

An Efficient SIMD Implementation of Pseudo-Verlet Lists for Neighbour Interactions in Particle-based Codes

Extended Abstract

James S. Willis
Institute for Computational
Cosmology (ICC)
Department of Physics, Durham
University,
Durham, UK DH1 3LE
james.s.willis@durham.ac.uk

Matthieu Schaller
Leiden Observatory
Leiden University
Leiden, Netherlands DH1 3LE
schaller@strw.leidenuniv.nl

Pedro Gonnet
School of Engineering and
Computing Sciences
Durham University
Durham, UK
Google Switzerland GmbH
8002 Zürich
Zürich, Switzerland
gonnet@google.com

ACM Reference format:

James S. Willis, Matthieu Schaller, and Pedro Gonnet. 2018. An Efficient SIMD Implementation of Pseudo-Verlet Lists for Neighbour Interactions in Particle-based Codes. In *Proceedings of Super Computing Conference, Dallas, Texas USA, November 2018 (Super Computing 2018)*, 2 pages. DOI: 10.475/123.4

A ARTIFACT DESCRIPTION APPENDIX: AN EFFICIENT SIMD IMPLEMENTATION OF PSEUDO-VERLET LISTS FOR NEIGHBOUR INTERACTIONS IN PARTICLE-BASED CODES

A.1 Abstract

We provide a link to the source code that implements the optimisations described in the poster and outline how the code can be configured and compiled. We then go on to describe how the benchmark can be run in different configurations.

A.2 Description

A.2.1 Check-list (artifact meta information).

- **Algorithm:** Pseudo-Verlet List
- **Program:** SWIFT
- **Compilation:** ICC, GCC
- **Data set:** SWIFT provided benchmark
- **Hardware:** x86 CPUs
- **Output:** Timing results reported by benchmark
- **Publicly available?:** Yes

A.2.2 HOW SOFTWARE CAN BE OBTAINED: SWIFT is an open source project and can be downloaded from GitLab at <https://gitlab.cosma.dur.ac.uk/swift/swiftsim>. The Git revision used to produce the benchmark results was ab123564.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

Super Computing 2018, Dallas, Texas USA

© 2018 Copyright held by the owner/author(s). 123-4567-24-567/08/06...\$15.00

DOI: 10.475/123.4

A.2.3 Hardware dependencies: The code should run on many x86 platforms. It has been tested on the following Intel Xeon processors: Sandybridge, Broadwell, Skylake and Intel Xeon Phi Knights Landing processor.

A.2.4 Software dependencies: To configure and build the code the following tools are needed: libtool, HDF5, ICC/GCC compiler. A more detailed list can be found at <http://icc.dur.ac.uk/swift/onboarding.pdf>.

A.2.5 Datasets: We used a standardized benchmark of the core compute kernels of SWIFT. The benchmark, named *test27cells*, computes the interactions between one cell containing 216 randomly placed particles and its 26 neighbours which are similarly populated. This allows us to probe the speed-up from vectorisation in all the relevant geometrical configurations, and is representative of the workload in actual production runs using SWIFT. *test27cells* is found in the *tests* directory relative to the root directory of SWIFT.

A.3 Installation

Once the code is downloaded from the GitLab repository, it can be configured and built for the host architecture using:

```
$ ./autogen.sh
$ ./configure
$ make -j 6
```

The configure script should auto-detect which platform the code needs to be compiled for. The benchmark is then built with:

```
$ cd tests
$ make test27cells
```

This will compile the code with vectorisation enabled by default, to obtain timing results for a serial run the code can be configured via:

```
$ ./autogen.sh
$ ./configure --disable-vec
$ make -j 6
$ cd tests
$ make test27cells
```

A.4 Experiment workflow

The benchmark can be run using these commands:

```
$ cd tests
$ ./test27cells -n 6 -r 10000 -h 1.2348 -d 0.1 -v 1
```

Each command line argument, `-n` `-r` `-h` `-d` `-v`, changes the number of particles in each dimension of the box; the number of runs; sets the smoothing length of the particles; how much to perturb the particle positions and which velocity field to set.

A.5 Evaluation and expected result

After running the benchmark the timing results should be output to the terminal. The line containing "SWIFT calculation took" displays the total time taken, measured in clock ticks, averaged over the number of runs, which is specified by `-r`. The time can be converted to seconds by dividing it by the clock frequency of the processor that the benchmark was ran on. The benchmark can be run with vectorisation disabled to obtain a relative speed up due to the optimisations described in the poster.