Abstract. Last year, we discussed the issues surrounding the development of languages and compilers for a general, portable, high-level SIMD Within A Register (SWAR) execution model. In a first effort to provide such a language and a framework for further research on this form of parallel processing, we proposed the vector-based language SWARC, and an experimental module compiler for this language, called Scc, which targeted IA32+MMX-based architectures.

Since that time, we have worked to expand the types of targets that Scc supports and to include optimizations based on both vector processing and enhanced hardware support for SWAR. This paper provides a more formal description of the SWARC language, describes the organization of the current version of the Scc compiler, and discusses the implementation of optimizations within this framework.

1 Introduction

In the SWAR processing model, a wide data path within a processor is treated as multiple, thinner, SIMD-parallel data paths. Each register word is effectively partitioned into fields that can be operated on in parallel. In this paper, we refer to one word of fields as a fragment, and a multi-fragment object as a vector.

Current compiler support for the SWAR model falls into four major categories: compiler “intrinsics”, classes or types representing a fragment, semi-automatic vectorization of loops, and languages which provide first-class vector objects. The first two categories are typically limited to giving the programmer low-level access to the SWAR instructions of the target. The third provides an abstract model which hides the use of these instructions by employing well-known techniques to parallelize loops in existing C code. The fourth category provides an abstract model in which the semantics of the language indicate which operations can be automatically parallelized. We will briefly discuss these support methods as they apply to the MMX and 3DNow! families of SWAR extensions.

Compiler intrinsics are preprocessor macros which provide a function-call-like interface to the target’s SWAR instructions. Generally, these are easy to implement and simply hide inline assembly code which is used to execute a single SWAR instruction. Intel’s C/C++ compiler [67], MetroWerks CodeWarrior [10], Sybase’s Watcom C/C++ [14], and the lcc-win32 project...
(www.remcomp.com/lcc-win32) all provide some level of support for using MMX and/or 3DNow! instructions via intrinsics.

New class or type definitions which represent a fragment provide a first-class feel to these objects and the operations on them. To do this, class definitions provide overloaded operators, while non-class definitions typically require a modification to the language. Often, these models are built on top of a set of intrinsics, and support only the partitionings and operations directly associated with the targeted hardware. Several compilers employ this technique. The Intel compiler includes class libraries for MMX. Free Pascal [2] includes predefined array types for MMX and 3DNow!, and extends Pascal to allow some first-class operations on these types. Oxford Micro Devices’ compiler for its A236 Parallel Video DSP chip [13] provides predefined struct types for MMX, but requires modifications to the C language.

Under strict conditions, and with hints from the programmer, some compilers are able to vectorize simple data-parallel loops. This support is in the early stages and is limited in the data types and operations that can occur in the body of the loop. We expect the development of this style of parallelism to follow that of Fortran loop manipulation. The Intel and MetroWerks compilers provide limited vectorization for MMX. The MetroWerks’ compiler also supports 3DNow!.

In languages which provide first-class vector objects, both the precision of the data and the number of elements in the vector may be virtualized beyond the limits of the target’s SWAR extensions. A full set of operations on these vectors is provided, rather than a subset based on the SWAR instructions available on the target. To the best of our knowledge, it is still true that only the SWARC language provides this type of model.

We have chosen this last approach because we believe it offers the best opportunity for performance gains over a large range of applications and target architectures. Using the SWARC module language and related compilers, users can write portable SIMD functions that will be compiled into efficient SWAR-based modules and interface code that allows these modules to be used within ordinary C programs.

The level of hardware support for SWAR is inconsistent across architecture families. In our previous work [4], we discussed the reasons for this and the basic coding techniques needed to support a high-level language, such as SWARC, on any type of SWAR target including 32-bit integer instruction sets that have no explicit support for SWAR.

This paper will focus on our work since LCPC 98. Section 2 contains a brief explanation of the SWARC language. Section 3 provides an overview of the organization of Scc, the first implementation of an experimental SWARC compiler. In section 4 we consider the implementation of compiler optimizations which we introduced last year within the Scc framework. In section 5 we discuss some preliminary performance numbers. A brief summary, and pointers to the support code that we have developed, are given in section 6.