AUTO-VECTORIZATION OF LOOPS ON INTEL 64 AND INTEL XEON PHI: ANALYSIS AND EVALUATION*

Olga V. Moldovanova[1,2], **Mikhail G. Kurnosov**[1,2]

[1] Rzhanov Institute of Semiconductor Physics, Siberian Branch of Russian Academy of Sciences, Novosibirsk, Russia
[2] Siberian State University of Telecommunications and Information Sciences, Novosibirsk, Russia
{ovm, mkurnosov}@isp.nsc.ru, {ovm, mkurnosov}@sibguti.ru

AUTO-VECTORIZATION OF LOOPS

AUTO-VECTORIZATION – EASE OF USE, HIGH CODE PORTABILITY

- We studied the effectiveness of auto-vectorizing capabilities of modern compilers
- The main goal is to identify classes of problem loops

Compiler	Compiler Options	W/O Vectorizer
Intel C/C++ 17.0	`-O3 -xHost -qopt-report3 -qopt-report-phase=vec,loop -qopt-report-embed`	`-no-vec`
GCC C/C++ 6.3.0	`-O3 -ffast-math -fivopts -march=native -fopt-info-vec -fopt-info-vec-missed -fno-tree-vectorize`	`-fno-tree-vectorize`
LLVM/Clang 3.9.1	`-O3 -ffast-math -fvectorize -Rpass=loop-vectorize -Rpass-missed=loop-vectorize -Rpass-analysis=loop-vectorize`	`-fno-vectorize`
PGI C/C++ 16.10 CE	`-O3 -Mvect -Minfo=loop,vect -Mneginfo=loop,vect`	`-Mnovect`

LOOPS ARE THE MOST CRITICAL PARTS OF MANY APPLICATIONS

- Auto-vectorization of loops by compilers is one of the most significant techniques of loops optimization

USED BENCHMARKS

- **Test Suite for Vectorizing Compilers** (TSVC) – 122 loops in Fortran [1]
- **Extended Test Suite for Vectorizing Compilers** (ETSVC) – 151 loops in C [2-3]

```
      do 1 nl = 1, 2 * ntimes
         do 10 i = 2, n, 2
            a(i) = a(i-1) + b(i)
10       continue
1     continue
```

```
for (int nl = 0; nl < 2 * ntimes; nl++) {
    for (int i = 1; i < n; i += 2) {
        a[i] = a[i - 1] + b[i];
    }
}
```

[1] Levine D., Callahan D., Dongarra J. ***A Comparative Study of Automatic Vectorizing Compilers*** // Journal of Parallel Computing. 1991. Vol. 17. pp. 1223–1244.
[2] Maleki S., Gao Ya. Garzarán M., Wong T., Padua D. ***An Evaluation of Vectorizing Compilers*** // Proc. of the Int. conf. on Parallel Architectures and Compilation Techniques (PACT-11), 2011. pp. 372–382.
[3] ***Extended Test Suite for Vectorizing Compilers*** // URL: http://polaris.cs.uiuc.edu/~maleki1/TSVC.tar.gz

INTEL X86 ISA VECTOR EXTENSIONS

Data Type	SPEEDUP			
	Intel SSE (128-bit registers)	**Intel AVX** (256-bit registers)	**Intel AVX-512** (512-bit registers)	**ARMv8 Scalable Vector Extension** (128-2048-bit registers, RIKEN Post-K supercomputer)
`double`	x2	x4	x8	**x32**
`float`	x4	x8	x16	**x64**
`int`	x4	x8	x16	x64
`short int`	x8	x16	x32	x128

To achieve a maximum speedup during vector processing it is necessary to consider the (micro)architectural parameters:

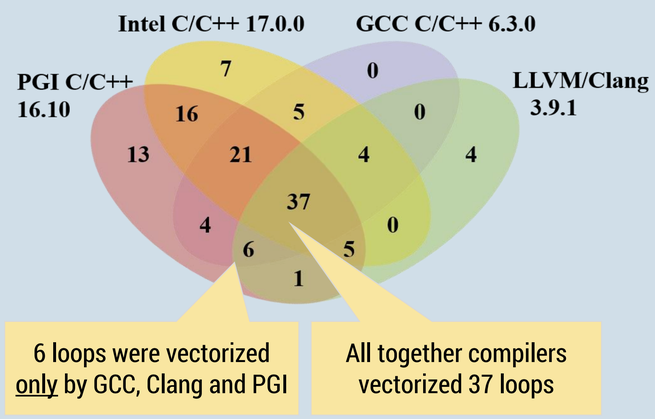
- Memory alignment: 32-byte for AVX and 64-byte for AVX-512
- Avoiding mixed usage of SSE and AVX extensions (AVX-SSE Transition Penalties)
- Parallel execution of vector instructions by several vector ALUs

An efficiently vectorized program overloads subsystems of a superscalar pipelined processor in a less degree. This is the reason of less processor energy consumption during execution of a vectorized program as compared to its scalar version [4].

[4] Jibaja I., Jensen P., Hu N., Haghighat M., McCutchan J., Gohman D., Blackburn S., McKinley K. ***Vector Parallelism in JavaScript: Language and Compiler Support for SIMD*** // Proc. of the Int. conf. on Parallel Architecture and Compilation (PACT-2015), 2015. pp. 407–418.

RESULTS OF EXPERIMENTS

- **NUMA-server**: 2 x Intel Xeon E5-2620 v4 (Broadwell, Hyper-Threading on, AVX 2.0), 64 GiB DDR4 RAM, GNU/Linux CentOS 7.3 (linux 3.10.0-514.2.2.el7 x86-64)
- **Intel Xeon Phi 3120A** co-processor (Knights Corner, 57 cores, AVX-512, 6 GiB RAM, MPSS 3.8)



6 loops were vectorized only by GCC, Clang and PGI

All together compilers vectorized 37 loops

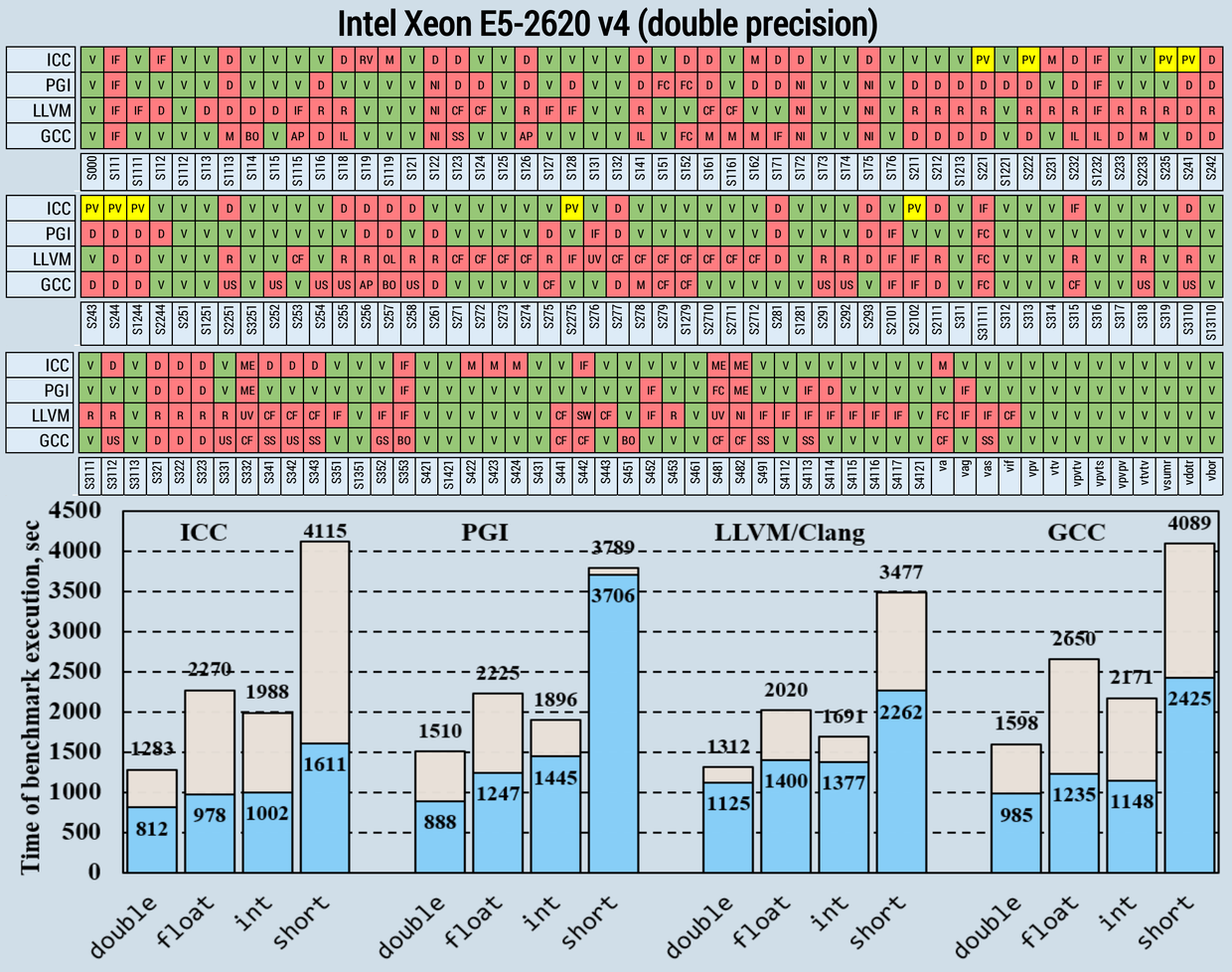
Benchmark Category / Subcategory	Number of loops	Non-vectorized loops
Dependence Analysis	**36**	**9**
Linear Dependence	14	2
Induction Variable Recognition	8	3
Nonlinear Dependence	1	1
Control Flow	3	1
Symbolics	6	2
Vectorization	**52**	**11**
Loop Distribution	3	2
Loop Interchange	6	2
Node Splitting	6	4
Scalar and Array Expansion	12	2
Control Flow	14	1
Idiom Recognition	**27**	**6**
Recurrences	3	3
Search Loops	2	1
Loop Rerolling	4	1
Reductions	15	1
Language Completeness	**23**	**2**
Nonlocal GOTO	2	2

28 loops (18.5 %) were not vectorized by any compiler!

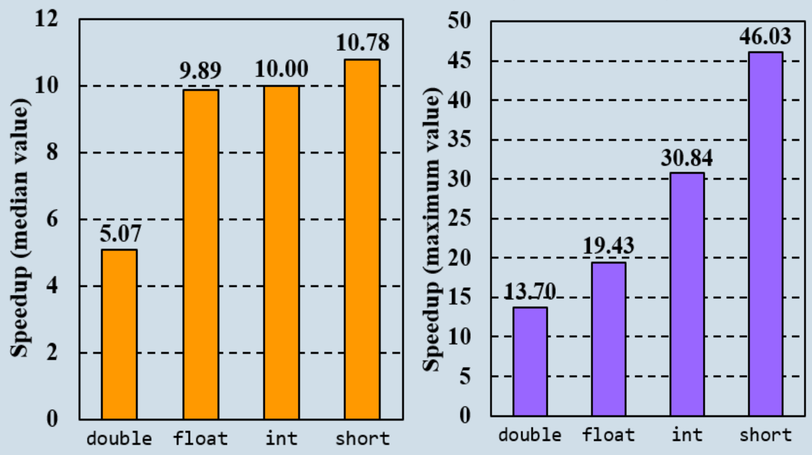
- ***Maximum speedups*** for Intel C/C++, GCC C/C++ and LLVM/Clang correspond to the loops executing reduction operations with elements of one-dimensional arrays of all data types
- For PGI C/C++ maximum speedup was achieved for the loop calculating an identity matrix for the `double` and `float` data types. And for `int` and `short` this value was obtained in the loop calculating product reduction
- The speedup value 68.0 for the `short` data type can be explained by the fact that calculations in a loop are not executed at all because of the compiler optimization
- The compilers ***failed to vectorize*** loops containing conditional and unconditional branches, function calls, induction variables, variable loop bounds and iteration count, as well as such idioms as 1st or 2nd order recurrences, search loops and loop rerolling

V	Loop is vectorized	M	Loop is multiversioned	NI	Number of iterations cannot be computed	OL	Value cannot be used outside the loop
PV	Loop is partially vectorized	BO	Bad operation or unsupported loop bound	CF	Control flow cannot be substituted for a select	UV	Loop control flow is not understood by vectorizer
RV	Remainder is not vectorized	AP	Complicated access pattern	SS	Loop is not suitable for scatter store	SW	Loop contains a switch statement
IF	Vectorization is possible but seems inefficient	R	Value that could not be identified as reduction is used outside the loop	ME	Loop with multiple exits cannot be vectorized	US	Unsupported use in statement
D	Vector dependence prevents vectorization	IL	Inner-loop count not invariant	FC	Loop contains function calls or data references that cannot be analyzed	GS	No grouped stores in basic block

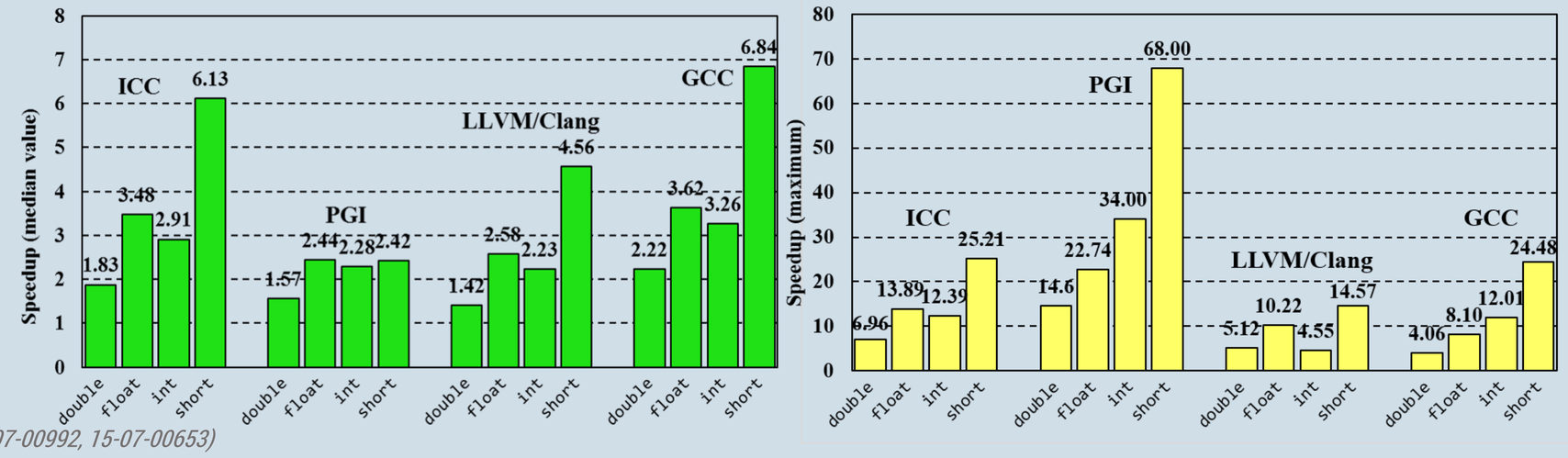
Intel Xeon E5-2620 v4 (double precision)



Intel Xeon Phi 3120A (Intel C/C++)



Intel Xeon E5-2620 v4



** This work is supported by Russian Foundation for Basic Research (projects 16-07-00992, 15-07-00653)*