On the usage of the Arm C Language Extensions for a High-Order Finite-Element Kernel

Sylvain Jubertie[1] Guillaume Quintin[2] Fabrice Dupros[3]

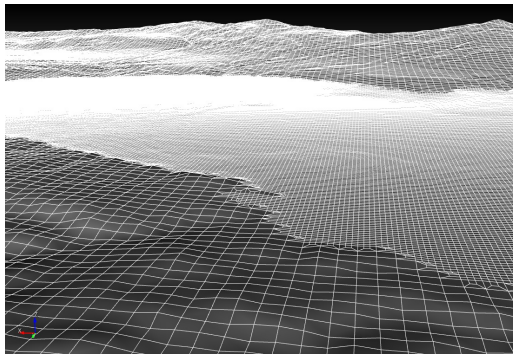
1 UNIVERSITE D'ORLEANS

2 AGENIUM SCALE

3 arm

EAHPC workshop, 14/09/20

High-Order Finite-Element Kernel



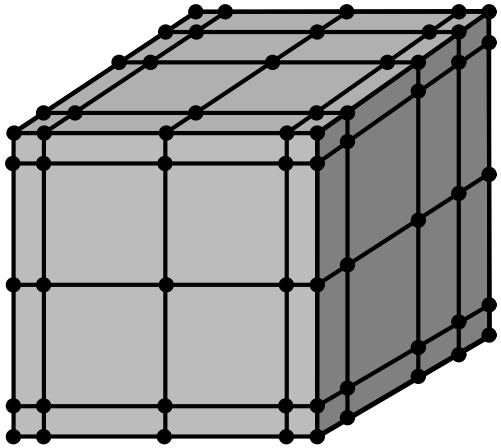
SEM kernel

- ground motion simulation based on the Spectral Finite Element Method
- similar to SPECFEM3D, extracted from EFISPEC3D
- unstructured mesh: elements share points at boundaries with neighbors $\rightarrow$ indirection array to avoid storing points multiple times
- C++ with SIMD intrinsics

EFISPEC

- developed at BRGM: the French geological survey
- Fortran 95, MPI
- good inter node scaling, up to 2048 nodes (65536 cores) on KAUST Shaheen II
- no/poor automatic vectorization (even with pragmas) on all architectures/compilers

Computing acceleration

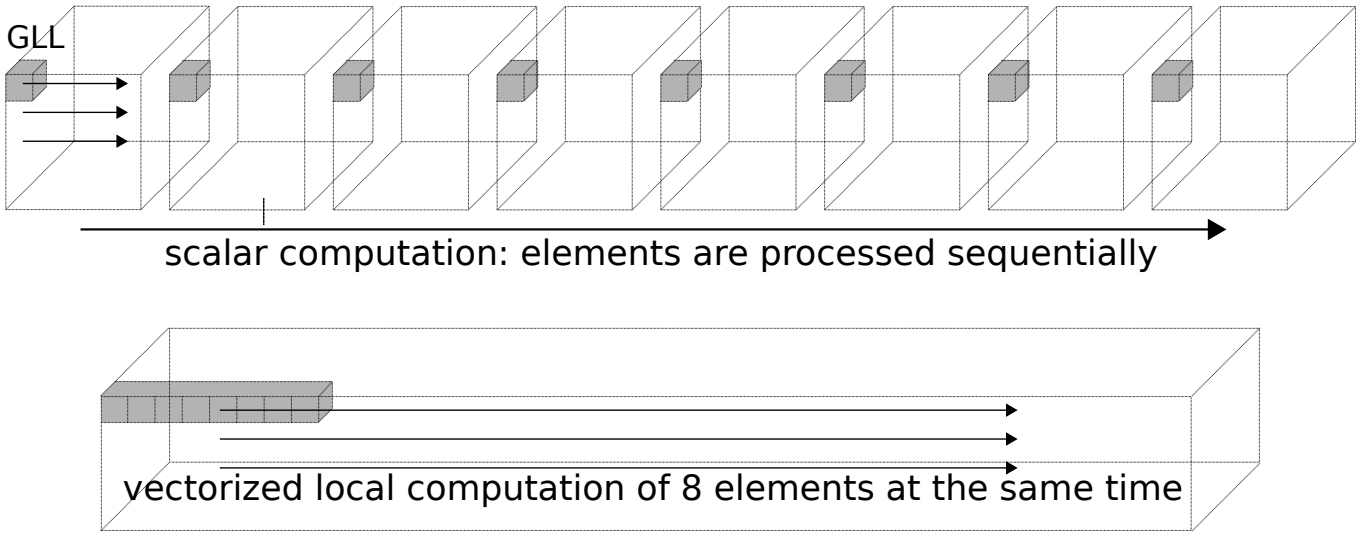


For each element:

1. gathering points to form the element (5x5x5 points at order 4)
2. computing internal forces
3. assembly: updating point accelerations

Vectorization approach

Processing n elements at the same time depending on the vector length (8 for 256-bit VL):



Current SIMD instruction sets supported:

- AVX, AVX2, AVX-512 (WPMVP 2018)
- **NEON**, **SVE**

Explicit vectorization: NEON & SVE differences

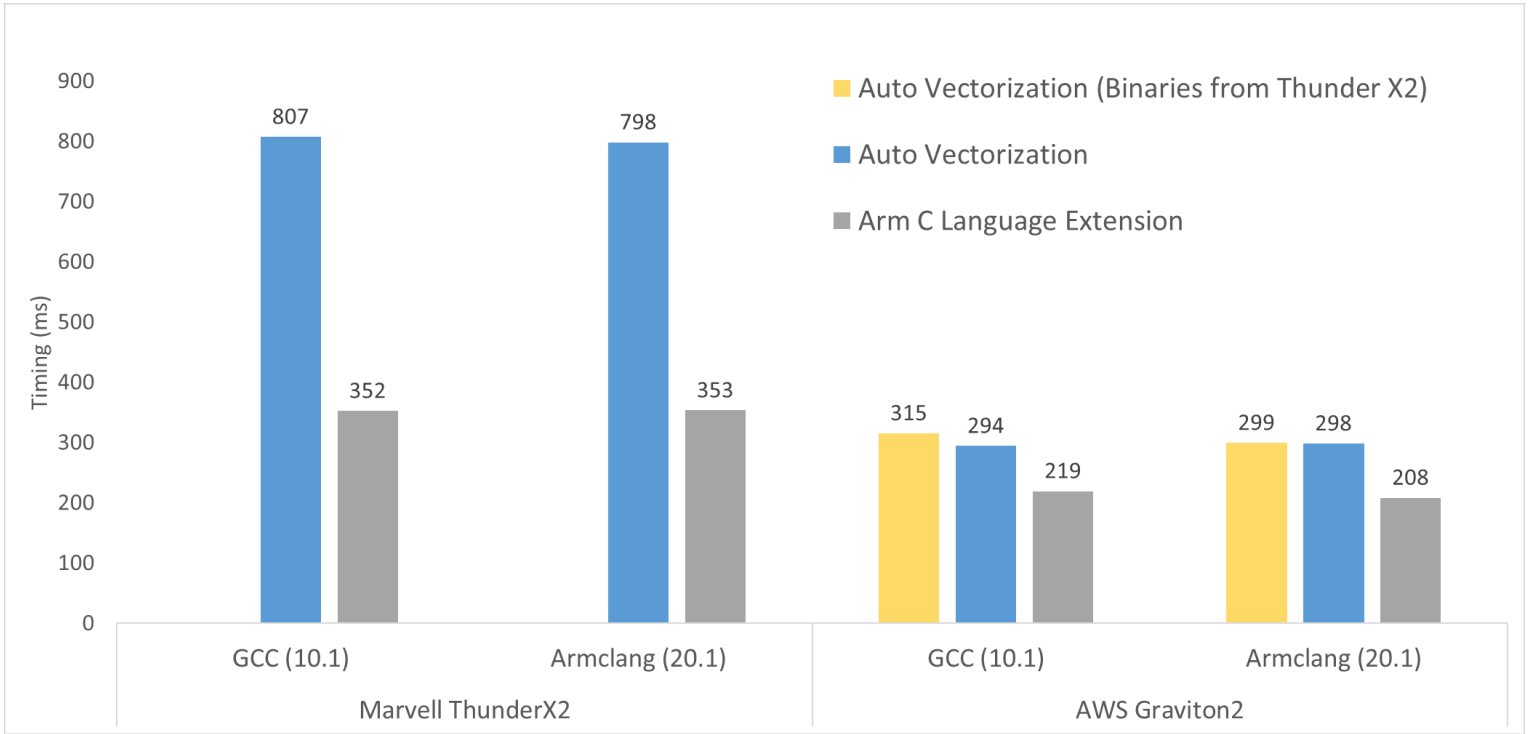
- **NEON (fixed size VL, similar as SSE/AVX):**

```
// Local array declaration.
float32x4_t  rl_displ_gll[5*5*5*3];
// Gathering elements: no gather instr. in NEON.
rl_displ_gll[ id ][ 0 ] = rg_gll_displ[ id0 ];
// Computing: operator overloading, implicit loads.
auto duxdxi = rl_displ_gll[ idx ] * coeff;
```

- **SVE (VL Agnostic):**

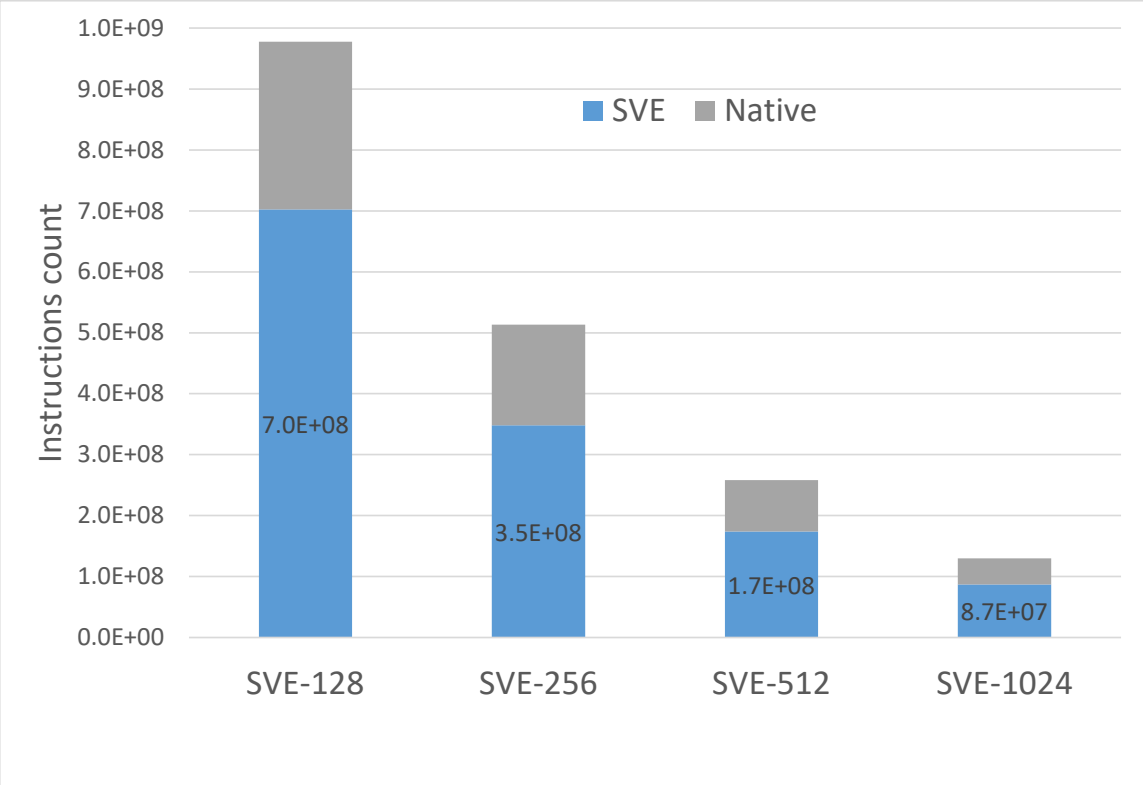
```
// Local array declaration: size not available at compile time.
float  rl_displ_gll[5*5*5*3*svcntw() ];
// Gathering elements.
svst1( mask,
       &rl_displ_gll[ svcntw() * (3 * lid ) ],
        svld1_gather_index( mask, &rg_gll_displ[ 0 ], ids ) );
// Computing: masks, no operator overloading, explicit loads.
auto duxdxi = svmul_z(mask,
                      svld1(mask, &rl_displ_gll[svcntw()*idx])
                      coeff );
```

Results: NEON



- speedups: 2.3x on TX2, 1.4x on G2
- huge architectural improvement of G2: 2.7x scalar, 1.7x NEON
- memory bound (also on x86 architecture)

Results: SVE instructions



- Good vectorization ratio: 70%
- Good scaling: same ratio for all vector lengths

Conclusion & Future work

Conclusion

- Significant speedup (NEON) limited by memory
- Better speedup expected with SVE thanks to HBM ?
- Some difficulties with SVE VLA: code verbosity, compilers

Work in progress & Future work

- Porting the mini-app to NSIMD (submitted to HPCS 2020)
- Tests on the A64FX
- Memory layout optimization
- Integrating the kernel back into EFISPEC

NSIMD

- A single code to generate SSE/AVX/AVX-512/NEON/SVE codes
- Available here : `https://github.com/agenium-scale/nsimd`
- Same performance with NEON intrinsics and NSIMD for the EFISPEC kernel
- Also tested on GROMACS with the same result

```
// Single pack data structure with static or dynamic VL.
using pf32 = nsimd::pack< float >;
auto const len = nsimd::len( f32{});
// Gathering elements.
nsimd::storea<f32>( &rl_displ_gll[ len * (3 * lid) ],
                    nsimd::gather<pf32>(&rg_gll_displ[0],
                                        ids )
                  );
// Computation: explicit loads, operator overloading, default mask.
auto duxdxi = nsimd::loada<pf32>(&rl_displ_gll[len*idx]) * coeff;
```