Vectorization Efficiency Metrics

Zakhar A. Matveev, CJ Newburn

with input from Dmitry Prohorov, Hideki Saito

Metrics list

Vectorization Metrics

- Actual speed-up (could be: wall-clock, total, inclusive/exclusive):
  - **S = Time (Scalar Loop) / Time (Vector Loop)**
  - **Efficiency = G/Max_S , Max_S <= Max_VL**
- Gain/Efficiency :
  - **G = Scalar Loop Cost (cycles) / Vector Loop Cost (cycles)**
  - **Efficency = G/Max_Gain , Max_G <= Max_VL**

  How much faster the vector iterations are? Reflects quality of compiler code-generation.

  *Sh*ould be equal to **S** in case of VPU-bound codes

Vectorization Metrics

- Path Reduction:

  Scalar Loop Path (# instructions) / Vector Loop Path (# instructions)

  Gives a sense of the fraction of non-vector (overhead) instructions in a loop

- Vector Utilization / Intensity ("elements active") – only works on KNC

  VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED

  Fraction of vector instructions that do work on vector registers. Reflect vector registers utilization

  If a mask bit is set for an element, it was presumed to be used. Drops for branchy if-else codes

Intensity/Utilization

Advantage:

- Dynamically “measures” fraction of vector instructions that do work (mask-aware)

Disadvantages:

- Only available on KNC (other metrics could be computed on IVB or Broadwell)
- More work/utilization doesn’t mean more speed-up (if you care)
  - Some code may have good vector utilization, but scalar version could be faster than it!
  - Shifts/shuffles/“misc.”, prefetch instructions are counted “inappropriately”.
- Assumption: Mask bits are only set for elements in which useful work is done
  - This is false: Extra mask bits can be set, as long as there are no side effects.
- This is per binary loop, so separate values for peel/remainder..

Gain/Efficiency Estimate

Advantage:

- Accounts all nuances of vector vs. scalar assembly and maps it to speed-up

Disadvantages:

- Usually not available if you program in assembly/intrinsics.
- This is code generation performance model, not measurement. This doesn't account dynamic mask *values* as well as other dynamic data (trip counts).

Advisor Gain/Efficiency:

- Recalculate (calibrate) Compiler Gain/Efficiency taking into account dynamic knowledge of trip counts, peel-remainder times.
- But limited to Xeon right now.

Tools to calculate metrics

Intensity/Utilization: VTune Amplifier XE 2016 Beta for KNC

General Exploration General Exploration viewpoint (change) Intel

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Top-down Tree | Tasks and Frames

Grouping: Function / Call Stack

Function / Call Stack	Cache Usage			Vectorization Usage		
	s	L1 Hit Ratio	Estimated Latenc...	Vectorization Intensity	L1 Compute to D...	L2 Compute...
[Loop at line 137 in single_iteration]	000	0.684	162.181	14.787	13.832	43.729
[Outside any loop]	000	0.974	417.034	9.500	0.084	3.276
[Loop at line 81 in single_iteration]	000	0.964	448.529	1.688	0.688	31.765
[Loop@0x4e384 in func@0x4e2da]	000	0.999	14,025.889	0.000	0.000	0.000
[Loop@0x52ec8 in _IO_vfscanf]	0	0.000	0.000	0.000	0.000	0.000
[Loop at line 195 in compute_tran_temp]	0	0.000	0.000	0.000	0.000	0.000
[Loop@0x53450 in _IO_vfscanf]	0	1.000		0.000	0.000	0.000

Part of Intel Parallel Studio XE 2016

VTune: General Exploration Analysis Type

Gain/Efficiencty: Intel Compiler (>=15.x)

```
LOOP BEGIN at C:\work\!LBZ\DL_MESO_LBE_2.6 - Copy\DL_MESO_LBE_2.6 - Copy\lbpSUB.cpp(1090,7)
   remark #25408: memset generated
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at C:\work\!LBZ\DL_MESO_LBE_2.6 - Copy\DL_MESO_LBE_2.6 - Copy\lbpSUB.cpp(1090,7)
   <Peeled>
      remark #25015: Estimate of max trip count of loop=12
   LOOP END

   LOOP BEGIN at C:\work\!LBZ\DL_MESO_LBE_2.6 - Copy\DL_MESO_LBE_2.6 - Copy\lbpSUB.cpp(1090,7)
      remark #15388: vectorization support: reference pt2 has aligned access   [ C:\work\!LBZ\DL_MESO_LBE_2.6 -
      remark #15305: vectorization support: vector length 4
      remark #15399: vectorization support: unroll factor set to 2
      remark #15300: LOOP WAS VECTORIZED
      remark #15442: entire loop may be executed in remainder
      remark #15449: unmasked aligned unit stride stores: 1
      remark #15475: --- begin vector loop cost summary ---
      remark #15476: scalar loop cost: 4
      remark #15477: vector loop cost: 3.500
      remark #15478: estimated potential speedup: 1.110
      remark #15488: --- end vector loop cost summary ---
      remark #25015: Estimate of max trip count of loop=1
   LOOP END

   LOOP BEGIN at C:\work\!LBZ\DL_MESO_LBE_2.6 - Copy\DL_MESO_LBE_2.6 - Copy\lbpSUB.cpp(1090,7)
   <Remainder>
      remark #25015: Estimate of max trip count of loop=12
   LOOP END
LOOP END
```

Intel Compiler:
-O2 -opt-report5

Part of Intel Parallel Studio XE 2016

Gain/Efficiencty: Advisor XE 2016 Beta (for Xeon only)

Elapsed time: 5,46s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Loops	Loop Type	Self Time	Vectorized Loops		
			Vecto...	Efficiency	Estimated Gain
[loop in fCollisionBGK at lbpBGK.cpp:840]	Vectorized: Expand	0,020s	AVX	100%	2,05
[loop in fGetFracSite at lbpGET.cpp:152]	Vectorized: Expand	0,030s	AVX	36%	2,90
[loop in fGetOneMassSite at lbpGET.cpp: ...	Vectorized: Expand	0,089s	AVX	36%	2,86
[loop in fGetEquilibriumF at lbpSUB.cpp:729]	Vectorized: Collapse	0,579s	AVX	25%	2,00
[loop in fGetEquilibriumF at lbpSUB.cpp: ...	Vectorized (Body)	0,431s	AVX		
[loop in fGetEquilibriumF at lbpSUB.cpp: ...	Vectorized (Remainder)	0,087s	AVX		
[loop in fGetEquilibriumF at lbpSUB.cpp: ...	Remainder	0,061s			
[loop in fPropagationSwap at lbpSUB.cpp:1 ...	Vectorized (Body)	1,259s	AVX	~14%	0,54

Part of Intel Parallel Studio XE 2016

Advisor : Survey Analysis Type

Some word on methodology..

WHAT to measure?

Actual Speed-up vs. Efficiency vs. Intensity. ??

- *Measuring all and comparing results –*

  is "useful enough" exercise already.

- Normally stick with at least one of them for workshop exercises.

Kernel vs. Sub-part vs. Workload ??

- Per-workload speed-up/efficiencies are lower than per-kernel (Amdahl's law)
- Both are important to understand, but don't mix them up!
- For big HPC codes you rarely even look into everything. Define sub-set.
- Measuring / establishing proper baselines is very important/not-trivial itself

Some take-aways

Vectorization efficiency/gain

- Take it as input, but treat it as performance *estimate*
- Use Advisor if you want to overcome some of “static code-generation knowledge” limits

Vectorization intensity

- Take it as input, but *don’t treat it as accurate*:
  - Low intensity definitely means you have an issue. Otherwise – who knows.
- If higher than expected, inspect code for masks that are all 1 even through conditionals
- The VPU_ELEMENTS_ACTIVE won’t be available for *anything* other than KNC

Don’t compare apples with oranges (kernel and workload, etc)

Don’t mix up dimensional and non-dimensional metrics