Portable Support for Explicit Vectorisation in C

Artjoms Šinkarovs[1] and Sven-Bodo Scholz[2]

[1] University of Hertfordshire, Hatfield, Hertfordshire, AL10 9AB, United Kingdom
[2] Heriot-Watt University, Riccarton, Edinburgh, EH14 4AS, United Kingdom

Abstract. In pursuit of requirements of modern software, high-performance computing often offers a narrowly-tailored solutions that reduce portability of software. As one of such examples in this paper, we consider the situation with SIMD accelerators, whose importance seriously increased in the last decade. Firstly appeared in the early 90es, being oriented exclusively on graphics acceleration, nowadays SIMD CPU extensions are used in a variety of fields. The lack of standard and incompatibility of instruction sets through the different types of CPUs substantially increase the complexity of developing portable applications within the extensions. In this paper we present an abstraction layer implemented as a set of C language extensions within the GNU GCC compiler which provides an interface for SIMD vectors and operations independently from the architecture. First of all, these abstractions allow to exploit SIMD extensions of a CPU explicitly, which is useful when auto-vectoriser fails. Secondly, the abstractions are general enough to be mapped to any hardware supporting SIMD paradigms; hence the new abstractions could be considered as a step forward to a new C language standard.

1 Introduction

Starting from the early 90es most of the computational platforms have incorporated Single Instruction Multiple Data (SIMD) extensions into their processors. The SIMD mechanisms allow architectures to explore data-parallelism by executing one and the same operation on multiple data objects packed into a vector register that holds several scalar values. Efficient use of SIMD instructions proves increasingly important for achieving excellent performance in a variety of applications. Most computationally intensive applications spend most of their time inside a few hot-spots, typically the innermost loops. These very often apply arithmetic operations on large sets of indexed data, a situation very amenable to SIMD instructions. If applicable, such a use of SIMD instructions immediately increases performance of the loop several times depending on the size of the scalar data type. On current hardware one can expect speed-ups of factor two or four when operating with floating point numbers. Besides the immediate gains, the use of SIMD operations is typically orthogonal to any gains obtained from multi-threaded execution, i.e., in case of a successful SIMD optimisation the overall speed-up can be obtained by multiplying the speed-up due to the use of SIMD instructions with the number of cores available. Finally, we can observe that the length of vector registers provided by new architectures doubles every