Developing Fast DSP Libraries for Advanced Processors.

David Murray, N.A. Software Ltd (davem@nasoftware.co.uk)
Mike Delves, N.A. Software Ltd (delves@nasoftware.co.uk)

Introduction

The high performance embedded DSP processor market has moved steadily over the past decade from one dominated by specialised fixed point processors towards those with recognisable similarities to the general-purpose processor market: DSP processors have provided floating point, substantial caches, and capable C/C++ compilers, as well as SIMD (Single Instruction Multiple Data) features providing fast vector operations. The inclusion of SIMD features to general purpose processors has led to them replacing specialised DSPs in many high performance embedded applications (such as Radar, SAR, SIGINT and image processing).

More recently, multicore processors have become common and will likely dominate in the next five years. The easiest way for existing DSP code to make use of the increased power available, is via standard DSP libraries; but it remains a major task to multithread these.

We describe here a tool which automates the threading of a wide class of DSP routines, and give examples of its effectiveness.

Liberator

Liberator is our system for automatically generating numerical libraries. It produces highly efficient libraries for a range of processors together with a variety of API's (Application Programmers Interface); it also generates test and timing programs, together with standard documentation. Liberator thus allows DSP libraries to be produced and updated cost effectively without loss of efficiency: indeed, -produced libraries typically match or out-perform hand coded libraries.

How It Works

Liberator targets processors with a SIMD capability such as the PowerPC G4 with Altivec, or Intel with SSE2/SSE4. It uses a four-level hierarchy of metadata providing information on how to generate the code for a function:

Level-0: API function. This level describes how routines will be called. In development versions of the libraries, error checking on arguments is done here.

Level-1: provides a simple C-based description of what each routine does.

Level-2: contains the bulk of the operations carried out by Liberator. It generates code which packages the data into single vectors and performs a range of optimisations. It:

blocks the data;

unrolls block loops;

prefetches blocks when this is helpful;

I*mplements appropriate cache management strategies* (these are target dependent);

Implements strategies which depend upon the data. For example, aligned and unaligned data are treated separately, as are vectors with stride 1 (contiguous data) or 2 (typically interleaved complex data);

Handles "edge effects": vector or matrix sizes which are not a multiple of the SIMD length.

Level-3: implements basic vector operations on SIMD-sized vectors. This is the level at which hand coded or manufacturer-provided code building blocks are accessed, including assembler code blocks for FFT and other routines.

Adding Multithreading

Multithreading fits naturally into the Liberator framework at Level 2. To include it we need to:

- Design multithtreading strategies for classes of algorithms
- Teach Liberator about these
- Tune Liberator's knowledge base for a specific target system so that the library routines utilise the optimum number of threads for a specific call.

The tuning phase is itself automated, alongside the corresponding tuning for loop unrolling depth etc; Liberator will emit a library tuned for a specific board, if requested.

Currently vector, matrix and FFT routines (1D and 2D) utilise separate threading strategies.

How Well does it Work?

Figure 1 shows Mflop rates obtained for a multiple 1D complex FFT routine, running on an Intel dual-core SL9400 system at 1.66GHz. Results are given for comparison on a 1GHz 8641D, and a 2.16GHz Intel T7400 board. The threading is obviously very effective. Similar results are obtained in other routines, and we now have a production threaded VSIPL library (other APIs arealso supported by Liberator) for multicore Intel SSE processors.

F*urther results on systems utilising up to 24 cores will be shown in the full paper*.

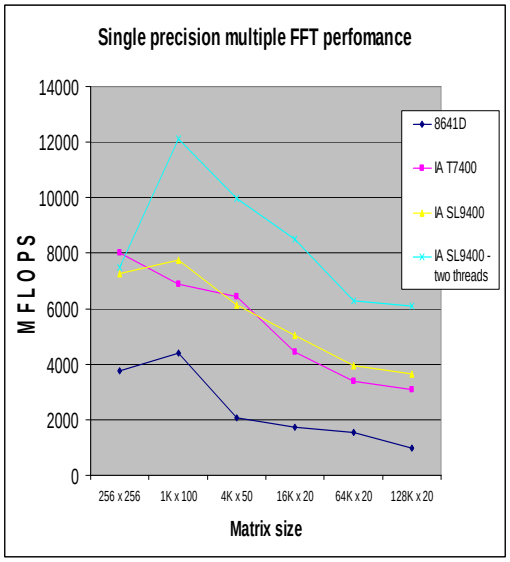

Figure 1 Complex to complex multiple 1D in-place FFTs: MFLOPS = 5 N Log_2(N) / (time for one row FFT in microseconds)

Data: M rows of length N; FFT the rows.

Other Conversion Tools

The performance figures in Figure 1 also illustrate why there is high interest in moving from the PowerPC to the Intel range of processors for compute-intensive DSP applications. However, in moving to a new processor it's not only the hardware and the libraries that need porting: applications that make use of low level processor features have to be ported too. We have two (Intel funded) porting tools under development:

1. **altivec2sse.h:** for PPC/Altivec programmers who address the altivec via the include file *altivec.h*, a*ltivec2dde.h* targets Intel SSE instructions rather than altivec instructions.
2. *PPC/Altivec assembler compiler:* will provide more general assembler conversion facilities.

Item 1 is available no (it's free). Item 2 is at a relatively early development stage.

The full paper will give a brief progress report on Item 2.

Acknowledgements

The performance figures given here were provided courtesy of GE Fanuc; we are grateful to David Tetley for running these benchmarks.