The 5th Joint International Conference on Multibody System Dynamics
June 24 – 28, 2018, Lisboa, Portugal

Efficient multibody dynamics simulation based on Intel's Advanced Vector Extensions

Johannes Gerstmayr[1] and Stefan Holzinger[1]

[1] *Department of Mechatronics, University of Innsbruck, {johannes.gerstmayr, stefan.holzinger }@uibk.ac.at*

Nowadays, the real-time simulation of realistic machines is possible. Virtual prototypes of tracked vehicles, autonomous cars or robots are necessary for accurate prediction of durability or system behavior under feedback control. While multibody systems became more complex in the last decades, the *serial* (single-threaded and non-vectorized) computing power did not increase over the last decade, see Fig. 1(middle) for the change of Intel's processors clock rates. For that reason, a new parallelized framework in C++ has been developed for a redesign of the multibody simulation code Hotint.

The current paper focuses on the efficient usage of multithreading and Intel's Advanced Vector Extensions (AVX). As shown in Fig. 1, the number of cores in desktop computers raised to 18 recently [1]. Additionally, the power of single cores has increased by so-called Single-Instruction-Multiple-Data (SIMD) instructions. SIMD has advanced to so-called Advanced Vector Extensions (AVX) which in the current AVX 512-bit version allows to simultaneously compute 16 floating point multiplications and additions in every clock cycle. This leads to high performance processors, compare Fig. 1(right). For the Core i9-7980XE CPU a theoretical 1,5 trillion floating point operations per second (FLOPS) or 1,5 TFLOPS are available, while in serial computation it is only 2,6 GFLOPS – a mere 0,17% of the peak performance.

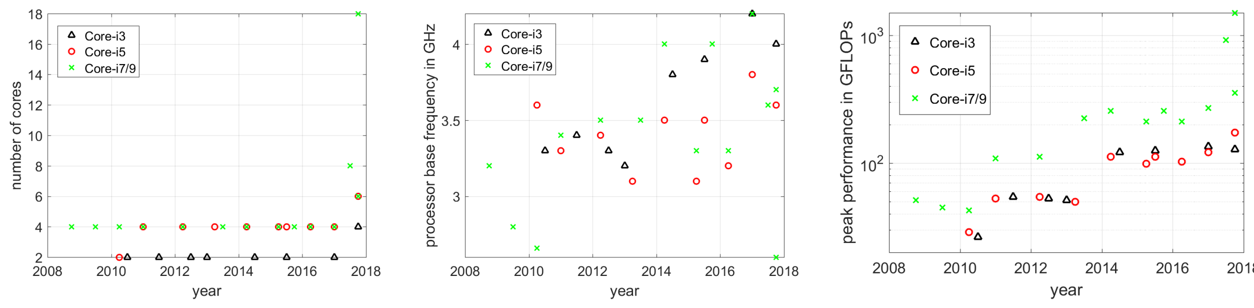

Fig. 1: Evolution of number of cores (left), processor base frequency (middle), and of the processors' theoretical peak performance in GFLOPS (logarithmic, right) for Intel's desktop processors (source: Intel CPU datasheets), not mentioning high-end XEON and XEON-Phi processors.

```
const int n = 1000000; //large scale
float xi[n], fi[n]; //given vectors
float xi1[n]; //result vector
float h; //time integration step size
... //fill vectors xi and fi
for (int j=0; j<n; j++)
{
//perform one time step
  xi1[j]=xi[j]+h*fi[j];
}
```

Listing 1: Pseudo code for a serial calculation of $x_{i+1} = x_i + h \cdot f_i$.

```
const int n = 1000000; //large scale MBS problem
__m256 xi1[n/8], xi[n/8], fi[n/8];
//packed step sizes (8x):
__m256 h8 = _mm256_setr_ps(h,h,...);
... //fill vectors xi and fi
#pragma omp parallel for
for (int j = 0; j < n/8; j++)
{ //vectorized calculation: x_i+1=x_i+h*f_i
  xi1[j]=_mm256_fmadd_ps(h8,fi[j],xi[j]);
}
```

Listing 2: Pseudo code for a parallelized calculation of $x_{i+1} = x_i + h \cdot f_i$ using Intel AVX2 instrinsics and OpenMP.

Current State of Parallelized Multibody Simulation Codes

Parallelization has been applied in multibody system dynamics since three decades [2, 3]. State of the art parallelization of multibody systems is based on multithreading (MT) and message parsing interfaces (MPI) on CPUs, as well as graphics processing units (GPU) [3]. A closer look at the implementation of parallelized codes shows that vectorization and multithreading is implemented for specific parts of the code. However, an extension of such codes to general purpose multibody dynamics simulation leads to large implementation efforts and is error prone. The application of parallelized linear algebra packages is often ineffective, because matrices and vectors at the body level are too small. The straightforward solution would be to rewrite every serial code for a specific parallel hardware. However, this approach is time consuming and not sustainable at all. Furthermore, an open source platform is not likely to be used if it heavily utilizes hardware-dependant commands such as the AVX language, compare Listings 1 and 2. As an example, the currently available version of Hotint [4] includes approximately 70 computable objects which need to be handled in a parallel framework.

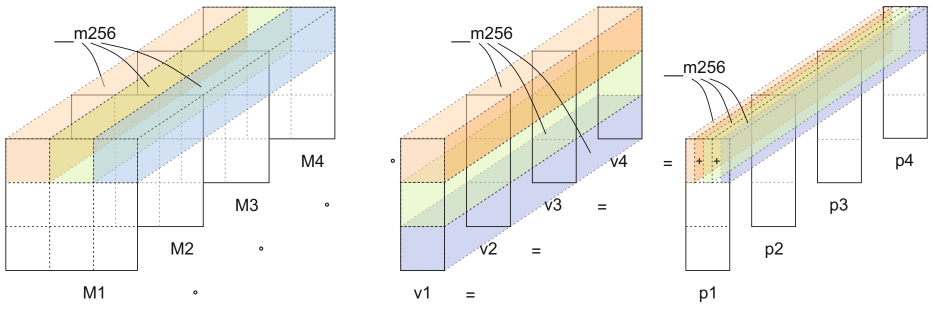

Fig. 2: Sketch of clustered multiplication of matrices M*i* times vectors v*i* resulting in vectors p*i* using AVX intrinsics.

A Template-Based Approach to Parallelization

The solution for the above mentioned problems of parallelization and code development is based on a template-based meta language, which allows to design computable objects, such as rigid bodies, mass points, finite elements, reduced order models and joints mostly hardware-independent and suitable for parallelization. Furthermore, a redundant multibody system formulation is used for mostly independent computation of the computable objects. OpenMP is only effective in the case of a very large number of bodies, therefore an individual task manager is based on so-called `pthreads`, which allows fast execution of tasks. The approach to vectorization is based on arrays of structures, see Fig. 2. As in most computable objects the matrix and vector sizes are no multiples of four or eight, thus the vectorized computation of those objects would be inefficient. Furthermore, a drawback of most SIMD commands is that there is a significant latency from the start of the execution of an instruction to finalization. This means that in every clock cycle a vectorized instruction can be started, however, the results are available about 5 clock cycles later, depending on the instruction and hardware. Therefore, a vectorized computation of a scalar vector product for short vectors cannot be parallelized efficiently. In the proposed approach, the clustering of each computable object leads to so-called arrays of structures, i.e. for AVX2 a scalar consists of 8 floats and a vector of length 3 consists of 3×8 floats. It is important to note that this approach works only, if there are many similar computable objects, such as rigid bodies or generic joints, which is usually the case. In order to reduce code duplication, templates are used for the definition of objects, using a C++-style meta language, to define the objects, e.g. mass matrix or the elastic forces by means of code blocks which are restricted to a set of simple, but parallelizable instructions. As a significant difference to conventional approaches, the meta language is independent of hardware-specific instructions (such as AVX instrinsics), such that the code is portable to hardware in the future.

In the paper we show the detailed design of the template language and some implementation examples for multibody systems. The focus is put on moderately small multibody systems, where memory latency does not play a major role, and a comparison of CPU times to conventional multibody codes is provided.

References

[1] "Intel® Core$^{\text{TM}}$ i9-7980XE Extreme Edition Processor Product Specifications," *www.intel.com*, 2017.

[2] I. Sharf and G. M. D'Eleuterio, "Parallel Simulation Dynamics for Elastic Multibody Chains," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 597–606, 1992.

[3] D. Negrut, R. Serban, H. Mazhar, and T. Heyn, "Parallel Computing in Multibody System Dynamics: Why, When, and How," *Journal of Computational and Nonlinear Dynamics*, vol. 9, no. 4, pp. 41007–41012, 2014.

[4] "HOTINT," *www.hotint.org*.