latency[®]:

0.190

This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetches (through the compiler).

Local DRAM[®]:

1.000

The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase the performance.

Remote DRAM [@] :	0.000
Remote Cache ⁽²⁾ :	0.000
∋ <u>Store Bound</u> [®] :	0.134



Elapsed Time^②: 35.740s (\bigtriangledown)

<u>Clockticks</u> :	126,124,189,186
Instructions Retired:	48,346,072,519
CPI Rate [®] :	2.609

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

MUX Reliability [©] :	0.996
Eront-End Bound [®] :	0.7%
Bad Speculation [®] :	0.0%
Back-End Bound [®] :	87.3%

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

Memory Bound[™]:

66.2% The metric value is high. This can indicate that the significant fraction of execution p

Our program is memory bound

Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

\odot L1 Bound[®]:

0.040

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB Overhead ⁽²⁾:

0.441

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

Loads Blocked by Store Forwarding ⁽²⁾ :	0.000
Lock Latency [©] :	0.000
Split Loads ^② :	0.000
<u>4K Aliasing</u> [®] :	0.002
L2 Bound [®] :	0.000
L3 Bound [®] :	0.000
DRAM Bound [®] :	0.576

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance

Memory Bandwidth [©]:

0.787

This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA multisocket systems.

⊗ Memory Latency[®]:

0.190

This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetches (through the compiler).

Local DRAM[®]:

1.000

The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase the performance.

Remote DRAM ⁽²⁾ :	0.000
Remote Cache ⁽²⁾ :	0.000
Store Bound [®] :	0.134



⊘ Elapsed Time^②: 35.740s

<u>Clockticks</u> :	126,124,189,18
Instructions Retired:	48,346,072,51
CPI Rate [®] :	2.60

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

MUX Reliability	3.	0.996
Front-End Bo Bo	ound [®] :	0.7%
Bad Speculat	ion [®] :	0.0%
Back-End Bo	und [®] :	87.3%

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

66.2%

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

0.040

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB Overhead ⁽²⁾:

0.441

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

\odot	DRAM Bound [®] :	0.576	
\odot	L3 Bound [®] :	0.000	
	L2 Bound [®] :	0.000	
	<u>4K Aliasing</u>	0.002	
	Split Loads ⁽²⁾ :	0.000	
	Lock Latency [©] :	0.000	
	Loads Blocked by Store Forwarding	0.000	

Our program is memory bound

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance.

Memory Bandwidth

0.787

This metric represents a fraction of cycles during which an application could be stalled due to approaching bandwidth limits of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider improving data locality in NUMA multisocket systems.

⊘ Memory Latency[®]:

0.190

1.000

This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetches (through the compiler).

Local DRAM[®]:

The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase the performance.

	Remote Cache ~:	0.000
<u>J Store Bound</u> . 0.134	Store Bound [®] :	0.134



Elapsed Time^②: 35.740s (\bigtriangledown)

<u>Clockticks</u> :	126,124,189,18
Instructions Retired:	48,346,072,51
CPI Rate [®] :	2.60

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

MUX Reliability ^② :	0.996
Front-End Bound [®] :	0.7%
Bad Speculation [®] :	0.0%
Back-End Bound [®] :	87.3%

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

⊘ Memory Bound[®]:

66.2%

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.

\odot L1 Bound[®]:

0.040

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB Overhead ⁽²⁾:

0.441

0.787

0.190

1.000

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

Loads Blocked by Store Forwarding ⁽²⁾ :	0.000
Lock Latency [®] :	0.000
Split Loads ⁽²⁾ :	0.000
<u>4K Aliasing</u> [®] :	0.002
L2 Bound [®] :	0.000
D L3 Bound [®] :	0.000
ORAM Bound [®] :	0.576

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the latency and increases performance

Memory Bandwidth [©]:

This metric represents a fraction of cycles during which an application cou This metric does not aggregate requests from other threads/cores/sockets socket systems.

Bandwidth saturates fast because we move an entire cache line for almost every data item

⊗ Memory Latency[®]:

This metric represents a fraction of cycles during which an application could be stalled due to the latency of the main memory (DRAM). This metric does not aggregate requests from other threads/cores/sockets (see Uncore counters for that). Consider optimizing data layout or using Software Prefetches (through the compiler).

Local DRAM[®]:

The number of CPU stalls on loads from the local memory exceeds the threshold. Consider caching data to improve the latency and increase the performance.

Remote cache .	
Remote Cache®	0 000
Remote DRAM [®] :	0.000



	normate exercit	0.000	
	Store Bound [®] :	0.134	
	⊘ Core Bound [®] :	21.1%	
	This metric represents how much Core n instructions are both categorized under overloaded or dependencies in program's	on-memory issues were of a bo Core Bound. Hence it may indi data- or instruction- flow are limi	tleneck. Shortage in hardware compute resources, or dependencies software's cate the machine ran out of an OOO resources, certain execution units are ing the performance (e.g. FP-chained long-latency arithmetic operations).
	Divider ^③ :	0.099	
	Port Utilization [®] :	0.373	
	This metric represents a fraction of data-dependency between nearby in: feature auto-Vectorization options toda	f cycles during which an appli structions, or a sequence of ins ay - reduces pressure on the exe	ation was stalled due to Core non-divider-related issues. For example, heavy tructions that overloads specific ports. Hint: Loop Vectorization - most compilers aution ports as multiple elements are calculated with same uop.
	Cycles of 0 Ports Utilized [®] :	0.627	
	This metric represents cycles fract	ion CPU executed no uops on any	execution port.
	Cycles of 1 Port Utilized ⁽²⁾ :	0.132	
	Cycles of 2 Ports Utilized	0.089	
	Over the second seco	0.068	
\odot	Retiring [®] :	12.0%	
	Total Thread Count:	2	
	Paused Time [®] :	Os	

CPU Usage Histogram \odot

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.





	normate exercit	0.000	
	Store Bound [®] :	0.134	
	⊘ Core Bound [®] :	21.1%	
	This metric represents how much Core n instructions are both categorized under overloaded or dependencies in program's	on-memory issues were of a bo Core Bound. Hence it may indi data- or instruction- flow are limi	tleneck. Shortage in hardware compute resources, or dependencies software's cate the machine ran out of an OOO resources, certain execution units are ing the performance (e.g. FP-chained long-latency arithmetic operations).
	Divider ^③ :	0.099	
	Port Utilization [®] :	0.373	
	This metric represents a fraction of data-dependency between nearby in: feature auto-Vectorization options toda	f cycles during which an appli structions, or a sequence of ins ay - reduces pressure on the exe	ation was stalled due to Core non-divider-related issues. For example, heavy tructions that overloads specific ports. Hint: Loop Vectorization - most compilers aution ports as multiple elements are calculated with same uop.
	Cycles of 0 Ports Utilized [®] :	0.627	
	This metric represents cycles fract	ion CPU executed no uops on any	execution port.
	Cycles of 1 Port Utilized ⁽²⁾ :	0.132	
	Cycles of 2 Ports Utilized	0.089	
	Over the second seco	0.068	
\odot	Retiring [®] :	12.0%	
	Total Thread Count:	2	
	Paused Time [®] :	Os	

CPU Usage Histogram (\checkmark)

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.





	normate exercit	0.000	
	Store Bound [®] :	0.134	
	⊘ Core Bound [®] :	21.1%	
	This metric represents how much Core n instructions are both categorized under overloaded or dependencies in program's	on-memory issues were of a bo Core Bound. Hence it may indi data- or instruction- flow are limi	tleneck. Shortage in hardware compute resources, or dependencies software's cate the machine ran out of an OOO resources, certain execution units are ing the performance (e.g. FP-chained long-latency arithmetic operations).
	Divider ^③ :	0.099	
	Port Utilization [®] :	0.373	
	This metric represents a fraction of data-dependency between nearby in: feature auto-Vectorization options toda	f cycles during which an appli structions, or a sequence of ins ay - reduces pressure on the exe	ation was stalled due to Core non-divider-related issues. For example, heavy tructions that overloads specific ports. Hint: Loop Vectorization - most compilers aution ports as multiple elements are calculated with same uop.
	Cycles of 0 Ports Utilized [®] :	0.627	
	This metric represents cycles fract	ion CPU executed no uops on any	execution port.
	Cycles of 1 Port Utilized ⁽²⁾ :	0.132	
	Cycles of 2 Ports Utilized	0.089	
	Over the second seco	0.068	
\odot	Retiring [®] :	12.0%	
	Total Thread Count:	2	
	Paused Time [®] :	Os	

CPU Usage Histogram \odot

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.





- \cdot Advisor is profiler
 - · Analyzes your code on a per statement basis
 - · Looks at the assembly code to analyze vectorization
 - \cdot Hints to the location of the problem
- Vtune Amplifier accumulates statistics on hardware events such as expensive instructions, vector instructions, cache misses, ...
 - Statistics accumulated on a per subprogram (function, subroutine) basis, not per statement
 - \cdot Hints to the nature of the problem
- · Both are complementary


- Compiler does good job at producing vectorized code
- Advisor will tell you if and why the compiler is sometimes not able to produce vectorized code, and will suggest solutions
- Advisor tells you which parts of your code consume the most cputime and are candidates for optimization



- Most often performance problems on modern cpus are due to memory access problems (DRAM latency hits you)
- VTune amplifier gives you clues on how and where to fixes the issues
 - · CPI and Cache Misses
- \cdot Optimize
 - 1. If there are cache misses, try to reduce them
 - · Easier said than done (we'll come to that in the next section)
 - 2. If you are memory bound and CPI is high,
 - 1. Verify vectorization
 - 2. Increase the computational complexity (do more useful work on the data while it is in cache)



- \cdot Suppose we have 10⁹ atoms
- · Computing all interactions in single precision
 - $\cdot 10^{9}(10^{9}-1)/2 \sim 0.5 \ 10^{18}$
 - · complexity $O(N^2)$ not a good idea
 - Adding 1 atom increases the work by a factor $N = 10^9$
 - · Adding 2 atoms increases the work by a factor $N \cdot N = 10^{18}$

• ...

- Lennard-Jones is short range
 - $\cdot \lim_{r \to \infty} 4\pi r^2 V_{LJ}(r) \to 0$
 - In practice cut-off $r_c \cong 2.5$



Implementing cut-off

```
forces = 0
do i=1,N
   do j=1,N-1
      r2 = squared_distance(i,j)
      if r2<rcutoff2
         force ij = ljforce(r2)
         force(i) = force(i) + force_ij
         force(j) = force(j) - force_ij
      endif
   enddo
enddo
integrate forces to update atom positions
```

• Still $O(N^2)$ \otimes

Might be ok for small N



Implementing cut-off

· Verlet lists

- Verlet list of atom i is list of all atoms j for which j < i and $r_{ij} < r_c$
- Increase cutoff slightly so that we do not have to update the Verlet lists at every timestep (depending on how vigorously the atoms move)
- \cdot Verlet list construction is amortized



- Construction of Verlet lists is still $O(N^2)$
- Is dominant data structure: typically between 50 and 100 neighbour atoms/atom



- Put atoms in cells of width $r_c : O(N)$
- · Only atoms in neigbouring cells can satisfy $r_{ij} < r_c$
- Because of symmetry only half of the neighbouring cells must be examined
- Construct Verlet lists as follows
 - Loop over all cells [O(N)]
 - Loop over all neighbours of the current cell using the neighbour stencil [0(1)]
 - Construct the Verlet list of all atoms in the current cell [0(1)]
- Now our MD algorithm is O(N)





- Put atoms in cells of width $r_c : O(N)$
- · Only atoms in neigbouring cells can satisfy $r_{ij} < r_c$
- Because of symmetry only half of the neighbouring cells must be examined
- Construct Verlet lists as follows
 - Loop over all cells [O(N)]
 - Loop over all neighbours of the current cell using the neighbour stencil [0(1)]
 - Construct the Verlet list of all atoms in the current cell [0(1)]
- Now our MD algorithm is O(N)





- Atoms move!
- Iterating over the Verlet lists to compute the interactions will soon jump randomly through memory
- Performance evolves naturally to the random access case
- · Fix data access pattern using spatial sorting
 - Spatial sort = ensure that atoms which are close in space are also close in memory
 - · This reduces cache misses



Fixing the data access pattern

- · Space filling curve
- Linearize a space of dimension >1
- Hilbert curve
- Hilbert index: coordinate of a cell along the Hilbert curve
- Locality guarantee: points close in space are also close along the space filling curve (on average)















- Sort atom property arrays (rx,ry,rz,vx,vy,vz,...) based on the Hilbert index h of the cell of the atoms (spatial sort).
 Atoms which are close in space (and hence will interact) will be close in memory (and hence will be in the cache with high probability)
 - 2. Build a table containing the index of the first atom in each cell, and the number of atoms in the cell (Hilbert list)
 - 3. Build Verlet list from the Hilbert list (discard the latter)
 - 4. Compute the interactions by looping over the Verlet list
 - 5. Integrate forces, updating velocities and positions and time
 - If need_to_rebuild_verlet_list is true jump back to step 1.
 - continue at step 4.



- Sort atom property arrays (rx,ry,rz,vx,vy,vz,...) based on the Hilbert index h of the cell of the atoms (**spatial sort**). Atoms which are close in space (and hence will interact) will be close in memory (and hence will be in the cache with high probability)
- 2. Build a table containing the index of the first atom in each cell, and the number of atoms in the cell (Hilbert list)
- 3. Build Verlet list from the Hilbert list (discard the latter)
- → 4. Compute the interactions by looping over the Verlet list
 - 5. Integrate forces, updating velocities and positions and time
 - If need_to_rebuild_verlet_list is true jump back to step 1.
 else
 - continue at step 4.-



Fixing the data access pattern

- \cdot We need to
 - · Compute Hilbert indices
 - Sort atom property arrays
 - Build Hilbert list and Verlet list
- · Fixing data access patterns can be a lot of work



- Implementation in Fortran? C? C++?
- · Arguments
 - · C++ is inefficient
 - Modern compilers good enough to generate efficient code
 - $\cdot\,$ After all your are using the same hardware



- **Implementation in Fortran? C? C++?** •
- Arguments



- Modern compilers good enough to generate efficient code
- · After all your are using the same hardware



- Implementation in Fortran? C? C++?
- · Arguments
 - \cdot C++ is inefficient



- · Modern compilers good enough to generate efficient code
- $\cdot\,$ After all your are using the same hardware
- Fortran is efficient



- Implementation in Fortran? C? C++?
- · Arguments
 - · C++ is inefficient



Ie #∠

• After all your are using the same hardware

· Fortran is efficient

- Also fortran has constructs that sometimes come in handy, but can kill performance
- But C++ has quite a bit more features which can kill performance than Fortran
 - $\cdot\,$ Because C++ is a general purpose language and Fortran is meant for scientific computing
- · Yet these features can be extremely useful if you use them wisely
 - For computational kernels where performance is an issue you generally need to stay close to the C subset and far away from the C++ features such as classes, inheritance, virtual functions, etc. (templates are an exception)





- Implementation in Fortran? C? C++?
- · Arguments
 - · C++ is inefficient



Ie #∠

• After all your are using the same hardware

· Fortran is efficient

- Also fortran has constructs that sometimes come in handy, but can kill performance
- But C++ has quite a bit more features which can kill performance than Fortran
 - $\cdot\,$ Because C++ is a general purpose language and Fortran is meant for scientific computing
- · Yet these features can be extremely useful if you use them wisely
 - For computational kernels where performance is an issue you generally need to stay close to the C subset and far away from the C++ features such as classes, inheritance, virtual functions, etc. (templates are an exception)





· I'll use C++ because I know it better



· I'll use C++ because I know it better

Most often a lie too!



I'll use C++ because I know it better

Most often a lie too!

- Unless you have read and understood all the C++ books by Scott Meyers, Herb Sutter, Andrei Alexandrescu, Nicolai Josuttis
- In which case you probably also understand which C++ features can kill performance and when they should be used to your advantage
- For number-crunching I find myself advancing faster using Fortran than using C++ (which I do know better!)





- I'll use C++ because it is better documented
 - There aren't too many books on Fortran like the above ones on C++
 - There is no website of the same quality as cplusplus.com or cppreference.com for Fortran (imho)
 - · But still it is much harder to learn and to learn to use efficiently
 - Not a valid argument







Not a lie



- I'll use C because that is the language in which Python was written and I want too integrate my code with Python Good point!
 - · Python integration leverages your code with
 - A high level programming interface:
 - Providing initial data for your simulation is much easier and flexible through a Python script than having to parse input files...
 - Compose and customize high level solution schemes with ease (e.g. choosing another solver for a subproblem)
 - Hundreds of very useful open source Python libraries: Numpy, Scipy, Pandas, matplotlib, ...
 - But ...



- I'll use C because that is the language in which Python was written and I want too integrate my code with Python [continued]
 - ...
 - But the easiest way to create your own module that can be imported in Python is through **f2py**
 - $\cdot\,$ Automatically turns your Fortran code into a Python module
 - $\cdot\,$ As simple as
 - \cdot F2py -c mysource.f90 -m myPythonModule
 - Automatically integrates with Numpy! Pass Numpy arrays to your own Fortran library with no effort and no copying of data!
 - \cdot Much harder in C
 - Also feasible in C++ with the help of boost.python and boost.multi_array, easier than C but not as easy as f2py
 - ∴ stick to Fortran

Choosing a programming language

- I'll use C/C++ because I don't want to mess with storage orders
 - Fortran uses row-major ordering indices start at 1
 - · C/C++ use column-major ordering indices start at 0
 - Plenty of ways to mess up!
 - Inadvertent copying of the array when passing to fortran!
 - Trivial for 1D arrays (but mind the indexing)
 - Numpy arrays by default use the C convention, but arrays can be easily made to follow the Fortran convention:
 - A=numpy.empty((2,2), dtype=np.float32, order='F')
 - \cdot A little bit of experimentation will take away the confusion
 - Simplest way to avoid problems:
 - $\cdot\,$ If you use fortran modules adhere to fortran convention in numpy arrays
 - $\cdot\,$ If you use C/C++ modules adhere to C/C++ convention in numpy arrays
 - · If you use both pay attention..



Intermezzo





- \cdot I'll use X because I need to use some library Y that is written in X
 - Point taken, often the easiest way
 - Valid argument for libraries that deal with parallelization issues:
 - \cdot TBB, for shared memory parallelization (C++)
 - Libraries abstracting vector instructions Vc, boost.simd (C++)
 - However, there is a lot of support for mixed language programming
 - If you do not master X you might end up writing inefficient code and loosing the advantage of using Y
 - it may be worthwhile to find out how to tackle the mixed language challenge
 - · Once done, you will proceed faster and write efficient code





- Stick to Fortran (unless you are a seasoned C++ programmer)
- Using Python and Numpy for high level programming and Fortran (C++) for your own number-crunching routines is a very practical approach
 - \cdot Use f2py to turn your fortran routines into a Python module that is compatible with Numpy



- Details of Python+Numpy+Fortran/C++ is topic of another talk
- Including shared memory parallelism (multithreading) and distributed memory parallelism (multi-node)







 Take derivative of (Lennard-Jones) potential with respect to interatomic distance vector = force exerted on the atoms

$$\cdot \ \frac{dV(r)}{d\vec{r}} = \frac{dV_2(r^2)}{dr^2} \frac{dr^2}{d\vec{r}} = \frac{dV_2(r^2)}{dr^2} \ 2 \ \vec{r} = f(r^2) \vec{r}$$

- · Loop over i
 - · Loop over $j \in VL_i$
 - $\cdot \vec{a}_i += f(r_{ij}^2)\vec{r}_{ij}$
 - $\cdot \ \vec{a}_j = f(r_{ij}^2)\vec{r}_{ij}$
- · 3 x load (\vec{r}_j)
- · 3 x load (\vec{a}_j)
- · 3 x store (\vec{a}_j)





- Baseline case:
 - \cdot N atoms
 - \cdot Compute interaction forces of atom 0 with all other atoms
 - Contiguous memory access
 - · Bandwidth saturated





































In terms of interactions/s MD is about 5 times slower than MC

- A bit more instructions per interaction, but MC is memory bound, that should not matter
- 3 times more memory access
- With a read:write ratio of 2:1 the bandwidth drops from 11 GB/s to 9.5 GB/s
- 3x11/9.5 = 3.47
- Still factor 1.44 slower than expected


• Three cases:

- 1. Atoms on FCC lattice
- 2. Permute the atoms (=random memory access)
- 3. Spatial sort by hilbert index
- Every experiment build the Verlet list and computes the interactions
- CPUtime is measured only for computing the interactions
- Plot result relative to baseline
 - \cdot 90 10⁶ interactions/s





- $\cdot\,$ Put atoms on FCC lattice
- · 4 atoms per unit cell
- Closest neighbor distance = LJ r_{min}
- $\cdot r_{cutoff} = 3r_{min}$

























































































- Sort atom property arrays (rx,ry,rz,vx,vy,vz,...) based on the Hilbert index h of the cell of the atoms (spatial sort).
 Atoms which are close in space (and hence will interact) will be close in memory (and hence will be in the cache with high probability)
 - 2. Build a table containing the index of the first atom in each cell, and the number of atoms in the cell (Hilbert list)
 - 3. Build Verlet list from the Hilbert list (discard the latter)
 - 4. Compute the interactions by looping over the Verlet list and measure the performance (e.g. interactions/s)
 - 5. Integrate forces, updating velocities and positions and time
 - If performance degrades jump back to step 1. else
 - continue at step 4.



- Sort atom property arrays (rx,ry,rz,vx,vy,vz,...) based on the Hilbert index h of the cell of the atoms (spatial sort). Atoms which are close in space (and hence will interact) will be close in memory (and hence will be in the cache with high probability)
- 2. Build a table containing the index of the first atom in each cell, and the number of atoms in the cell (Hilbert list)
- 3. Build Verlet list from the Hilbert list (discard the latter)
- 4. Compute the interactions by looping over the Verlet list and measure the performance (e.g. interactions/s)
 - 5. Integrate forces, updating velocities and positions and time
 - 6. If performance degrades jump back to step 1. else

continue at step 4.



Performance analysis

- atoms : 2,13 10⁶
- pairs : $162 \ 10^6$
- · Ratio : 76
- Pairs computed per second : 88.6 10⁶

•	B/atom: 376,5	(320B is in the verlet list)
•	Bandwidth = $9,5 \text{ GB/s}$	(measured by mlc 2 reads : 1 write)
•	maximum atoms per second:	27.000.000
•	actual atoms per second:	1.160.000
•	ratio:	0,04
•	flops_per_pair :	27
•	flops_per_atom :	2055,5
•	gflops_per_second:	2,39
•	peak performance :	11,2
•	ratio :	0,21



Performance analysis

- atoms : 2,13 10⁶
- pairs : 162 10⁶
- Ratio : 76
- Pairs computed per second : 88.6 10⁶
- B/atom: 376,5
- Bandwidth = 9,5 GB/s
- maximum atoms per second: 27
- actual atoms per second:
- ratio:
- flops_per_pair :
- flops_per_atom :
- gflops_per_second:
- peak performance :
- ratio :

(320B is in the verlet list) (measured by mlc 2 reads : 1 write) 27.000.000 1.160.000 0,04 Not memory bound 27 2055,5 Lot of flops per atom! 2,39 11,2

0,21

Not compute bound either!



Roofline MD setting





Roofline MD setting





What do the intel tools tell? Advisor

• • •	X /user/antwerpen/201/vsc20170/intel/advixe/projects/pyMD - Intel Advisor					
<u>F</u> ile View Help						
Welcome e000 🕱						
Vectorization Threading	Where should I add vectorization and/or threading parallelism?					
Workflow Workflow	Elapsed time: 55.75s 👩 Vectorized 💿 Not Vectorized 🔄 🛛 FILTER: All Module 💽 All Sources 💽 Loops 💽 Al	II Threads	· Q			
	🌳 Summary 🛭 🔩 Survey Report 🔥 Refinement Reports 💧 Annotation Report					
OFF Batch mode			A			
1. Survey Target	Function Call Sites and Loops & Vector Issues Self Time Total Time	Type Why No Veo	ctorization?			
	🗵 🖸 [loop in compute_interactions_verlet_linear at md.f90:328] 🔲 💡 2 Assumed dependency present 9.390s 📩 9.390s	Scalar 🗧 vector de	ependence prevents vectoriz			
	☑♂ [loop in helper::VerletListBuilder::add_cell_cell] □ 5.034s □ 5.034s □ 5	Scalar	•			
1 1 Find Trip Counts			<u> </u>			
LI Fild Tip Coults	Source Top Down Loop Analytics Loop Assembly 💡 Recommendations 🖬 Compiler Diagnostic Details					
Collect 🖿 📃	File: md.f90:328 compute_interactions_verlet_linear					
	Line Source	Total Time %	Loop Time % Traits			
Mark Loops for Deeper An	317					
Select loops in the Survey Report	318 call cpu_time(cpustart)					
for Dependencies and/or Memory	319 do lter=1,n_lter					
There are no marked loops	320 H J=1		17.900s			
	321 do la=L, n_atoms	0.010-1				
2.1 Check Dependencies	322 Ia_parts = vertet_linear(j)) 2010.0	17 0005			
2.1 Check Dependencies	S25 - u0 (k - j+L, j+L, j+L, j+L, j+L, j+L, j+L, j+L,		17.9005			
Collect	Scalar loop Not vectorized vector dependence prevents vectorization					
	No loo transformations applied to applicable provints vectorization					
Nothing to analyze	<pre>324 ia = verlet linear(k)+1 ! +1 since fortran starts counting from 1 !</pre>	0.280s (
2.2 Check Memory Access P	dx = rx(ia) - rx(ia)	0.330s (
	326	0.320s (
Collect 🖿 📘	$327 dz = rz(ja) \cdot rz(ia)$	0.527s				
Nothing to analyze	328 🗉 aij = lj_force_factor2(dx**2 + dy**2 + dz**2)	9.729s 🗖	17.850s 🔳			
o analyze a	329 ! update particle ia acceleration					
	330 ax(ia) = ax(ia) + aij*dx	0.569s				
	331 ay(ia) = ay(ia) + aij*dy	2.758s				
	332 az(ia) = az(ia) + aij*dz	1.360s				
	333 ! update particle ja acceleration					
	334 ax(ja) = ax(ja) - a1)*dx	0.510s				
	335 $3y(3a) = 3y(3a) - 3a(3a) + 3a(3a)$	0.1905 (
	330 az(ja) = az(ja) - alj*0z	0.6885				
	A A A A A A A A A A A A A A A A A A A	4				



```
j=1
do ia=1,n atoms
    ia pairs = verlet linear(j) ! Size of the Verlet list of atom ia
    do k = j+1, j+ia pairs
        ja = verlet linear(k)+1 ! +1 since Fortran starts counting from 1 !
        dx = rx(ja) - rx(ja)
        dy = ry(ja) - ry(ja)
        dz = rz(ja) - rz(ia)
        aij = lj_force_factor2( dx * 2 + dy * 2 + dz * 2)
      ! update particle ia acceleration
        ax(ia) = ax(ia) + aij*dx
        ay(ia) = ay(ia) + aij*dy
        az(ia) = az(ia) + aij*dz
      ! update particle ja acceleration
        ax(ja) = ax(ja) - aij*dx
        ay(ja) = ay(ja) - aij*dy
        az(ia) = az(ia) - aii*dz
    enddo
    j = j + 1 + ia pairs
enddo
```



j=1

do ia=1,n atoms

```
ia_pairs = verlet_linear(j) ! Size of the Verlet list of atom ia
   do k = j+1, j+ia pairs
        ja = verlet_linear(k)+1 ! +1 since Fortran starts counting from 1 !
        dx = rx(ja) - rx(ia)
        dy = ry(ja) - ry(ja)
        dz = rz(ja) - rz(ia)

    SIMD vectorization means that you

        aij = lj_force_factor2( dx**2/
                                         update ax(aj) for 4 successive ja
      ! update particle ia acceleration
                                         values (also ay(aj) and az(ja))
        ax(ia) = ax(ia) + aij*dx
                                        • The compiler cannot know that the ja
        ay(ia) = ay(ia) + aij*dy
        az(ia) = az(ia) + aij*dz
                                         are different
      ! update particle ja acceleration

    Assumed dependency

        ax(ja) = ax(ja) - aij*dx
                                        • We know that the j a are different by
        ay(ja) = ay(ja) - aij*dy
                                         construction of the Verlet list
        az(ja) = az(ja) – aij∗dz
   enddo
                                        • We must tell the compiler to ignore
    j = j + 1 + ia pairs
                                         assumed dependencies
enddo
```



What do the intel tools tell?

```
j=1
do ia=1,n atoms
    ia pairs = verlet linear(j) ! Size of the Verlet list of atom ia
    !DIR$ SIMD
                            Ignore assumed dependencies and vectorize the loop
    do k = j+1, j+ia pairs
        ja = verlet_linear(k)+1 ! +1 since Fortran starts counting from 1 !
        dx = rx(ja) - rx(ja)
        dy = ry(ja) - ry(ja)
        dz = rz(ja) - rz(ia)
        aij = lj_force_factor2( dx * 2 + dy * 2 + dz * 2)
      ! update particle ia acceleration
        ax(ia) = ax(ia) + aij*dx
        ay(ia) = ay(ia) + aij*dy
        az(ia) = az(ia) + aij*dz
      ! update particle ja acceleration
        ax(ja) = ax(ja) - aij*dx
        ay(ja) = ay(ja) - aij*dy
        az(ia) = az(ia) - aii*dz
    enddo
    j = j + 1 + ia pairs
enddo
```



- "inserts present" = hint for gather/scatter
- Filling a vector register element per element (in the case of non-contiguous elements)
- AVX (highest SIMD extension available on Hopper) has no built-in support for gather/scatter
- AVX2 has special instructions for gather/scatter
 - · Available on BrENIAC
 - Available on successor of Turing



```
i=1
do ia=1,n atoms
    ia pairs = verlet linear(j) ! Size of the Verlet list of atom ia
    !DIR$ SIMD
    do k = j+1, j+ia pairs
        ja = verlet_linear(k)+1 ! +1 since Fortran starts counting from 1 !
        dx = rx(ja) - rx(ia)
                                            Gather operation moving
                                          rx(ja), ry(ja), rz(ja)
        dy = ry(ja) – ry(ia)
                                           for 4 successive ja values
        dz = rz(ja) – rz(ia)
                                            into the vector registers
        aij = lj force factor2( dx * 2 + dy * 2 + dz * 2)
      ! update particle ia acceleration
        ax(ia) = ax(ia) + aij*dx
        ay(ia) = ay(ia) + aij*dy
        az(ia) = az(ia) + aij*dz
      ! update particle ja acceleration
        ax(ja) = ax(ja) - aij*dx
                                           Scatter operation moving
                                           ax(ja), ay(ja), az(ja)
        ay(ja) = ay(ja) - aij*dy
                                           for 4 successive ja values
        az(ja) = az(ja) - aij*dz
                                           out of the vector registers.
    enddo
    j = j + 1 + ia pairs
enddo
```



- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers





- $\cdot C = A + B$
- No gather/scatter, 4 successive items moved as a block into/out of vector registers




 $\cdot C = A + B$

gather/scatter, items moved one by one into/out of vector registers





- $\cdot C = A + B$
- gather/scatter, items moved one by one into/out of vector registers





- $\cdot C = A + B$
- gather/scatter, items moved one by one into/out of vector registers





- $\cdot C = A + B$
- gather/scatter, items moved one by one into/out of vector registers





- LLC misses
 LJ_force_factor2
 Compute_interactions
 0.008 %
 0.59 cpi
 0.77 cpi
- · cpi at best 0.25
- · cpi \leq 1 considered acceptable in HPC
- \cdot There is still room for improvement
 - \cdot No more low hanging fruit, though



- \cdot Assembly code
 - · Architecture dependent code
- · Libraries for writing explicit vector code
 - · Architecture independent code
 - \cdot C++ only
 - **Vc** (github.com/VcDevel/Vc)
 - **Boost.simd** (github.com/NumScale/boost.simd)
 - Similar performance
- Rewrite algorithm in terms of vector operations rather than scalar operations



- Gain:
 - · 120–127 10⁶ interactions per second (instead of 90 10⁶)
 - · 30% improvement
 - Not bad, but not effortless (took me about 1¹/₂ week)
 - · AVX2 will probably do even better
- · Coding details in other talk





1. Data access pattern is crucial to performance





- Monte Carlo case (3N reads, no writes)
- · Contiguous data access:
 - \cdot 450 10⁶ interactions/s
 - · Bandwidth saturation (machine limit)
- Random access:
 - · Performance drops by factor 15 (cache misses, gather/scatter)





- Bandwidth saturation means that the CPU is waiting for the data to arrive
 - Try to do more computations with the data that is available, e.g.:
 - · Additional computations
 - · A more complex model
 - Program will not run faster but will do more work in the same time





- Molecular dynamics case (6N reads, 3N writes + VL)
- · Contiguous data access:
 - · 90 10⁶ interactions/s
 - · No machine limits hit
 - gather/scatter
- Random access:
 - Performance drops by factor 7
- \cdot Spatial sort fixes the problem



- \cdot Fixing the data access pattern is not always easy
- Involves usually some form of sorting the data





1. Data access pattern is crucial to performance

2. Spatial sort using space filling curves is useful technique for fixing data access patterns





- 1. Data access pattern is crucial to performance
- 2. Spatial sort using space filling curves
- 3. Intel tools (Advisor, VTune) provide useful clues to optimizing your code
 - Hot spots
 - The nature of hot spots (data access, expensive instructions, ...)
 - Issues with vectorization



- 1. Data access pattern is crucial to performance
- 2. Spatial sort using space filling curves is useful
- 3. Intel tools (Advisor, VTune) provide useful clues
- 4. Fortran has many advantages for programming number-crunching routines
 - F2py for producing python modules



Python/Fortran/C++

Python/Numpy

- verify code correctness
- · generate FCC lattice
- generate arrays filled with random numbers
- generate permutations
- zero accelerations between time steps
- build Verlet list
- spatial sort of atom property arrays
- define coarse computational strategy and data structures
- control and initialize the experiments
- plot results

code **80**% cputime 5%

code 10% cputime **90**%

Speedup of 1200x !

10% cputime 5%

Fortran

- Lennard-Jones potential
- · Lennard-Jones forces
- iterate over Verlet list and compute interactions
- iterate over array and compute interactions (baselines)

C++

- compute hilbert indices
- build Verlet list
 - iterate over Verlet list and compute inter– actions using simd libraries



- 1. Data access pattern is crucial to performance
- 2. Spatial sort using space filling curves is useful
- 3. Intel tools (Advisor, VTune) provide useful clues
- 4. Fortran has many advantages
- 5. Use a simple but relevant baseline that you understand to judge the performance of your code and direct your efforts





- 1. Data access pattern is crucial to performance
- 2. Spatial sort using space filling curves is useful
- 3. Intel tools (Advisor, VTune) provide useful clues
- 4. Fortran has many advantages
- 5. Use a simple relevant baseline

6. Be aware of the machine limits

- · Bandwidth
- Peak performance
- Roofline model



- 1. Data access pattern is crucial to performance
- 2. Spatial sort using space filling curves is useful
- 3. Intel tools (Advisor, VTune) provide useful clues
- 4. Fortran has many advantages
- 5. Use a simple relevant baseline
- 6. Be aware of the machine limits
- 7. SIMD libraries are useful
 - · Vc, Boost.simd



1. Data access pattern is crucial to performance

- 2. Spatial sort using space filling curves is useful
- 3. Intel tools (Advisor, VTune) provide useful clues
- 4. Fortran has many advantages
- 5. Use a simple relevant baseline
- 6. Be aware of the machine limits
- 7. Simd libraries are useful

Thank you

You are always welcome to discuss your (computational) problems