Impact de l'utilisation de la vectorisation et du multithreading sur la performance et la consommation énergétique

Étude sur les cartes de développement Nvidia Jetson

Sylvain Jubertie, Emmanuel Melin, Naly Raliravaka
sylvain.jubertie@univ-orleans.fr

LIFO, Université d'Orléans

JI 2020
19/11/2020

Architectural optimization and performance/consumption

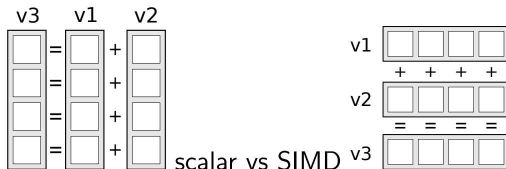
Almost all today's architectures (smartphones ... supercomputers) contain processors with:

- multiple cores
- each core contains a SIMD unit (Single Instruction on Multiple Data)

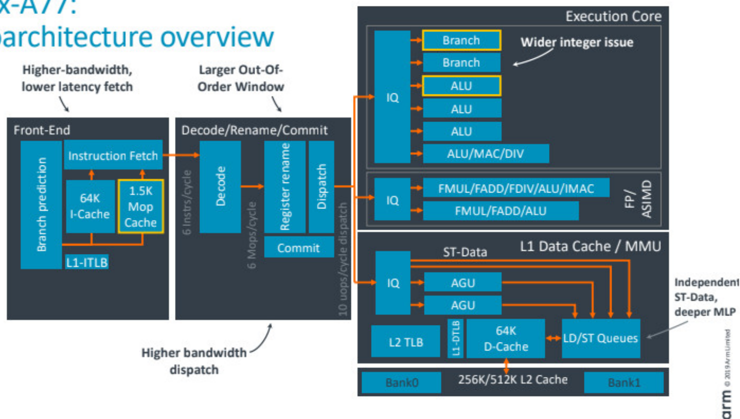
Different scenarios:

- maximize energy efficiency
- maximize performance
- adjust to a performance constraint
- respect a power envelope/thermal constraint

SIMD units



- Apply the same instruction on multiple data (vector of data)
- Several flavours (vector lengths):
    - x86 (Intel/AMD): SSE(128-bit), AVX(256-bit), AVX-512(512-bit)
    - Arm: NEON/ASIMD(128-bit), SVE(up to 2048-bit)
    - Power (IBM): VMX, VSX(128-bit)
    - NEC Aurora Tsubasa
    - GPUs



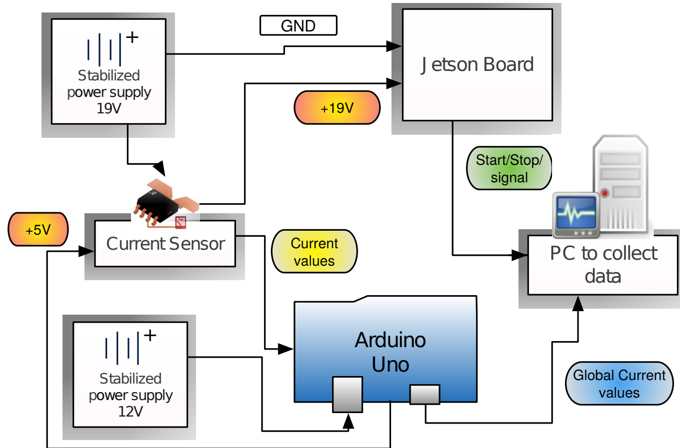
Arm Cortex-A77: 2 ASIMD(128-bit) units, up to 8 SP FMAs/cycle

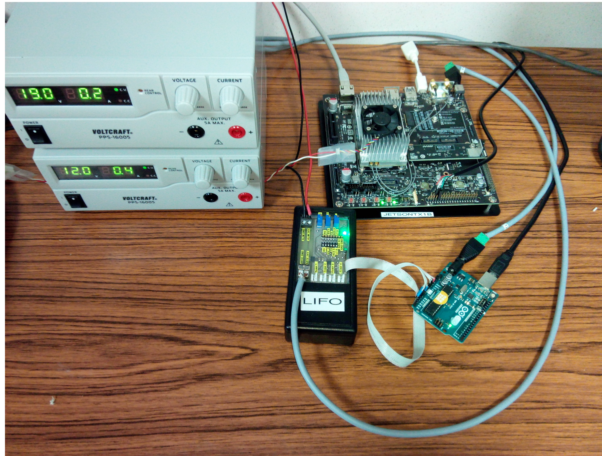
Impact on power/energy consumption

Use DVFS (Dynamic Voltage and Frequency Scaling) to:

- Measure the impact of memory and CPU frequencies on performance/energy consumption.
- Measure the impact of using SIMD units, multiple cores or both on performance and energy: Using SIMD units and multiple cores may require more power but for a shorter time.
- Consider different codes: The impact may depend on the scenario: memory bound vs compute bound
- Compare a single core with SIMD vs 4 cores with scalar units.

Experimental platform




19.0
0.2
VOLTAGE
CURRENT
AUX. OUTPUT
5A MAX.
VOLTCRAFT
PPS-16005
12.0
0.4
VOLTAGE
CURRENT
VOLTCRAFT
LIFO

Jetson boards specifications

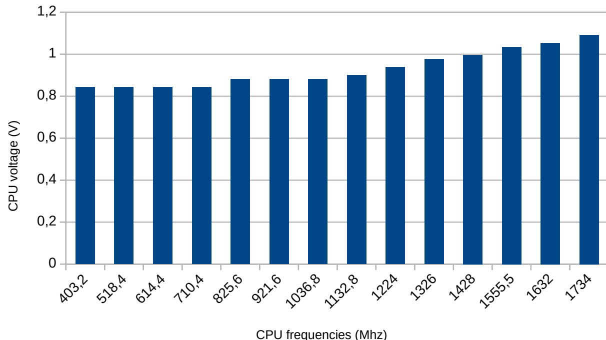
	Jetson TK1	Jetson TX1
processor	4x Cortex-A15 (32-bit)	4x Cortex-A57 (64-bit)
cpu freq.	51 - 2,065	102 - 1,734
cache	32KB L1D/128KB L2	32KB L1D/2MB L2
memory	2GB DDR3L	4GB LPDDR4
mem. freq.	12.75 - 924.0	40.8 - 1,600.0
GPU	192 Kepler cores	256 Maxwell cores
GPU freq.	72.0 - 852.0	76.8 - 998.4

Power consumption formula

$$P_{avg} = fCV^2 + P_{static}$$

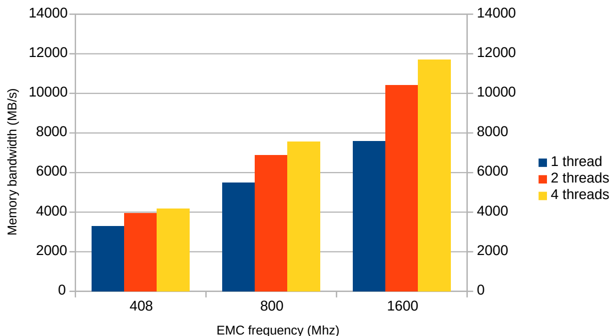
- P_{avg}: average power consumption
- f: CPU frequency
- C: transistor gates capacitance
- V: supply voltage
- P_{static}: static power when CPU is idle

CPU voltage



- Increasing the CPU frequency requires to increase the CPU voltage
- CPU voltage increases non linearly
- From 1.428GHz to 1.734GHz: f x1.21, V x1.1 $\rightarrow$ fV^2 x1.45

Memory bandwidth (STREAM)



- The memory bandwidth depends on the frequency and on the number of cores
- Bandwidth does not increase linearly with the frequency
- Need to use 4 threads to take full advantage of the memory bandwidth

Test codes

4 versions for each code: scalar, NEON, OpenMP, OpenMP+NEON (except OpenBLAS version):

- Image processing:
  - **grayscale**
  - yuv2rgb
- Vector normalization
- Matrix multiplication:
  - **Not optimized**: no vectorization
  - **Not optimized NEON**: NEON intrinsics
  - **OpenBLAS**: NEON assemby instructions, blocking, unrolling

SIMD units and vectorization

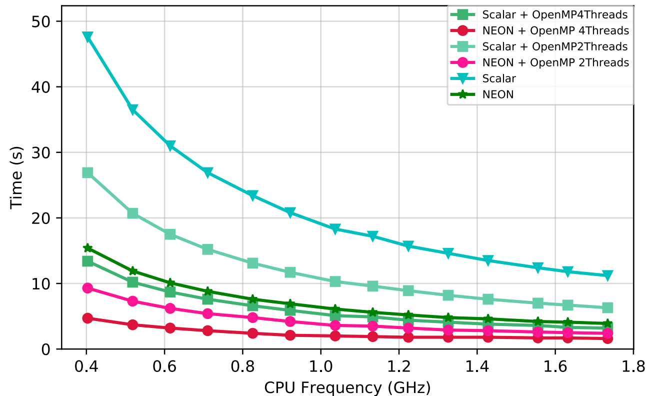
- Codes vectorized using intrinsics
- Automatic vectorization not efficient/portable:
  S. Jubertie, F. Dupros, F. De Martin: *Vectorization of a spectral finite-element numerical kernel, WPMVP 2018*
  S. Jubertie, I. Masliah, J. Falcou: Data layout and SIMD abstraction layers: decoupling interfaces from implementations, HPCS 2018

Grayscale scalar

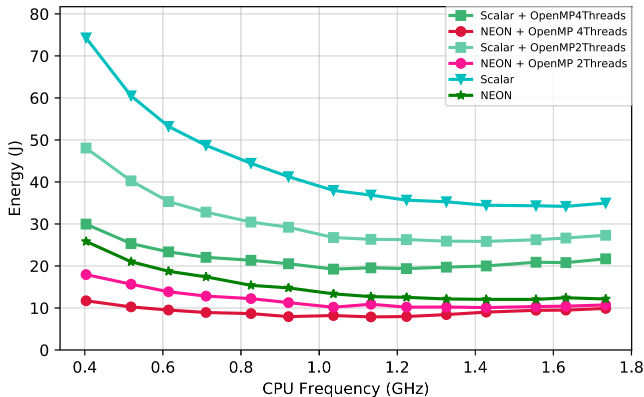
```
#pragma omp parallel for
for( std::size_t i = 0 ; i < out.size() ; ++i )
{
  // out = ( 307 * R + 604 * G + 113 * B ) / 1024
  out[ i ] = ( 307 * in[ 3 * i + 0 ]
             + 604 * in[ 3 * i + 1 ]
             + 113 * in[ 3 * i + 2 ]
             ) >> 10;
}
```

Grayscale NEON, more than 50 LOC

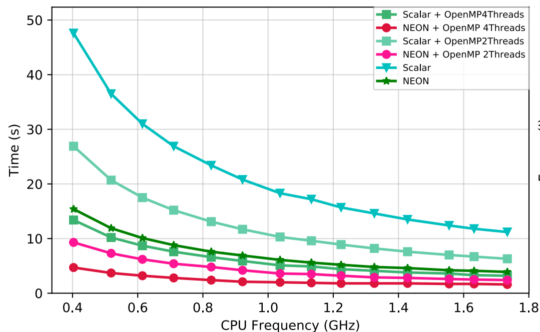
```
#pragma omp parallel for
for( i = 0 ; i < in.size() / 48 * 48 ; i+=48 )
{
  uint8x16x3_t rgb0 = vld3q_u8( p_in );

  // Separate low/high parts of each register
  uint8x8_t r0 = vget_low_u8( rgb0.val[ 0 ] );
  uint8x8_t r1 = vget_high_u8( rgb0.val[ 0 ] );
  uint8x8_t g0 = vget_low_u8( rgb0.val[ 1 ] );
  uint8x8_t g1 = vget_high_u8( rgb0.val[ 1 ] );
  uint8x8_t b0 = vget_low_u8( rgb0.val[ 2 ] );
  uint8x8_t b1 = vget_high_u8( rgb0.val[ 2 ] );
  ...
  uint32x4_t O00 = vmull_u16( R00, v307 );
  uint32x4_t O01 = vmull_u16( R01, v307 );
  uint32x4_t O10 = vmull_u16( R10, v307 );
  uint32x4_t O11 = vmull_u16( R11, v307 );
  ...
}
```

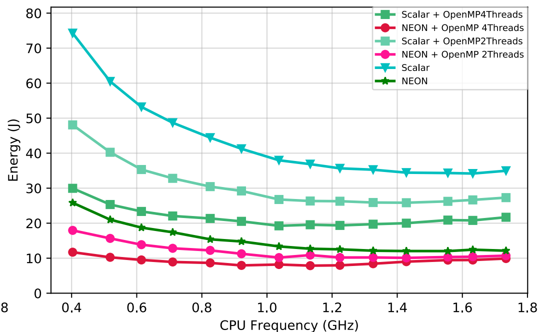


- Memory freq. at 1.6GHz, only modifying CPU freq., # of cores.
- 1 thread NEON ≈ 10-15% slower than 4 threads scalar
- 4 threads scalar & 1 thread NEON ≈ 2.5-3.5x faster than 1 thread scalar
- 4 threads NEON ≈ 6.0-10.0x faster than 1 thread scalar
- 4 threads NEON: frequency x4 → only 2.5 speedup

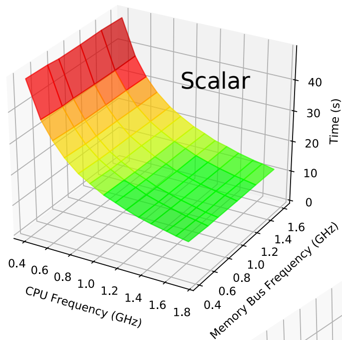


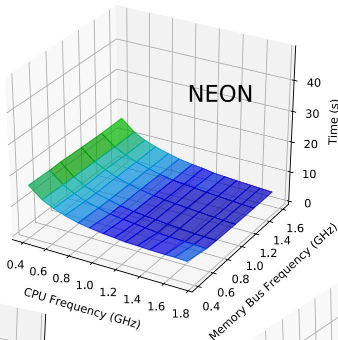
- 1 thread NEON ≈ 3x less energy than 1 thread scalar
- 4 threads scalar ≈ 1.5x less energy than 1 thread scalar
- lowest energy consumption with CPU frequency around 1.1GHz
- energy consumption tends to increase above 1.1GHz

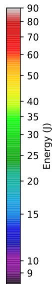


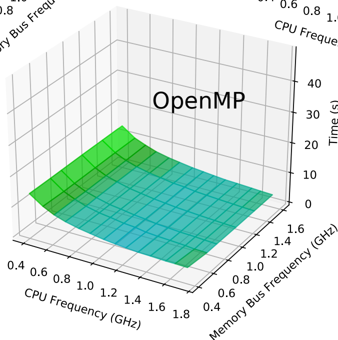


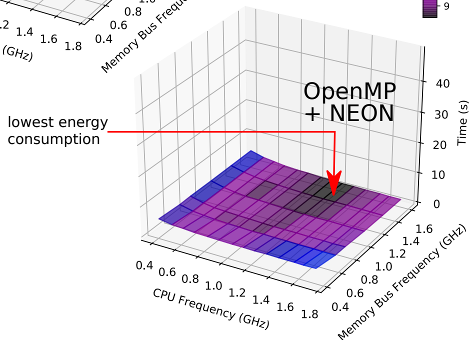
1. 4 threads scalar and 1 thread NEON $\approx$ similar performance for all frequencies...
   ...but energy consumption gap increases with the frequency.
2. 4 threads NEON $\approx$ 2x faster than 4 threads | NEON
3. Improving execution time $\rightarrow$ reducing energy consumption...
   ...up to some frequencies.
4. 2 different behaviors for low and high frequencies.


Scalar
Time (s)
CPU Frequency (GHz)
Memory Bus Frequency (GHz)


NEON
Time (s)
CPU Frequency (GHz)
Memory Bus Frequency (GHz)


Energy (J)


OpenMP
Time (s)
CPU Frequency (GHz)
Memory Bus Frequency (GHz)


OpenMP
+ NEON
lowest energy
consumption
Time (s)
CPU Frequency (GHz)
Memory Bus Frequency (GHz)

Remarks on grayscale

4 threads NEON → Memory-bound

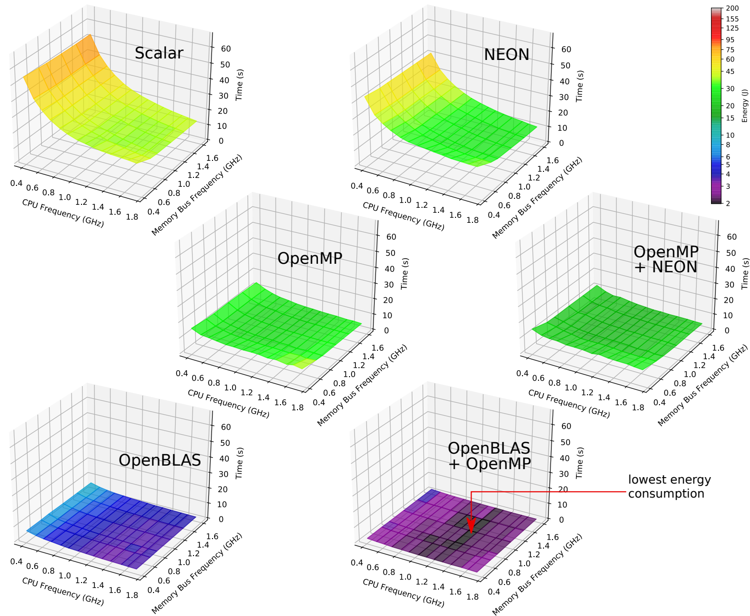
- No performance improvment above 1.2GHz CPU freq.
- Lowest energy consumption at max. memory freq.
- Pushing CPU freq. over 1.2Ghz → wasting energy

Matmul Not optimized

```
#pragma omp parallel for
for( std::size_t j = 0 ; j < dim ; ++j )
{
  for( std::size_t k = 0 ; k < dim ; ++k )
  {
    auto f_a = m_A[ j * dim + k ];
    for( std::size_t i = 0 ; i < dim ; ++i )
    {
      m_C[ j*dim+i ] += f_a * m_B[ k*dim+i ];
    }
  }
}
```

Matmul Not Optimzed NEON

```
#pragma omp parallel for
for( std::size_t j = 0 ; j < dim ; ++j )
{
  auto js = j * dim;

  for( std::size_t k = 0 ; k < dim ; ++k )
  {
    auto ks = k * dim;
    auto r1 = vld1q_dup_f32( &m_A[ js + k ] );

    for( std::size_t i = 0 ; i < dim ; i+=4 )
    {
      auto r3 = vld1q_f32( &m_C[ js + i ] );
      auto r2 = vld1q_f32( &m_B[ ks + i ] );
      r3 = vfmaq_f32( r3, r1, r2 );
      vst1q_f32( &m_C[ js + i ], r3 );
    }
  }
}
```


Scalar
NEON
OpenMP
OpenMP + NEON
OpenBLAS
OpenBLAS + OpenMP
lowest energy consumption
Time (s)
CPU Frequency (GHz)
Memory Bus Frequency (GHz)
Energy (J)

General remarks

Optimization:

1. Best performance: max. CPU and memory freq. (obvious)...
   ...but best performance costs a lot of energy (especially memory-bound codes).
2. Optimized codes (cache): similar speedup NEON vs 4 threads scalar (FP32) but less energy.
   The gap increases with the CPU frequency.
3. Multithreading $\rightarrow$ lower the freq. for min. energy.
4. Efficient vectorization more interesting for energy consumption than multithreading.

General remarks

1 thread NEON vs 4 threads scalar

Energy consumption gap not as huge as expected:
NEON requires only 1 core + L2 enabled, 3 cores disabled
but all cores in the same frequency/voltage domain.
Test: Running 1 thread with 1 core on/3 off or 4 cores on $\rightarrow$ same energy consumption.

Conclusion

- Perf.: NEON $\approx$ 4 threads (32-bit)
- Energy: NEON more efficient than 4 threads (32-bit)
- Vectorizing allows to reduce energy consumption (even when not done properly)
- Best perf/energy ratio around 1.2Ghz (CPU), mem. freq. depends on memory/compute-bound scenario

Always vectorize your code (at least verify if the compiler vectorizes) !!!

... but requires some development effort

Future work

- Tests on other boards (big.LITTLE, x86)
- Tests with integrated and discrete (PCAT Power Capture Analysis Tool) GPUs
- Considering more complex codes: Instrumenting a FEM kernel (extracted from EFISPEC BRGM)
- Integrating the control into the application