ENABLING HPC APPLICATIONS FOR SVE

Arm HPC Summit 2019, Austin (TX)

16.09.2019 | Bine Brank, Nam Ho, Stepan Nassyr, Dirk Pleiter, Toni Portero | Jülich Supercomputing Centre

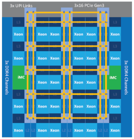
OUTLINE

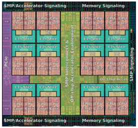
- Introduction
- Porting results and experiences
- Summary and conclusions

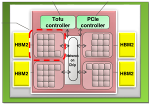
Part I: Introduction

LANDSCAPE OF CPUS FOR HPC

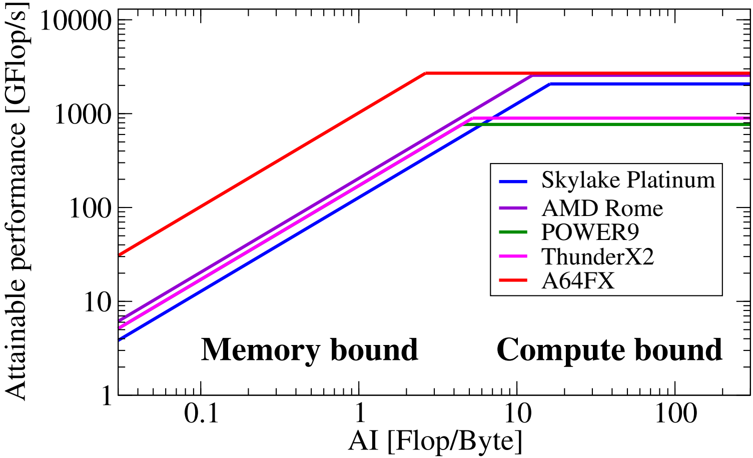
- Solutions based on x86 ISA
  - Extreme computing capabilities
  - Relatively low memory bandwidth
  - Intel processors dominating HPC market ($\gg$ 80 % of Top500)
- Solutions based on POWER ISA
  - Low computing capabilities
  - Relatively good data transport capabilities
  - Low uptake in HPC market
- Solutions based on Arm ISA
  - **Today: Low computing capabilities**
    - Arm’s Scalable Vector Extension (SVE) allows to change this
  - Relatively good data transport capabilities
  - Emerging proposition for HPC market




SMP/Accelerator Signaling
Memory Signaling
CPU Core
L3 Slice
PCIe
SMP Interconnect & Off-chip Accelerator Enablement
SMP Signaling
On-chip Accel
SMP/Accelerator Signaling
Memory Signaling



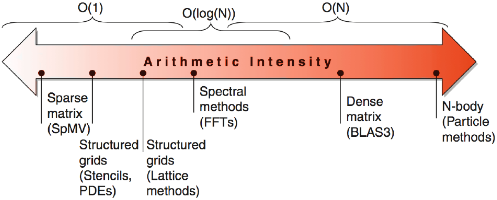
LANDSCAPE OF CPUS FOR HPC: ROOFLINE PERSPECTIVE



LANDSCAPE OF HPC APPLICATIONS

[PRACE, 2018]

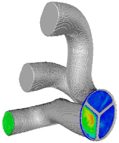
- Expanding the frontiers of fundamental sciences
  - **Sparse matrix**, particle methods
- Climate, weather, and earth sciences
  - **Sparse matrix**, **structured grids**, spectral methods
- Life sciences
  - Particle methods, dense matrix
- Energy
  - **Sparse matrix**, **structured grids**, dense matrix
- Infrastructure and manufacturing
  - **Sparse matrix**, **structured grids**
- Future materials
  - **Sparse matrix**, dense matrix, spectral methods, particle methods



[Hennessy et al., 2012]

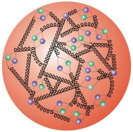
OVERVIEW OF PORTED APPLICATIONS

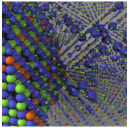
- Computational fluid dynamics using Lattice Boltzmann Methods (LBM)
- Simulation of biologically realistic brain models
- Simulation of particle physics theory for strong interactions: Quantum Chromodynamics (QCD)
- Materials science applications based on the Density Functional Theory (DFT) approach



[Th. Pohl et al., 2004]

[F. Huber, 2018]





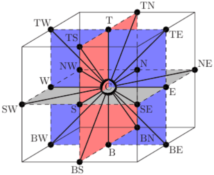
Common property: high scalability

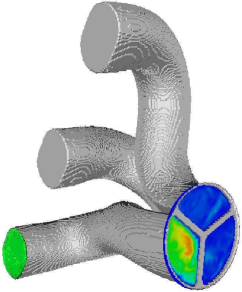
Most applications scaled to 458,752 BG/Q cores

Part II: Computer Fluid Dynamics

LATTICE BOLTZMANN METHODS

- Idea: Simulate simple synthetic dynamics of fictitious particles that provide the correct values of the macroscopic quantities of a flow
- Key benefits: Method is very flexible
  - Different fluid equations
  - Complex geometries and boundary conditions
- Key performance characteristics:
  - Typically low arithmetic intensity: $O(1)$ Flop/Byte for D3Q19
  - All updates are local
  - Relatively regular control flow

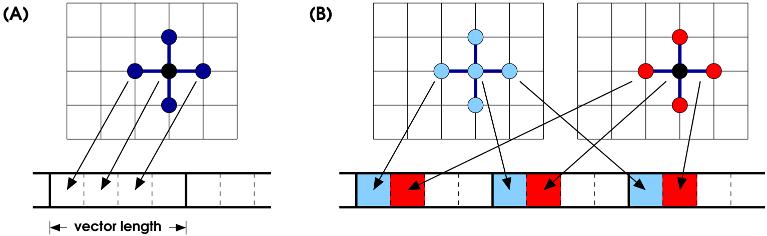




[Th. Pohl et al., 2004]

VECTORISATION OF STENCIL CODES

- Data layout options:



- Option (B) removes need for vector shuffling
- But: Data layout in case of (B) depends on vector length
  - Despite SVE being vector length agnostic (VLA), fixing vector length may be required and beneficial

WALBERLA SIMULATOR

[WaLBerla]

- Flexible LBM simulation application
  - Support of a broad range of LBM models
  - Optimised for different architectures
  - High scalability demonstrated
- WaLBerla's vectorisation strategy
  - Define internal vector type, e.g. based on QPX

```
struct double4_t
{
   double4_t() {}
   double4_t(vector4double init) : v(init) {}

   operator vector4double() const { return v; }

   vector4double v;
};
```

  - Define specialisations of various operations, e.g. add using QPX

```
inline double4_t operator+(const double4_t& a, const double4_t& b)
{
  return vec_add(a, b);
}
```

SVE-PORTING STRATEGY

- Define SVE vectors of length fixed at compile time

```
template <>
struct simd_traits<sve_double> {
    static constexpr unsigned width = SVE_LENGTH;
    using scalar_type = double;
    using vector_type = std::array<double, width>;
    using mask_impl = sve_mask;  // int64x2_t?
};
```

- Provide SVE-based operations, e.g. add

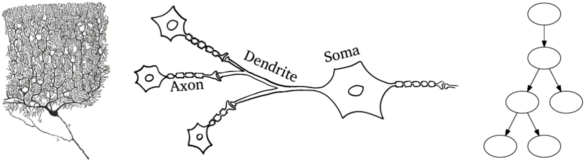
```
struct sve_double : implbase<sve_double> {
    using array = std::array<double, SVE_LENGTH>;

    static array add(const array& a, const array& b) {
        array res;
        svfloat64_t veca = svld1(svptrue_b64(), a.data());
        svfloat64_t vecb = svld1(svptrue_b64(), b.data());
        svfloat64_t vec = svadd_z(svptrue_b64(), veca, vecb);
        svst1(svptrue_b64(), res.data(),  vec);
        return res;
    }
}
```

- Same strategy applicable to other applications based on a similar vectorisation strategy

Part III: Brain Simulations

SIMULATING THE BRAIN: ARBOR

- Simulation of network of spiking neurons
  - Modelling of electrical transport within neurons based on non-linear cable equation
  - Communication between neurons via spikes using, e.g., exponential synapses
- For numerical simulations neurons are represented as branching sequence of compartments
  - Finite volume scheme with an implicit Euler method
  - Need for solving diagonally dominant Hines matrix



[F. Huber, 2018]

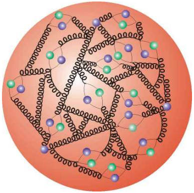
SVE ENABLEMENT

- Auto-vectorisation of operations of arrays of type `std::vector<>`
  - Gather/scatter data elements
  - Arm clang manages to leverage SVE gather/scatter instructions (e.g. `ld1w`)
- Vectorisation of transcendental functions
  - Expand implemented support for AVX, AVS512
  - Implemented functions: exp, expm1, log
  - Close to 100% vector instructions

Part IV: Particle Physics

LATTICE QUANTUM CHROMODYNAMICS

- Quantum Chromodynamics = theory of strong interactions
  - Quarks, the constituents of matter, interact strongly by exchanging gluons
- Particular phenomena:
  - Asymptotic freedom (Nobel Prize 2004)
  - Confinement (Nobel Prize still to winn)
- Lattice QCD = QCD formulated on a lattice
  - Pre-requisite for numerical simulations
- Key performance characteristics:
  - Low arithmetic intensity: O(0.5 . . . 1) Flop/Byte
  - Complex arithmetics
  - Updates are dominantly local
  - Regular control flow
- Ported LQCD application framework: Grid



[P. Boyle et al., 2015]

SVE PORTING STRATEGY

[N. Meyer et al., 2018]

- Leveraging compiler auto-vectorisation

```
struct Sum {
  template <typename T>
  inline vec<T> operator()(const vec<T> &a, const vec<T> &b){
    vec<T> out;

    #pragma clang loop unroll(full) vectorize(enable) \
            interleave(enable) vectorize_width(W<T>::r)
    for (unsigned int i = 0; i < W<T>::r; i++) {
      out.v[i] = a.v[i] + b.v[i];
    }

    return out;
  }
};
```

- Using intrinsics (here: `T` = `double`)

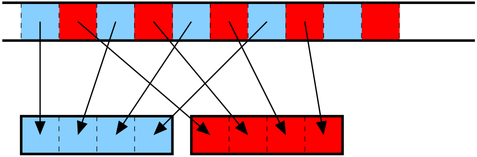
```
struct Sum {
  inline vecd operator()(const vecd &a, const vecd &b) {
    vecd out;
    svbool_t pg1 = svptrue_b64();
    svfloat64_t a_v = svld1(pg1, a.v);
    svfloat64_t b_v = svld1(pg1, b.v);
    svfloat64_t r_v = svadd_x(pg1, a_v, b_v);
    svst1(pg1, out.v, r_v);
    return out;
  }
}
```

COMPLEX ARITHMETICS

- Vectorisation challenge for complex multiplications

$$v \cdot w = (v_r \cdot w_r - v_i \cdot w_i, v_r \cdot w_i + v_i \cdot w_r)$$

- SVE implementation options assuming default AoS data layout
  - Structured loads allow transforming AoS to SoA layout

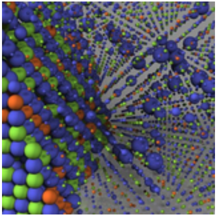


  - Native instructions for complex arithmetics
- Native instructions for complex arithmetics allow to reduce for a significant reduction in the number of instructions

Part V: Materials Sciences

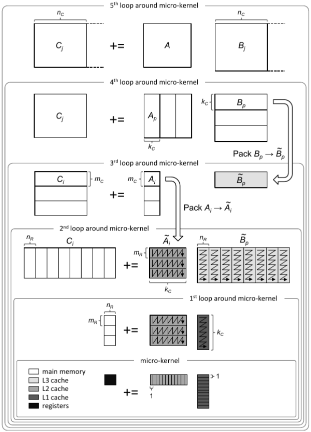
DENSITY FUNCTIONAL METHOD

- DFT = method for investigating electronic structures
  - Approach to solving the intractable many-particle Schrödinger equation
  - Very popular approach (about 15,000 papers per year)
- Many different formulations and applications available, e.g.
  - Quantum Espresso
  - KKRnano
- Common feature of Quantum Espresso and KKRnano: Much time is spent in BLAS3 routines
  - BLAS implementation selected for SVE enablement: BLIS



BLIS DESIGN OF `GEMM`

- BLIS is based on a systematic approach to multi-level blocking
  - Inner-most computations: hand-tuned micro-kernel
  - 5 outer loops
- Known analytic model for selecting implementation parameters based on hardware parameters [T.M. Low et al.]
  - Cache: line size, associativity, size
  - Vector instructions: throughput, **length**
- SVE implementation of `dgemm` and `zgemm` micro-kernels for
  - Fixed length (256 and 512 bit)
  - Variable length

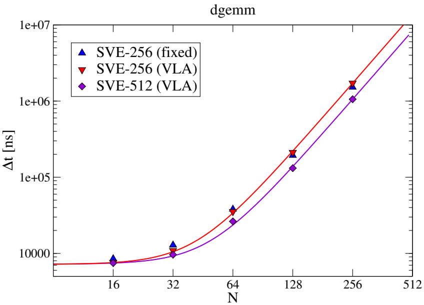


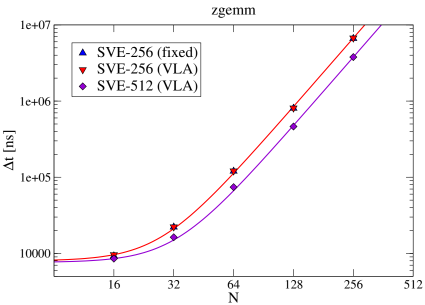
[Van Zee et al., 2017]

`https://gitlab.version.fz-juelich.de/epi-wp1/blis_sve`

BLIS PERFORMANCE EVALUATION

- Gem5 simulation evaluation
  - Single Cortex-A72 like core, 256 kByte L2, 2 SVE pipelines, 2 GHz
- Observiations
  - Similar performance of fixed size and VLA implementations
  - Expected scaling $\Delta t(N) = \lambda + k \cdot N^3 / b_{\text{fp}}$
  - $b_{\text{bf}} \simeq 20$ GFlop/s (SVE-256), $b_{\text{bf}} \simeq 32 \ldots 36$ GFlop/s (SVE-512)





Part VI: Conclusions

CONCLUSIONS

- SVE ISA is a good match for all explored HPC applications
  - Particular interesting:
    - Structured loads
    - Support for complex arithmetics
  - SVE being vector length agnostic is not directly relevant
    - Optimal choice of data layout often requires fixing vector length at compile time
    - Indirect relevance: VLA requires good support of masking, which helps for vectorisation
- Main challenge: Many applications still not prepared for supporting vectorisation
  - Considered applications show that investing in suitable abstraction layer pays off
- Vector-length agnostic versions of dgemm and zgemm implemented
  - Gem5 simulations indicate similar good performance as for micro-kernels optimised for specific vector length

ACKNOWLEDGEMENTS

Funding for the work is received from the European Commission H2020 program under

- Grant Agreement 779877 (Mont-Blanc 2020)
- Special Grant Agreement 826647 (EPI)