Russian Supercomputing Days
Russian Supercomputing Days
September 25-26, 2017, Moscow

# Как приблизиться к пиковому значению Flops/s? Сравнение архитектур x86 и ARMv8

NATIONAL RESEARCH UNIVERSITY

**V. Nikolskiy, V. Stegailov**

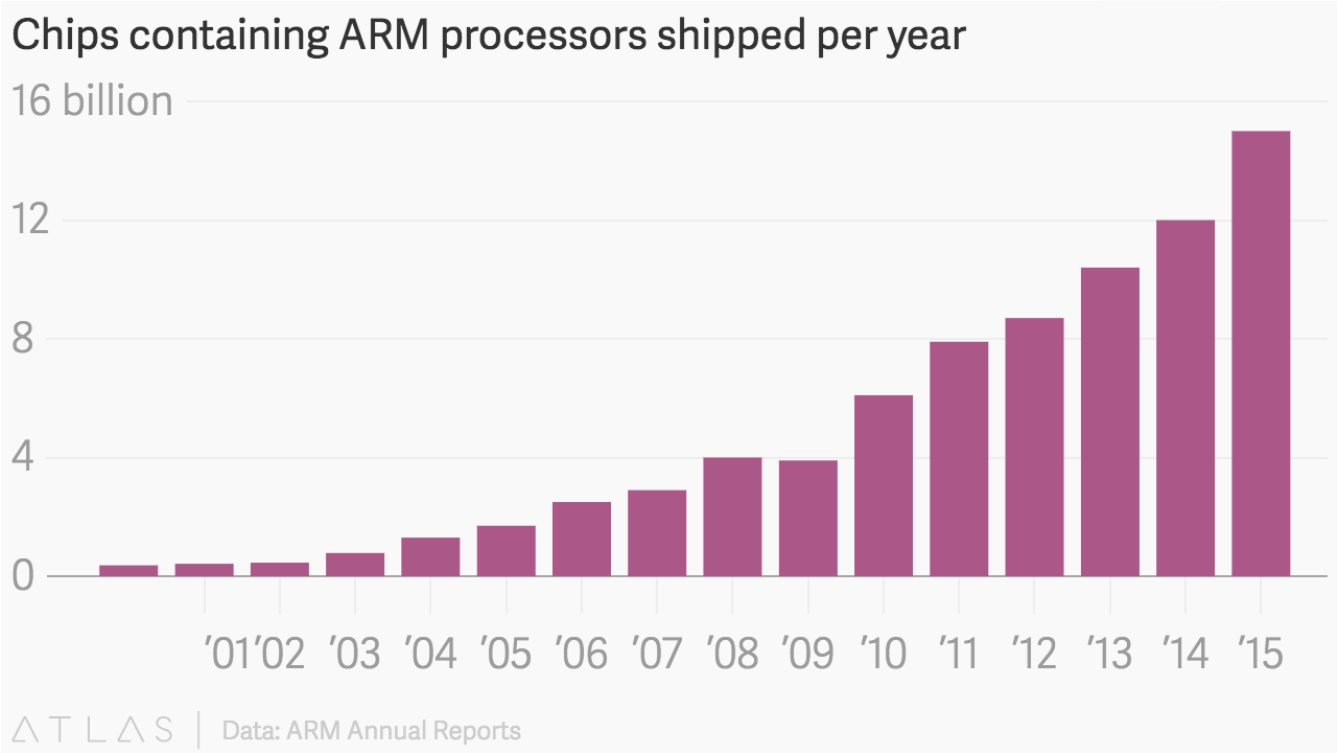
International Laboratory for Supercomputer Atomistic Modelling and Multi-scale Analysis, Higher School of Economics

National Research University Higher School of Economics Intl Lab

Supercomputer Atomistic Modelling and Multi-scale Analysis

**Web:** samma.hse.ru **Email:** {vnikolskiy, vstegailov}@hse.ru

# Mobile and embedded market leader



# Architecture evolution

| Свойство | ARMv6 | ARMv7 | ARMv8 |
|---|---|---|---|
| 64-bit | - | - | + |
| Multicore | - | + | + |
| Cache hierarchy | - | + | + |
| SP Scalar Floating-point | Optional | Optional | + |
| DP Scalar Floating-point | Optional | Optional | + |
| SP Floating-point SIMD | - | Optional | + |
| DP Floating-point SIMD | - | - | + |

# Mont-Blanc 2011-2018:

## European approach towards energy efficient high performance computation

- To produce an architecture that will provide Exascale performance using 15 to 30 times less energy
- To develop the Mont-Blanc software ecosystem, with emphasis on programmer tools and system resiliency
- To design a well-balanced architecture for an ARM based SoC or SoP capable of providing pre-exascale performance when implemented in the 2019-2020

# ARMv8 server-level processors

**Cavium** ThunderX2
- 54 cores @ 3.0 GHz (14 nm FinFET)
- 6 DDR4 controllers
- Integrated SATAv3
- Integrated PCIe Gen3

**MACOM (AppliedMicro)** X-Gene
- 32 cores @ 3.0+ GHz (16 nm FinFET)
- 8 DDR4-2667 memory channels
- 42 PCIe Gen3 lanes with eight controllers

**Qualcomm**
- 48 cores (10-nm FinFET)
- Multiple DDR4 memory controllers
- PCIe Gen3 x16
- Integrated IO and SATAv3 ports

**AMD** Opteron A1100
- 8 Cortex-A57 cores @ 2.0 GHz
- 2 DDR4-1866 memory channels
- 8 PCIe Gen3 lanes
- 14 SATA 3 (6 GB/s)

# The Scalable Vector Extension (SVE)

Aarch64 extension which expand vector up to 2048 bits

- Expand fine-grain parallelism for HPC scientific workloads
- Reduce software deployment effort
- Enables vectorization of complex data structures with non-linear access patterns
- Permits vectorization of uncounted loops with data-dependent exits

# Fujitsu Post-K

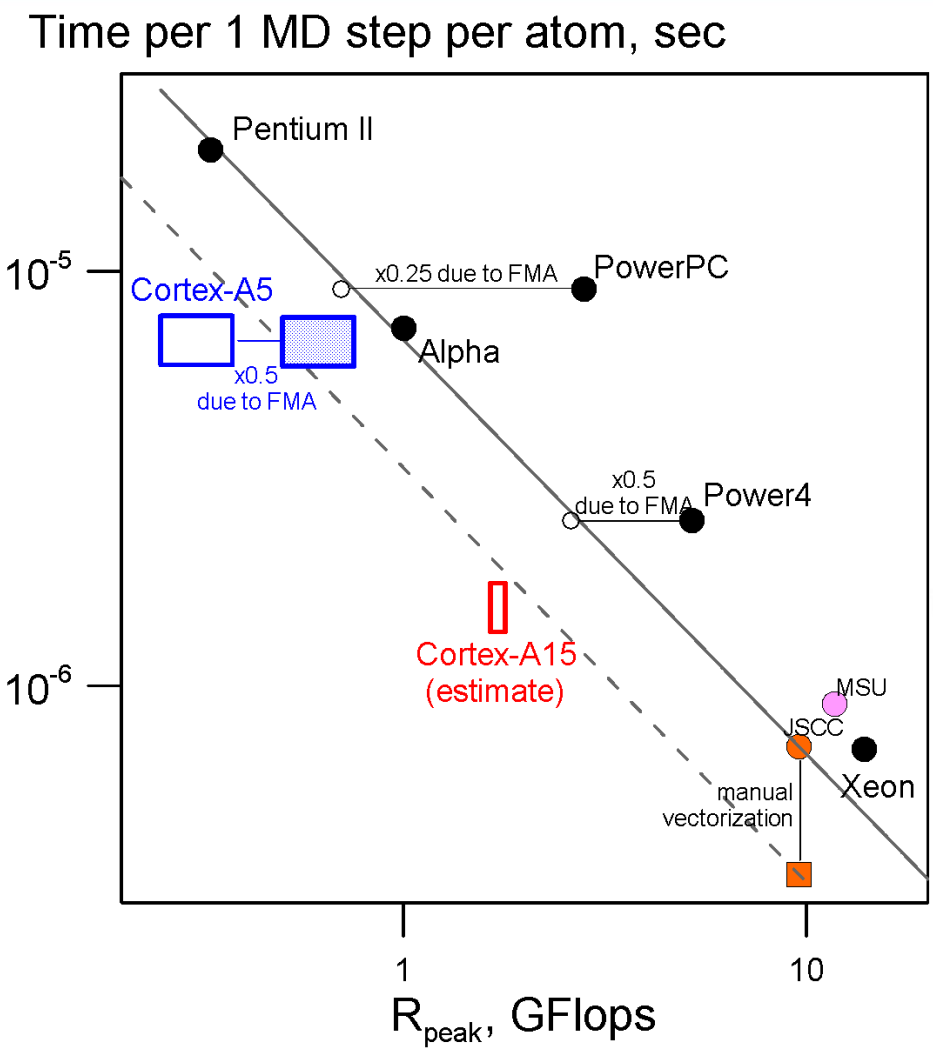
- Based on 64-bit ARMv8-A + SVE
- Exaflops supercomputer
- 6D mesh/torus interconnect “Tofu”
- 10nm FinFET

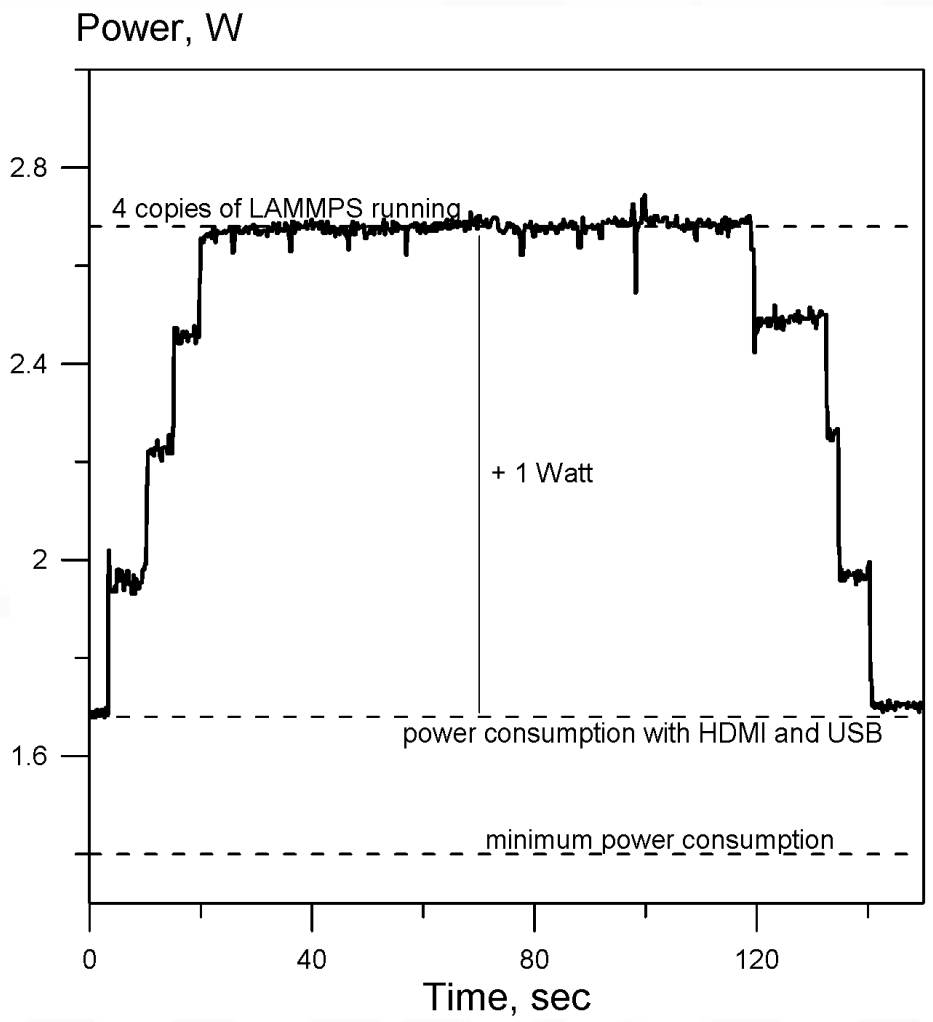
- Be completed by 2022
- Budget $910m
- Power consumption 30-40 MW

# ODROID-C1

- Quad core processor Amlogic S805 *Cortex-A5*
  - 1.5GHz
  - VFPv4 modules on each core
  - graphics accelerator Mali-450 MP2 (was not used)
- 1GB DDR3 SDRAM
- OS:
  - Linux Ubuntu 14.05.1
  - or *Android* 4.4.2 OS

# LAMMPS performance for different CPUs



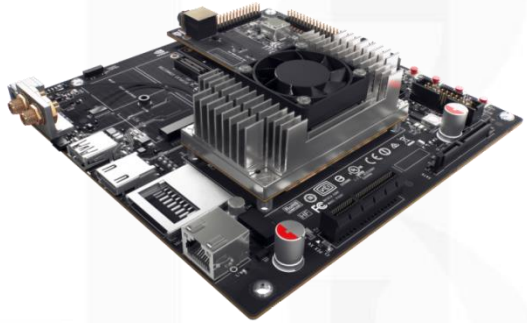



# Nvidia Jetson

## Nvidia Jetson TK1 (2014)

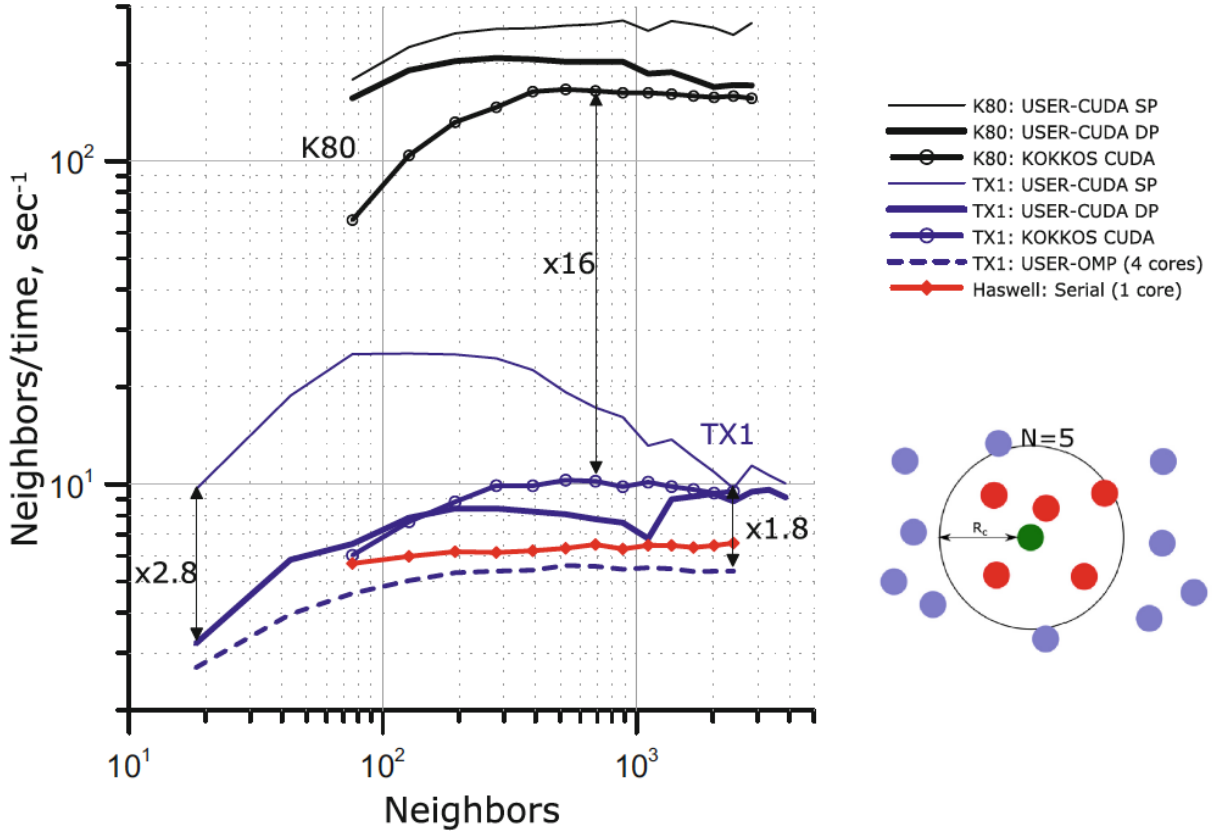

- 32-bit Tegra K1 SoC
- 4 GB memory
- 4 Cortex-A15 cores @ 2.3 GHz
- 1 Kepler SM @ 852 МГц, 128 CUDA cores

## Nvidia Jetson TX1 (2016)



- 64-bit Tegra X1 SoC
- 4 GB 64-bit memory
- 4 Cortex-A57 cores @ 2.1 GHz
- 2 Maxwell SM @ 998 МГц, 256 CUDA cores

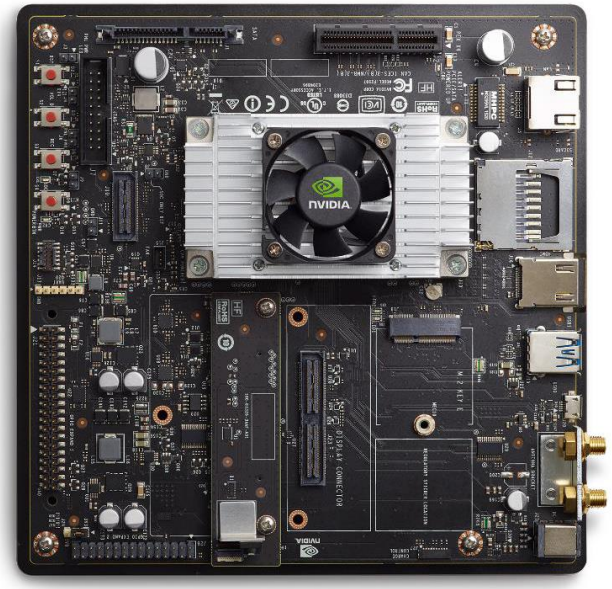
# MD algorithms performance




Nikolskiy, V.P., Stegailov, V.V., Vecher, V.S.: Efficiency of the Tegra K1 and X1 systems-on-chip for classical molecular dynamics. In: 2016 International Conference on High Performance Computing Simulation (HPCS), pp. 682–689 (2016). DOI 10.1109/HPCSim.2016.7568401

# Nvidia Jetson

## Nvidia Jetson TX2 (2017)



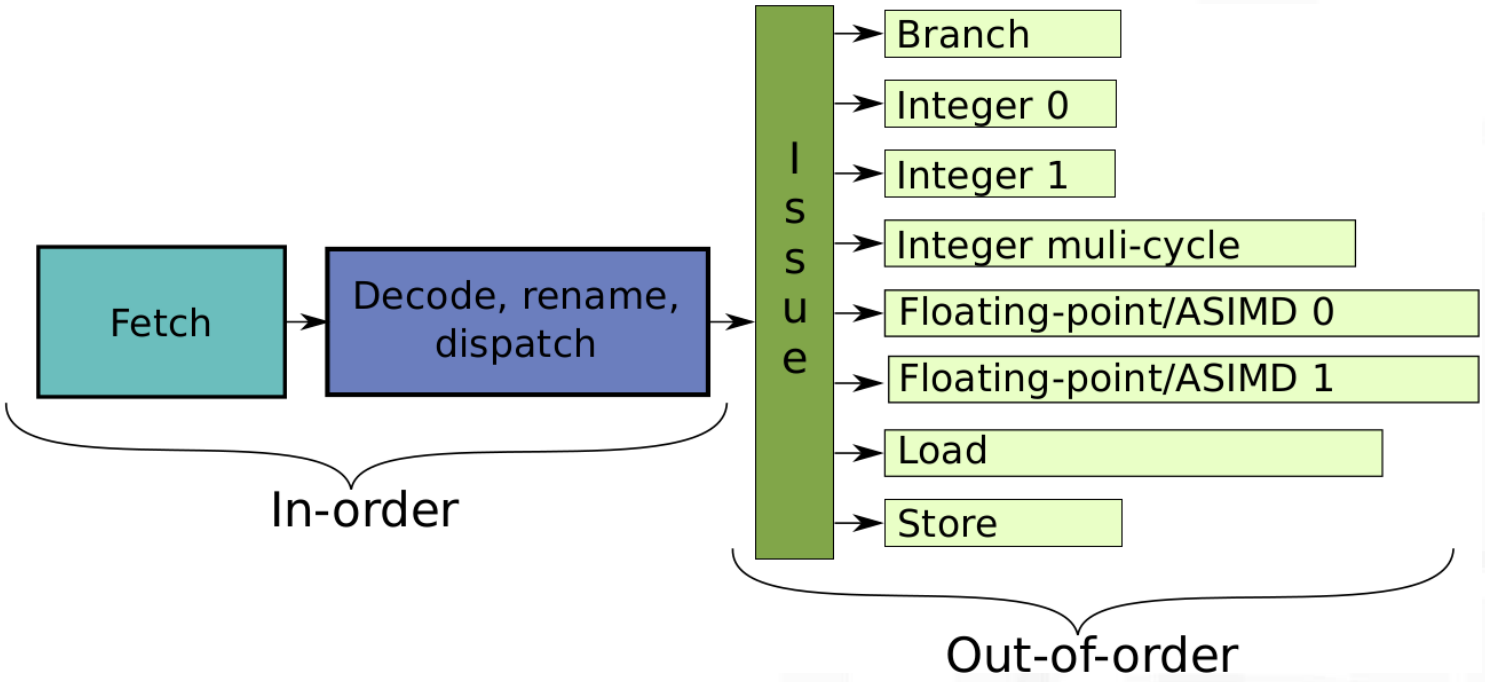

- 2 Denver2 cores + 4 Cortex-A57 cores @ 2.0 GHz
- 8 GB 128-bit memory
- Pascal GPU

# Julich Prototype cluster

- 32 Cortex-A57 cores @ 2.1 GHz (64-bit)
- 1 MB L2 cache, 32 MB L3 cache
- 128 GB memory

- Ubuntu 16.04 LTS
- gcc v5.3.1

# ARMv8 Cortex-A pipeline




$$R(op) = CPU_{freq} \cdot k_{OP}(op) \cdot k_{SIMD}(op) \cdot throughput(op)$$

# Microbenchmark development

## Based on **github.com/Mysticial/Flops**

bench_f64v1_aarch64.h — ~/workspace/Flops/version2/source — Atom

```
444         float64x2_t rA = {2.0f, 2.01f};
445         float64x2_t rB = {2.1f, 2.11f};
446
447         for (size_t i = 0; i < iterations; i++){
448                 r0 = vfmaq_f64(r0, mul0, add0); r1 = vaddq_f64(r1, add0);
449                 r2 = vfmaq_f64(r2, mul0, add0); r3 = vaddq_f64(r3, add0);
450                 r4 = vfmaq_f64(r4, mul0, add0); r5 = vaddq_f64(r5, add0);
451                 r6 = vfmaq_f64(r6, mul0, add0); r7 = vaddq_f64(r7, add0);
452                 r8 = vfmaq_f64(r8, mul0, add0); r9 = vaddq_f64(r9, add0);
453                 rA = vfmaq_f64(rA, mul0, add0); rB = vaddq_f64(rB, add0);
454
455                 r0 = vfmsq_f64(r0, mul1, add0); r1 = vsubq_f64(r1, sub0);
456                 r2 = vfmsq_f64(r2, mul1, add0); r3 = vsubq_f64(r3, sub0);
457                 r4 = vfmsq_f64(r4, mul1, add0); r5 = vsubq_f64(r5, sub0);
458                 r6 = vfmsq_f64(r6, mul1, add0); r7 = vsubq_f64(r7, sub0);
459                 r8 = vfmsq_f64(r8, mul1, add0); r9 = vsubq_f64(r9, sub0);
460                 rA = vfmsq_f64(rA, mul1, add0); rB = vsubq_f64(rB, sub0);
461
462         }
463         flops_reduce_chains12(
464               vaddq_f64,
465               r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, rA, rB
466           );
467         result = vaddvq_f64(r0);
468
469
```

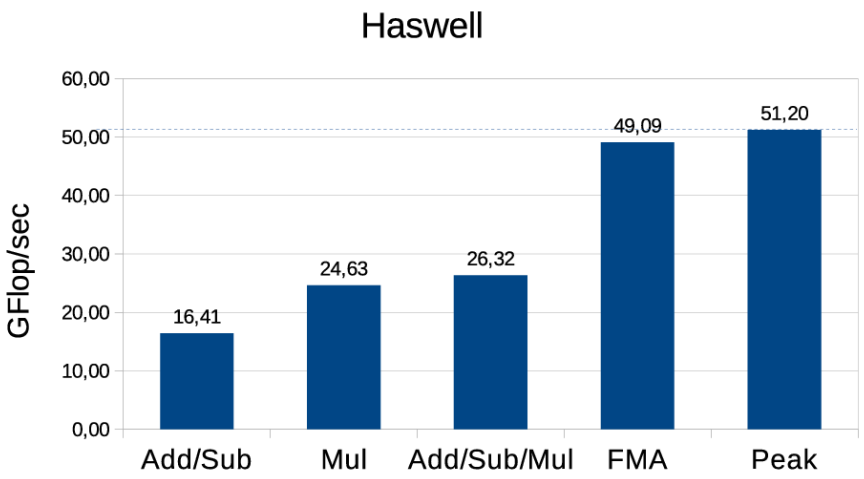
bench_128/bench_f64v1_aarch64.h 467:32 LF UTF-8 C++ master +479

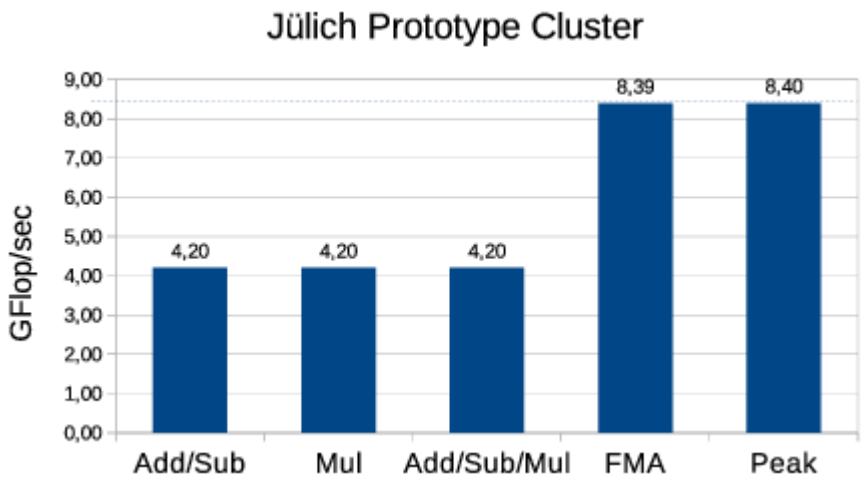
bench_f64v1_aarch64.h — ~/workspace/Flops/version2/source — Atom

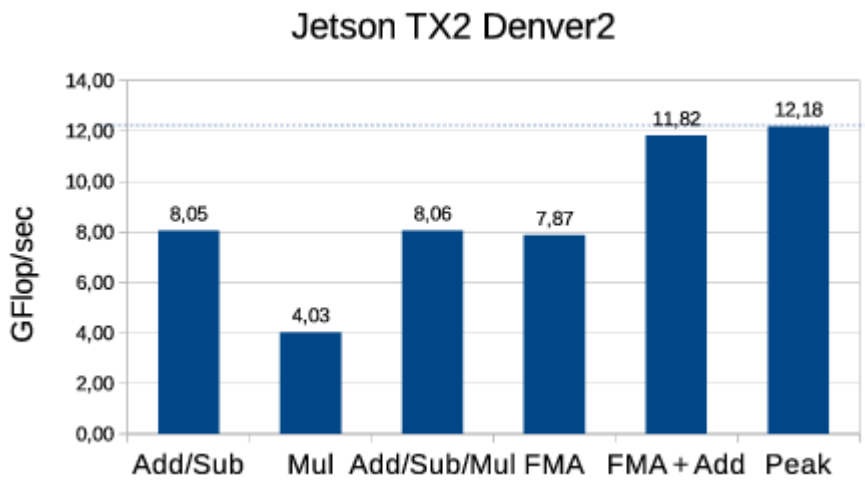
```
348                 vaddq_f64, vmulq_f64,
349                 add0, mul1, mul0,
350                 r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r1A, r1B
351             );
352             flops_muladd_chains12_ops12(
353                 vsubq_f64, vmulq_f64,
354                 sub0, mul0, mul1,
355                 r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, rA, rB
356             );
357             flops_muladd_chains12_ops12(
358                 vsubq_f64, vmulq_f64,
359                 sub0, mul1, mul0,
360                 r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r1A, r1B
361             );
362         }
363         flops_reduce_chains12(
364             vaddq_f64,
365             r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, rA, rB
366         );
367         flops_reduce_chains12(
368             vaddq_f64,
369             r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r1A, r1B
370         );
371         result = vaddvq_f64(r10)+vaddvq_f64(r0);
372
373         //  (2 ops / vector) * (12 ops / macro)
374         return iterations * 2 * 12 * 4;
```

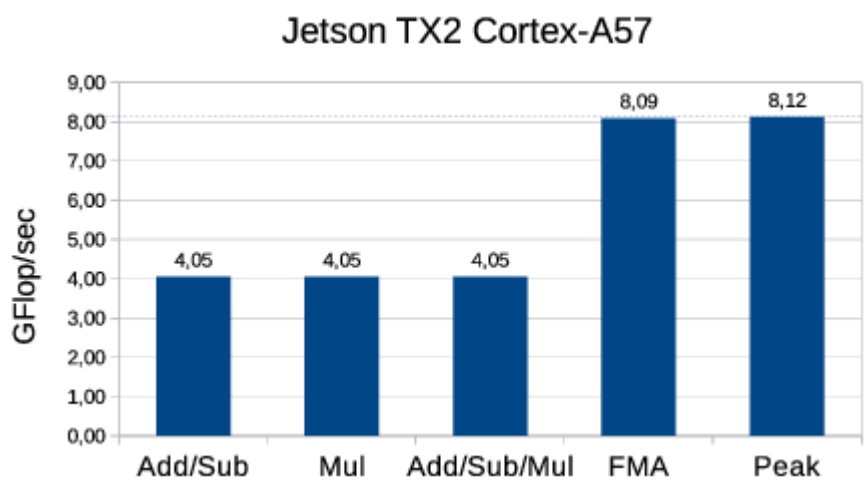
bench_128/bench_f64v1_aarch64.h 467:32 LF UTF-8 C++ master +479

# Microbenchmark results

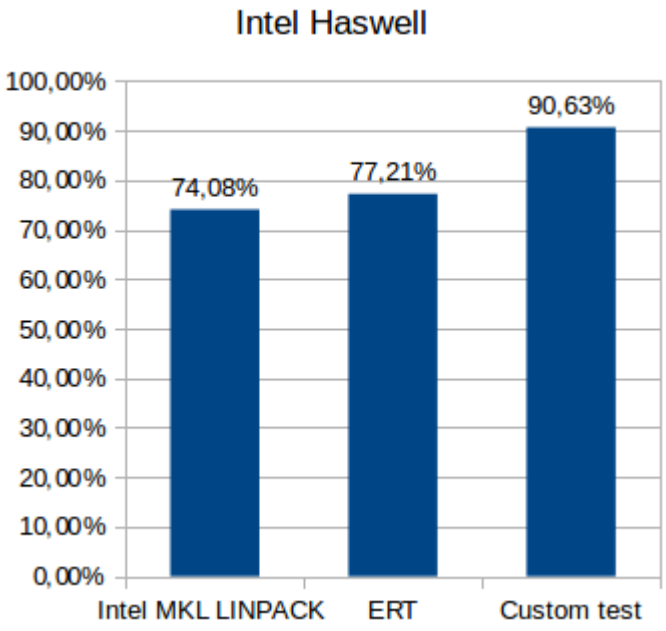


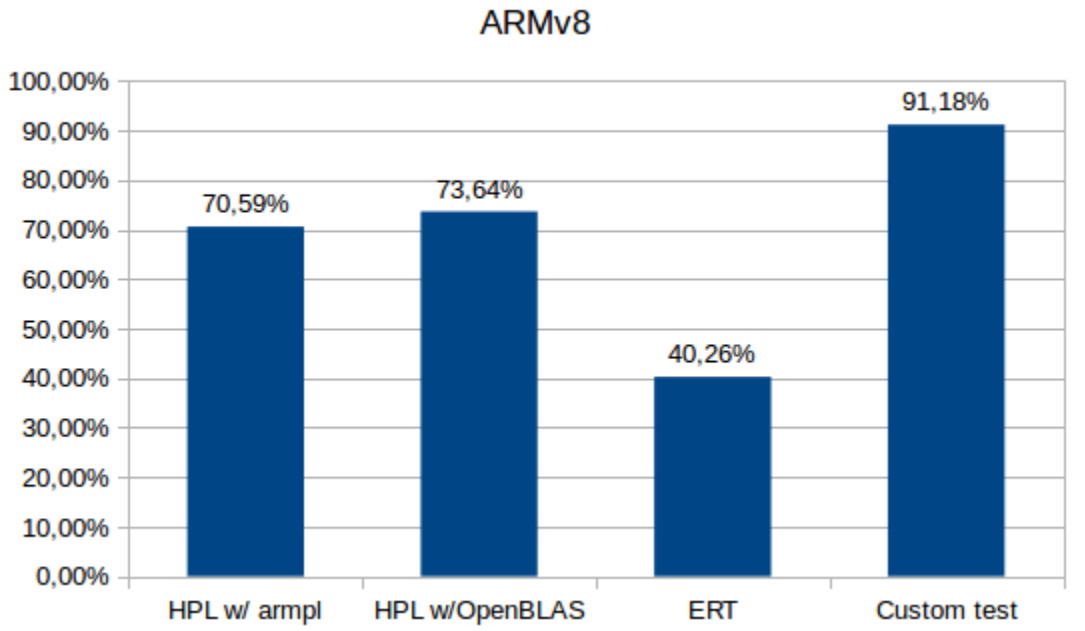



# Benchmark comparison




a = a·c + b

a = b·a + c

a = b·c + a

a = b·c + a

# Conclusions

- The efficiency of ARM Cortex-A was compared with other CPUs (Intel x86_64 and legacy) for the classical MD algorithm example (LJ liquid in LAMMPS).
- The theoretical peak performance of the processors ARM Cortex and Nvidia Denver2 was calculated
- The highly optimized test code was created, that achieves the highest proportion of the calculated peak performance
- We considered the case of incorrect results of popular test Empirical Roofline Toolkit with ARMv8 processors
- We have shown that LINPACK gives correct and predictable results