arm

Meet the experts

Optimize HPC across platforms - Vectorization, Why, When, How…

- Phil Ridley

- 29 May 2018

9 Step guide: optimizing high performance applications

arm

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster. This pragmatic, 9 Step best practice guide will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.

1 Bugs

- ✔ Correct application.

2 Analyze before you optimize

- ✔ Measure all performance aspects. You can't fix what you can't see.
- ✔ Prefer real workloads over artificial tests.

3 I/O

- ✔ Discover lines of code spending a long time in I/O.
- ✔ Trace and debug slow access patterns.

4 Workload

- ✔ Detect issues with balance.
- ✔ Slow communication calls and processes. Dive into partitioning code.

5 Communication

- ✔ Track communication performance.
- ✔ Discover which communication calls are slow and why.

6 Memory

- ✔ Reveal lines of code bottlenecked by memory access times.
- ✔ Trace allocation and use of hot data structures.

7 Cores

- ✔ Discover synchonization overhead and core utilization.
- ✔ Synchronization-heavy code and implicit barriers are revealed.

8 Vectorization

- ✔ Understand numerical intensity and vectorization level.
- ✔ Hot loops, unvectorized code and GPU performance revealed.

9 Verification

- ✔✔ Validate corrections and optimal Performance.

Key:

✔ arm PERFORMANCE REPORTS

✔ arm FORGE

arm

Computational Intensity

"My program is doing a lot of computation … How do I make it go faster"

```
…
DO k=y_min-2,y_max+2
        DO j=x_min-2,x_max+2
          pre_vol(j,k)=volume(j,k)+(vol_flux_x(j+1,k  )-
vol_flux_x(j,k)+vol_flux_y(j  ,k+1)-vol_flux_y(j,k))
          post_vol(j,k)=pre_vol(j,k)-(vol_flux_x(j+1,k  )-
vol_flux_x(j,k))
        ENDDO
      ENDDO
…
```

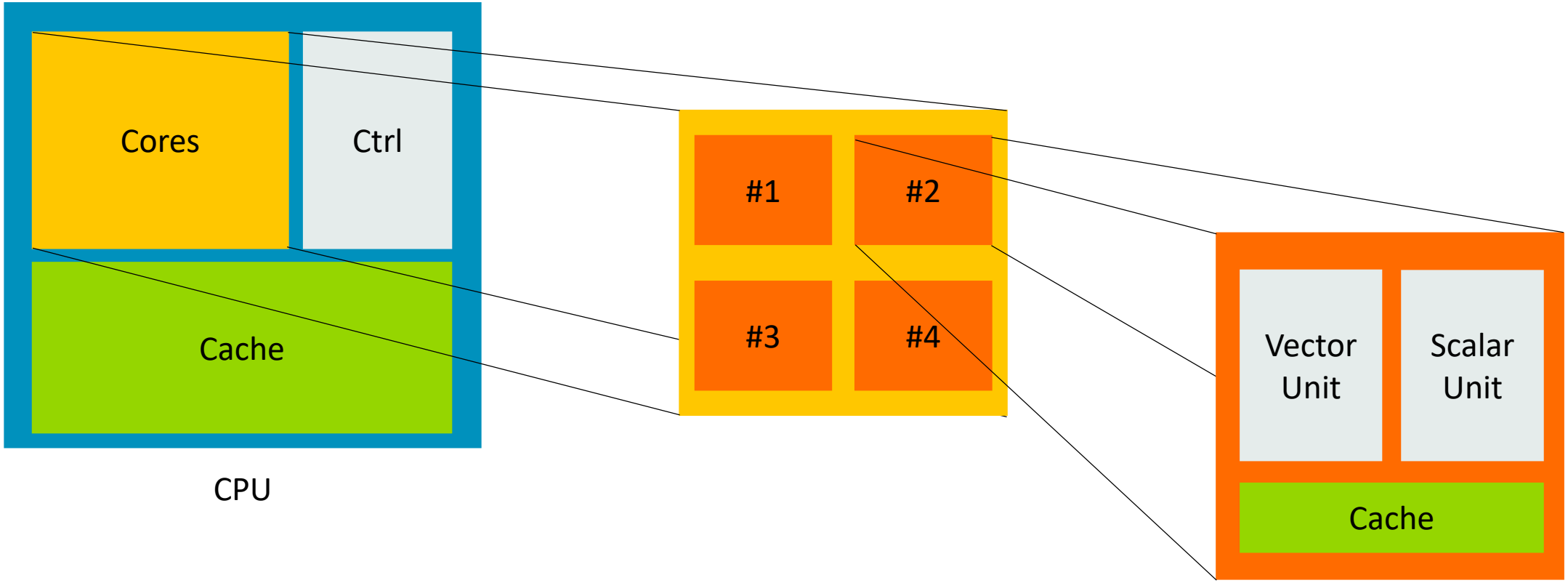
Example with modified version of CloverLeaf

- non-threaded version without OpenMP
- MPI, no IO

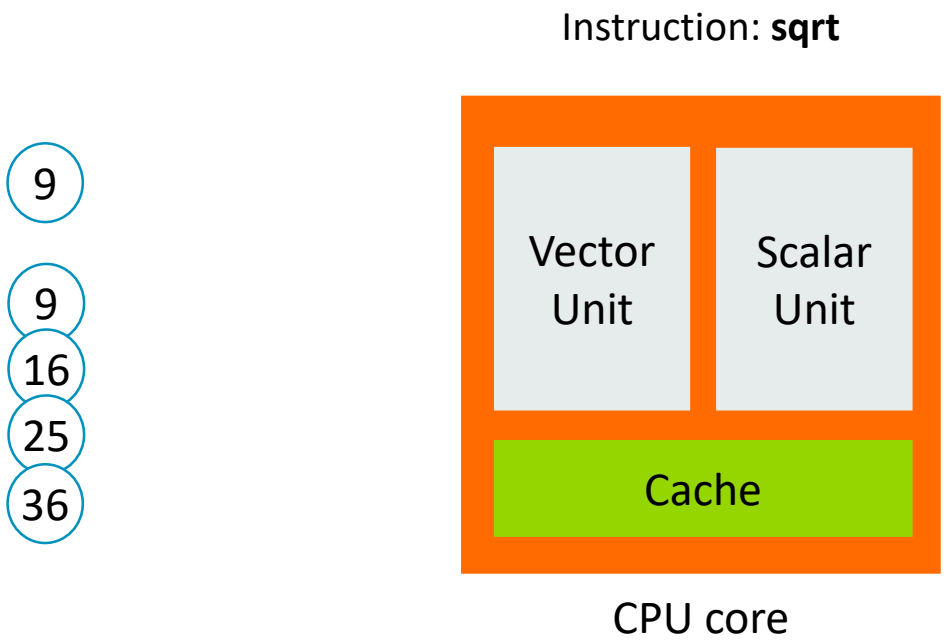
Outline

- What is vectorization?
- Why is vectorization important?
- When should I vectorize?
- How do I vectorize?

What? Vector Units



Vectorization / SIMD



Intel® AVX2: 256-bit vector unit ➔ 8 SP / 4 DP
Intel® AVX-512: 512-bit vector unit ➔ 16 SP / 8 DP
Arm® NEON: 128-bit vector unit ➔ 4 SP / 2 DP

Identifying the amount of vectorized code

- Arm Performance Reports is an application reporting tool for HPC
  - Easy to use: no re-compiling required
  - Gives a comprehensible and readable summary of the application behavior

arm PERFORMANCE

Executable: MADbench2
Resources: 16 processes, 1 node
Machine: sandybridge2
Start time: Mon Nov 4 12:27:50 2013

CPU
I/O

Summary: MADbench2 is I/O-bound in this configuration

The total wallclock time was spent as follows:

CPU	4.8%	Time spent running application code. High values are usually good. This is **low**; it may be worth improving I/O performance first.
MPI	41.3%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.
I/O	53.9%	Time spent in filesystem I/O. High values are usually bad. This is **high**; check the I/O breakdown section for optimization advice.

This application run was I/O-bound. A breakdown of this time and advice for investigating further is in the I/O section below.

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time was spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

All of the time is spent in collective calls with a very low transfer rate. This suggests a significant load imbalance is causing synchronization overhead. You can investigate this further with an MPI profiler.

I/O

A breakdown of how the 53.9% total I/O time was spent:

Time in reads	3.7%
Time in writes	96.3%
Estimated read rate	272 Mb/s
Estimated write rate	7.06 Mb/s

Most of the time is spent in write operations, which have a very low transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	160 Mb
Peak process memory usage	173 Mb
Peak node memory usage	17.2%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.

CPU

A breakdown of how the 4.8% total CPU time was spent:

Scalar numeric ops	4.9%
Vector numeric ops	0.1%
Memory accesses	95.0%
Other	0.0

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time was spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

Analyze the results

- Running Performance Reports with CloverLeaf using 8 MPI tasks indicates that:
  - Time spent in scalar ops is 14.7%
  - Time spent in vector ops 18.9%

Summary: clover_leaf is Compute-bound in this configuration

Compute	93.4%	Time spent running application code. High values are usually good. This is **very high**; check the CPU performance section for advice
MPI	6.6%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process count
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 93.4% CPU time:

Scalar numeric ops	14.7%
Vector numeric ops	18.9%
Memory accesses	66.3%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

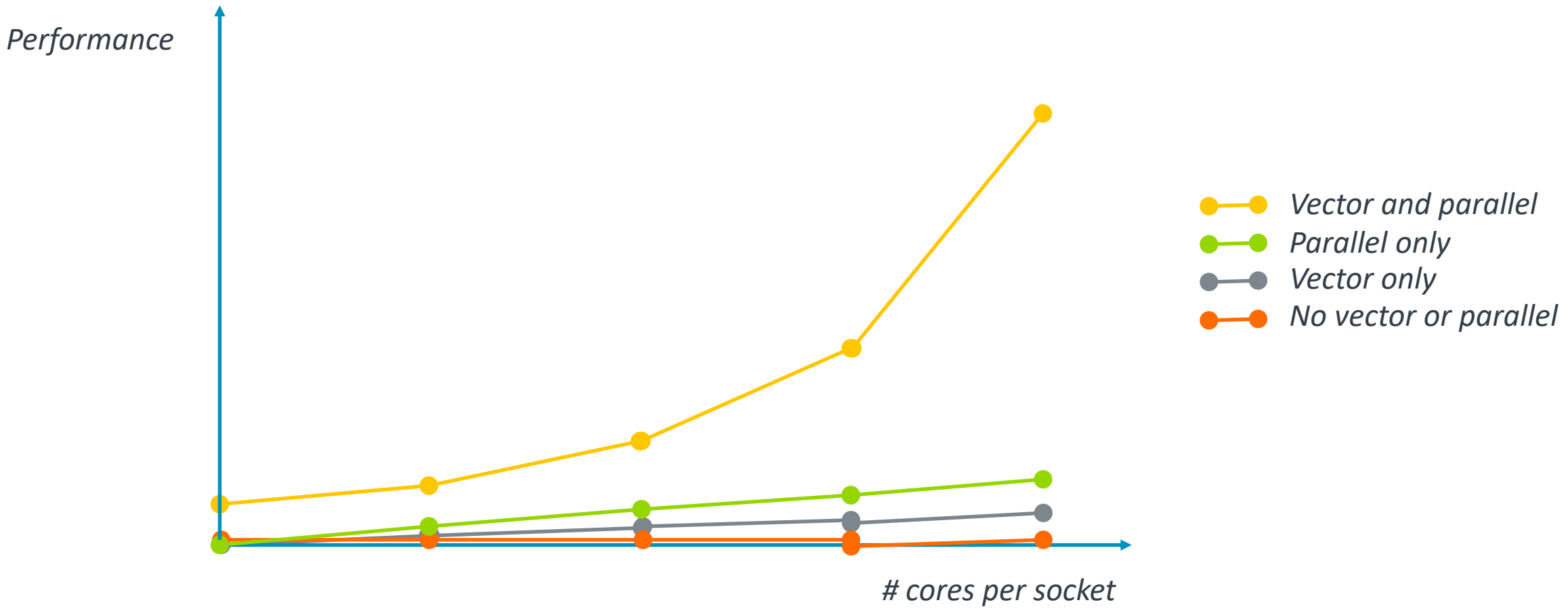
MPI

A breakdown of the 6.6% MPI time:

Time in collective calls	20.9%
Time in point-to-point calls	79.1%
Effective process collective rate	1.55 kB/s
Effective process point-to-point rate	33.1 MB/s

Most of the time is spent in point-to-point calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

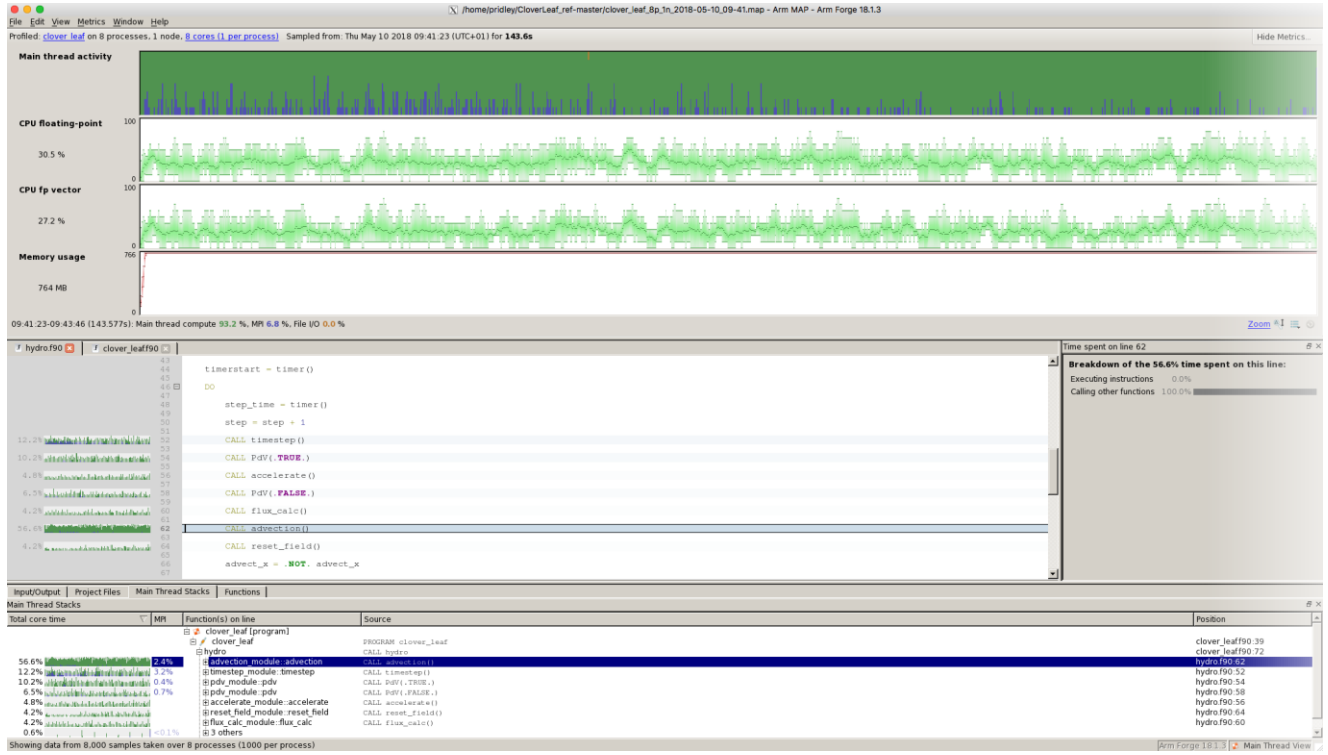
Why? Performance lies in the software



When? Time to use a profiler

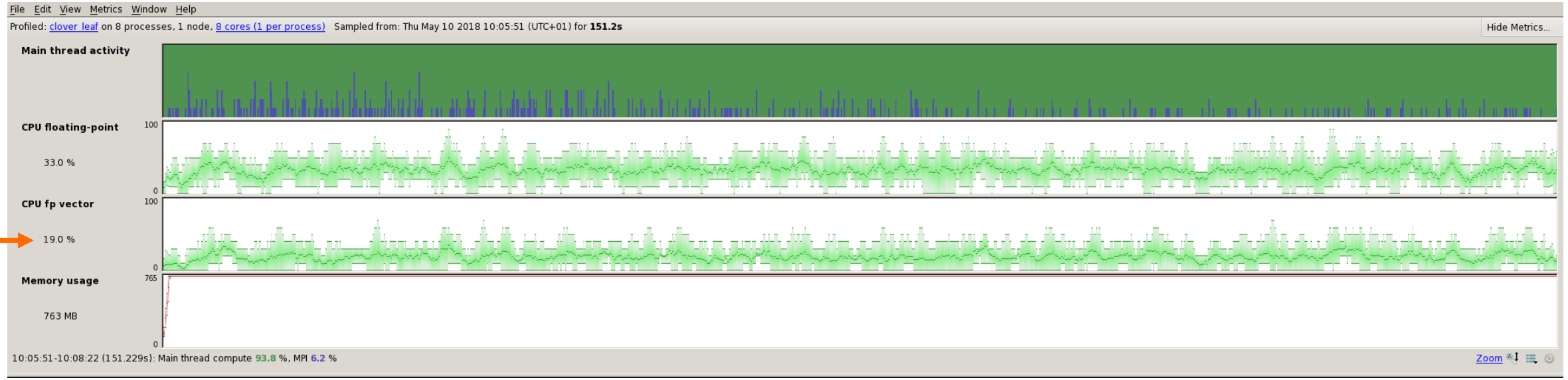
- Arm MAP is a lightweight multi-node profiling tool
  - Compiling with debugging flag required
  - Shows processes and threads activity over time
  - Source code is annotated
  - Information aggregated by stacks and function

- **Compute**, **IO** and **MPI**



How much of the code is vectorized?

Profiled: clover_leaf on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Thu May 10 2018 10:05:51 (UTC+01) for **151.2s**



Main Thread Stacks

Total core time	MPI	Function(s) on line	Source
		clover_leaf [program]	
		clover_leaf	PROGRAM clover_leaf
53.3%	2.0%	advection_module::advection	CALL advection()
16.5%	2.8%	timestep_module::timestep	CALL timestep()
10.2%	0.5%	pdv_module::pdv	CALL PdV(.TRUE.)
6.4%	0.7%	pdv_module::pdv	CALL PdV(.FALSE.)
4.9%		accelerate_module::accelerate	CALL accelerate()
4.2%		reset_field_module::reset_field	CALL reset_field()
3.4%		flux_calc_module::flux_calc	CALL flux_calc()
1.2%	0.2%	4 others	

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source
		clover_leaf [program]	
		clover_leaf	PROGRAM clover_leaf
		hydro	CALL hydro
		advection_module::advection	CALL advection()
		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
11.1%		advec_mom_kernel_mod::advec_...	CALL advec_mom_kernel(chunk%tiles(tile)%t_xmin, &
10.2%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
9.8%		advec_cell_driver_module::advec_...	CALL advec_cell_driver(tile,sweep_number,direction)
9.1%		advec_cell_driver_module::advec_...	CALL advec_cell_driver(tile,sweep_number,direction)
7.1%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
6.6%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
1.7%	1.7%	update_halo_module::update_halo	CALL update_halo(fields,2)
0.9%	0.7%	2 others	
12.2%	3.2%	timestep_module::timestep	CALL timestep()
10.2%	0.4%	pdv_module::pdv	CALL PdV(.TRUE.)
6.5%	0.7%	pdv_module::pdv	CALL PdV(.FALSE.)
4.8%		accelerate_module::accelerate	CALL accelerate()
4.2%		reset_field_module::reset_field	CALL reset_field()
4.2%		flux_calc_module::flux_calc	CALL flux_calc()
0.6%	<0.1%	3 others	
0.6%	0.1%	1 other	

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Where is the code vectorized?

advec_mom_driver.f90 | advec_mom_kernel.f90

```
150              dif=donor
151            ELSE
152              upwind=j-1
153              donor=j
154              downwind=j+1
155              dif=upwind
156            ENDIF
157            sigma=ABS(node_flux(j,k))/(node_mass_pre(donor,k
158            width=celldx(j)
159            vdiffuw=vel1(donor,k)-vel1(upwind,k)
160            vdiffdw=vel1(downwind,k)-vel1(donor,k)
161            limiter=0.0
162            IF(vdiffuw*vdiffdw.GT.0.0)THEN
163              auw=ABS(vdiffuw)
164              adw=ABS(vdiffdw)
165              wind=1.0_8
166              IF(vdiffdw.LE.0.0) wind=-1.0_8
167              limiter=wind*MIN(width*((2.0_8-sigma)*adw/widt
168            ENDIF
169            advec_vel_s=vel1(donor,k)+(1.0-sigma)*limiter
170            mom_flux(j,k)=advec_vel_s*node_flux(j,k)
171          ENDDO
172        ENDDO
173        !$OMP END DO
174        !$OMP DO
```

Time spent on line 159

Breakdown of the 0.1% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	63.6%
Vector floating point	0.0%
Scalar integer	18.2%
Vector integer	0.0%
Memory access*	81.8%
Branch	0.0%
Other instructions	0.0%

* 18.2% memory access instructions, 63.6% implicit memory accesses in other instructions, also counted in their categories

Follow Performance Reports advice

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source
		clover_leaf [program]	
		clover_leaf	PROGRAM clover_leaf
		hydro	CALL hydro
		advection_module::advection	CALL advection()
		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
11.1%		advec_mom_kernel_mod::advec_...	CALL advec_mom_kernel(chunk%tiles(tile)%t_xmin, &
10.2%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
9.8%		advec_cell_driver_module::advec_...	CALL advec_cell_driver(tile,sweep_number,direction)
9.1%		advec_cell_driver_module::advec_...	CALL advec_cell_driver(tile,sweep_number,direction)
7.1%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
6.6%		advec_mom_driver_module::advec...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
1.7%	1.7%	update_halo_module::update_halo	CALL update_halo(fields,2)
0.9%	0.7%	2 others	
12.2%	3.2%	timestep_module::timestep	CALL timestep()
10.2%	0.4%	pdv_module::pdv	CALL PdV(.TRUE.)
6.5%	0.7%	pdv_module::pdv	CALL PdV(.FALSE.)
4.8%		accelerate_module::accelerate	CALL accelerate()
4.2%		reset_field_module::reset_field	CALL reset_field()
4.2%		flux_calc_module::flux_calc	CALL flux_calc()
0.6%	<0.1%	3 others	
0.6%	0.1%	1 other	

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Follow Performance Reports advice

advec_mom_kernel.f90

```
...
144     DO k=y_min,y_max+1
145       DO j=x_min-1,x_max+1
146         IF(node_flux(j,k).LT.0.0)THEN
147           upwind=j+2
148           donor=j+1
149           downwind=j
150           dif=donor
151         ELSE
152           upwind=j-1
153           donor=j
154           downwind=j+1
155           dif=upwind
156         ENDIF
157         sigma=ABS(node_flux(j,k))/(node_mass_pre(donor,k))
158         width=celldx(j)
159         vdiffuw=vel1(donor,k)-vel1(upwind,k)
160         vdiffdw=vel1(downwind,k)-vel1(donor,k)
...
```

-fopt-info-vec-missed

```
advec_mom_kernel.f90:145: note: not vectorized: control flow in loop.
advec_mom_kernel.f90:145: note: bad inner-loop form.
advec_mom_kernel.f90:145: note: not vectorized: Bad inner loop.
advec_mom_kernel.f90:145: note: bad loop form.
Analyzing loop at advec_mom_kernel.f90:145

advec_mom_kernel.f90:145: note: not vectorized: control flow in loop.
advec_mom_kernel.f90:145: note: bad loop form.
```

How well is the compiler vectorizing?

advec_mom_kernel.f90

```
...
144     DO k=y_min,y_max+1
145       DO j=x_min-1,x_max+1
146         IF(node_flux(j,k).LT.0.0)THEN
147           upwind=j+2
148           donor=j+1
149           downwind=j
150           dif=donor
151         ELSE
152           upwind=j-1
153           donor=j
154           downwind=j+1
155           dif=upwind
156         ENDIF
157         sigma=ABS(node_flux(j,k))/(node_mass_pre(donor,k))
158         width=celldx(j)
159         vdiffuw=vel1(donor,k)-vel1(upwind,k)
160         vdiffdw=vel1(downwind,k)-vel1(donor,k)
...
```

```
-qopt-report=2

LOOP BEGIN at advec_mom_kernel.f90(145,9)
 <Peeled loop for vectorization>
   remark #25456: Number of Array Refs Scalar Replaced In Loop: 2
 LOOP END

 LOOP BEGIN at advec_mom_kernel.f90(145,9)
   remark #15300: LOOP WAS VECTORIZED
 LOOP END

 LOOP BEGIN at advec_mom_kernel.f90(145,9)
 <Remainder loop for vectorization>
 LOOP END
```

Analyze the results

- Running Performance Reports with CloverLeaf using 8 MPI tasks indicates that:
  - Time spent in scalar ops is 4.8%
  - Time spent in vector ops 28.2%

Summary: clover_leaf is Compute-bound in this configuration

Compute	92.9%	Time spent running application code. High values are usually good. This is **very high**; check the CPU performance section for advice
MPI	7.1%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process count
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 92.9% CPU time:

Scalar numeric ops	4.8%
Vector numeric ops	28.2%
Memory accesses	67.0%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

MPI

A breakdown of the 7.1% MPI time:

Time in collective calls	24.4%
Time in point-to-point calls	75.6%
Effective process collective rate	1.35 kB/s
Effective process point-to-point rate	33.9 MB/s

Most of the time is spent in point-to-point calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

Where is the code vectorized?

Profiled: clover leaf on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Thu May 10 2018 09:41:23 (UTC+01) for **143.6s**

advec_mom_driver.f90 | advec_mom_kernel.f90

```
152            upwind=j-1
153            donor=j
154            downwind=j+1
155            dif=upwind
156          ENDIF
157          sigma=ABS(node_flux(j,k))/(node_mass_pre(donor,k))
158          width=celldx(j)
159          vdiffuw=vel1(donor,k)-vel1(upwind,k)
160          vdiffdw=vel1(downwind,k)-vel1(donor,k)
161          limiter=0.0
162          IF(vdiffuw*vdiffdw.GT.0.0)THEN
163            auw=ABS(vdiffuw)
164            adw=ABS(vdiffdw)
165            wind=1.0_8
166            IF(vdiffdw.LE.0.0) wind=-1.0_8
167            limiter=wind*MIN(width*((2.0_8-sigma)*adw/width+(1.0_8+sigma)*auw/celldx(dif))/6.0_8,auw,adw)
168          ENDIF
169          advec_vel_s=vel1(donor,k)+(1.0-sigma)*limiter
170          mom_flux(j,k)=advec_vel_s*node_flux(j,k)
171        ENDDO
172      ENDDO
173      !$OMP END DO
174      !$OMP DO
175      DO k=y_min,y_max+1
176        DO j=x_min,x_max+1
177          vel1 (j,k)=(vel1 (j,k)*node_mass_pre(j,k)+mom_flux(j-1,k)-mom_flux(j,k))/node_mass_post(j,k)
178        ENDDO
179      ENDDO
180    !$OMP END DO
181    ELSEIF(direction.EQ.2)THEN
182      IF(which_vel.EQ.1)THEN
183        !$OMP DO
184        DO k=y_min-2,y_max+2
185          DO j=x_min,x_max+1
```

Time spent on line 159

Breakdown of the 0.4% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	28.6%
Scalar integer	0.0%
Vector integer	0.0%
Memory access	68.6%
Branch	0.0%
Other instructions	2.9%

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source
		clover_leaf [program]	
		clover_leaf	PROGRAM clover_leaf
		hydro	CALL hydro
56.6%	2.4%	advection_module::advection	CALL advection()
12.2%	3.2%	timestep_module::timestep	CALL timestep()
10.2%	0.4%	pdv_module::pdv	CALL PdV(.TRUE.)
6.5%	0.7%	pdv_module::pdv	CALL PdV(.FALSE.)
4.8%		accelerate_module::accelerate	CALL accelerate()
4.2%		reset_field_module::reset_field	CALL reset_field()
4.2%		flux_calc_module::flux_calc	CALL flux_calc()
0.6%	<0.1%	3 others	

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Main Thread Stacks

Total core time	MPI		
10.9%		advec_mom_driver_module::advec_...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
9.7%		advec_mom_driver_module::advec_...	CALL advec_mom_driver(tile,xvel,direction,sweep_number)
8.9%		advec_cell_driver_module::advec_ce...	CALL advec_cell_driver(tile,sweep_number,direction)
8.5%		advec_cell_driver_module::advec_ce...	CALL advec_cell_driver(tile,sweep_number,direction)
6.8%		advec_mom_driver_module::advec_...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
6.2%		advec_mom_driver_module::advec_...	CALL advec_mom_driver(tile,yvel,direction,sweep_number)
1.5%	1.4%	update_halo_module::update_halo	CALL update_halo(fields,2)
0.8%	0.6%	2 others	
16.5%	2.8%	timestep_module::timestep	CALL timestep()
10.2%	0.5%	pdv_module::pdv	CALL PdV(.TRUE.)
6.4%	0.7%	pdv_module::pdv	CALL PdV(.FALSE.)
4.9%		accelerate_module::accelerate	CALL accelerate()
4.2%		reset_field_module::reset_field	CALL reset_field()
3.4%		flux_calc_module::flux_calc	CALL flux_calc()
1.2%	0.2%	4 others	

How?

- Different compilers may have different capabilities, but here are guidelines
  - Remove conditionals inside loop
  - Make sure that loop size is known on entry
  - Pay attention to work on contiguous, unit-stride arrays
  - Remove data dependencies to enable vectorization
  - Use compiler directives to force loop vectorization

Conclusion

- Vectorizing an application is a difficult task
- Arm Performance Reports and Arm MAP make it easier
  - Analyze application efficiency and get advices with Performance Reports
  - Identify bottlenecks and line by line performance with MAP
- Figure out quickly if your application uses vectorization
- Find candidates for vectorization
- Inspect vectorization over time

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה

arm