A PROPOSAL FOR IMPLEMENTING A SIMD TYPE EXECUTION UNIT

Jose M. Alvarez and Mark W. Brehob
Department of Computer Science and Engineering, Michigan State University
East Lansing, MI 48824-1226
Alvarez7@cse.msu.edu

Abstract

Recently there has been an increasing interest in the implementation of SIMD in mainstream processors. These implementations mainly consist of multimedia data types extensions, such as Intel MMX, Digital MAX, and SUN VIS. In these implementations the processing is done by packing data into one register and applying the operations to the pack. These operations are mainly targeted at multi-media applications making it difficult for those trying to use the operations for other purposes. And even within the realm of multi-media, the *algorithms* tend not to be directly portable between architectures.

In order to attack these flaws, we propose to add a more general SIMD-type execution unit to a general-purpose processor. Our proposed SIMD execution unit will be made up of 64 processing elements (PE's) arranged in an 8x8 torus. Each PE would have eight 32-bit registers that can store single precision floating point and integer registers. The PEs can perform basic arithmetic and logic operations, and have the ability to move data between adjacent processing elements.

Compilers currently have a difficult time taking advantage of current MMX-like instructions. As such we believe that programmers wishing to take advantage of the SIMD execution unit will have to write their own assembly code or take advantage of pre-written libraries. Because the SIMD unit will be a super-set of MMX-like units they will be able to take advantage of compiler improvements targeted at multimedia instructions. In addition the more general purpose SIMD programmers will often use well known algorithms such as matrix multiplication, scatter-gather, and Fast Fourier Transformations. Libraries can be easily provided for those generic activities.

Upon examining a number of different algorithms for the SIMD execution unit we have discovered latency issues due to data movement problems. The problem is that there are 3 "stages" to most of these algorithms: loading the execution unit with the data, executing the algorithm, and storing the data. Fortunately these algorithms lend themselves to pipelining by overlapping those three "stages." This pipelining can hide most of the load and store latencies.

We consider this to be a useful architectural improvement for many reasons:

- Performance can be *greatly* improved for those algorithms that can take advantage of the SIMD unit. We expect that the kernels of many algorithms can see speed ups greater than four.
- A wide variety of applications can take advantage of the SIMD execution unit. We believe that word processing, printer drivers, numeric applications, multimedia technology, and graphics processing can all find uses for our unit.
- The cost of the SIMD unit is low. A number of instruction encodings will need to be dedicated to it, and it may take up one or two percent of a billion-transistor die.
- If the instruction and data bandwidth permit, the super-scalar core should be able to execute in parallel with the SIMD execution unit.
- While the use of truly SIMD machines, like the MasPar, has greatly declined, the programming model is well understood. Those working with the SIMD execution unit can also take advantage of the large volumes of research in this field.

Thus, we propose to bring the power of SIMD to every user, and every application, no matter if on a laptop, desktop or server. SIMD computers went the way of the dinosaurs because while they perform some task very well, they were inefficient for many important tasks. However by combining the general-purpose processor with a SIMD execution unit we can have the best of both worlds.