Single-instruction stream multiple-data stream architecture

Introduction to SIMD Architectures

SIMD (Single-Instruction Stream Multiple-Data Stream) architectures are essential in the parallel world of computers. Their ability to manipulate large vectors and matrices in minimal time has created a phenomenal demand in such areas as weather data and cancer radiation research. The power behind this type of architecture can be seen when the number of processor elements is equivalent to the size of your vector. In this situation, componentwise addition and multiplication of vector elements can be done simultaneously. Even when the size of the vector is larger than the number of processors elements available, the speedup, compared to a sequential algorithm, is immense. There are two types of SIMD architectures we will be discussing. The first is the True SIMD followed by the Pipelined SIMD. Each has its own advantages and disadvantages but their common attribute is superior ability to manipulate vectors.

True SIMD: Distributed Memory

The True SIMD architecture contains a single contol unit(CU) with multiple processor elements(PE) acting as arithmetic units(AU). In this situation, the arithmetic units are slaves to the control unit. The AU’s cannot fetch or interpret any instructions. They are merely a unit which has capabilities of addition, subtraction, multiplication, and division. Each AU has access only to its own memory. In this sense, if a AU needs the information contained in a different AU, it must put in a request to the CU and the CU must manage the transferring of information. The advantage of this type of architecture is in the ease of adding more memory and AU’s to the computer. The

disadvantage can be found in the time wasted by the CU managing all memory exchanges.

True SIMD: Shared Memory

Another True SIMD architecture, is designed with a configurable association between the PE's and the memory modules(M). In this architecture, the local memories that were attached to each AU as above are replaced by memory modules. These M's are shared by all the PE's through an alignment network or switching unit. This allows for the individual PE's to share their memory without accessing the control unit. This type of architecture is certainly superior to the above, but a disadvantage is inherited in the difficulty of adding memory.

Pipelined SIMD

Pipelined SIMD architecture is composed of a pipeline of arithmetic units with shared memory. The pipeline takes different streams of instructions and performs all the operations of an arithmetic unit. The pipeline is a first in first out type of procedure. The size of the pipelines are relative. To take advantage of the pipeline, the data to be evaluated must be stored in different memory modules so the pipeline can be fed with this information as fast as possible. The advantages to this architecture can be found in the speed and efficiency of data processing assuming the above stipulation is met.

SIMD BASICS

Early microprocessors didn’t actually have any floating-point capabilities; they were strictly integer crunchers.? Floating-point calculations were done on separate, dedicated hardware, usually in the form of a math coprocessor.? Before long though, transistor sizes shrunk to the point where it became feasible to put a floating-point unit directly onto the main CPU die, and the modern integer/floating-point microprocessor was born.? Of course, the addition of floating-point hardware meant the addition of floating-point instructions.? For the x86 world, this meant the introduction of the x87 floating-point architecture and its (now hopelessly archaic) stack-based register model.

Actually, the addition of SIMD instructions and hardware to a modern, superscalar CPU is a bit more drastic than the addition of floating-point capability.? A microprocessor is a SISD device (Single Instruction stream, Single Data stream), and it has been since its inception.

As you can see from the above picture, a SIMD machine exploits a property of the data stream called data parallelism.? You get data parallelism when you have a large mass of data of a uniform type that needs the same instruction performed on it.? A classic example of data parallelism is inverting an RGB picture to produce its negative.? You have to iterate through an array of uniform integer values (pixels), and perform the same operation (inversion) on each one — multiple data points, a single operation.? Modern, superscalar SISD machines exploit a property of the instruction stream called instruction-level parallelism (ILP).? In a nutshell,

this means that you execute multiple instructions at once on the same data stream.? (See my other articles for more detailed discussions of ILP).? So a SIMD machine is a different class of machine than a normal microprocessor.? SIMD is about exploiting parallelism in the data stream, while superscalar SISD is about exploiting parallelism in the instruction stream.

There were some early, ill-fated attempts at making a purely SIMD machine (i. e., a SIMD-only machine).? The problem with these attempts is that the SIMD model is simply not flexible enough to accoodate general purpose code.? The only form in which SIMD is really feasible is as a part of a SISD host machine that can execute conditional instructions and other types of code that SIMD doesn't handle well.? This is, in fact, the situation with SIMD in today's market.? Programs are written for a SISD machine, and include in their code SIMD instructions.

SIMD Machines

The three SIMD machines covered in this paper are the Connection Machine by Danny Hillis, the Abacus Project at the MIT AI Lab, and the CAM-8 machine by Norman Margolus. These three machines give a pretty accurate sampling of the type of SIMD machines that were constructed as well as an idea of the motivations for creating the machines in the first place.

The Connection Machine was composed of 65, 536 bit processors. Each die consisted of 16 processors with each processor capable of communicating with each other via a switch. These 4, 096 dies formed the nodes of a 12th dimension hypercube network.

Thus, a processor was guaranteed to be within 12 hops of any other processor in the machine. The hypercube network also facilitated communication by providing alternative routes from source processor to destination. Each node was given a 12-bit node ID, and different paths between two nodes in the network could be traversed based on how the node ID was read. The network allowed for both packet and circuit-based communication for flexibility.

The second machine discussed is the Abacus machine created at the MIT AI Lab. This machine was constructed primarily for vision processing. The machine consisted of 1024 bit processing elements set in a 2D mesh. The primary concept of interest from the design was that the processing elements were configurable, and used reconfigurable bit parallel " RBP" algorithms instead of traditional bit serial computation. This means that each PE emulated logic for part of an arithmetic circuit (be it an adder, shifter, multiplier, etc) based on a RBP algorithm. The motivation for having these configurable processingelements was to save on the silicon area needed to implement arithmetic. However, because there was a necessary overhead for reconfiguration and the implementation did not easily allow for pipelining due to data dependencies, it was not clear that having configurable processing elements was a definite win.

SIMD versus Loop Pipelining

We can consider two different models for mapping loops onto coarse-grained reconfigurable architecture – SIMD and loop pipelining. SIMD computation model is efficient for computation intensive, data-parallel applications

requiring less context words to configure reconfigurable processing elements. Since data load and computation are temporarily separated in this model,

array elements are not efficiently utilized. In the case of loop pipelining, different operations in a loop can be executed simultaneously in a pipeline. With this flexibility, data load and computation can be simultaneously executed and all reconfigurable array elements can be efficiently used. In some loops, the performance of pipelining is roughly the same as the performance of SIMD. However, if a loop has frequent memory operations, the pipelining will render much higher performance.

Reconfigurable Architecture

The reconfigurable architecture that we propose consists of an ARM 926EJ-S processor, an SDRAM, a DMA controller, and a coarse-grained Reconfigurable Core Module (RCM) template, which is similar to Morphosys and specified in the DSE flow. The communication bus is AMBA AHB , which couples the ARM 926EJ-S processor and the DMA controller as master devices and the RCM as a slave device. The ARM 926EJ-S processor executes control intensive, irregular code segments and the RCM executes data-intensive, kernel code segments.

Design Space Exploration

The design space exploration (DSE) flow of coarse-grained reconfigurable architecture. A design starts from profiling and partitioning of target application and defining an architecture from the tem plate. Data intensive, regular loops are selected from the profiling result and the rest of the

application is modified to take care of synchronization. The selected loops are analyzed to determine the RCM structure from the template and the configuration words are generated. Design space exploration flow From the architecture specification, we can generate a SystemC description for fast architecture evaluation . Then the loop pipelining model is applied to the SystemC description. Binary configuration data are included in the executable code and overall performance of the application is evaluated on the transaction level platform. The transaction level modeling enables fast design space exploration at early stage . Finally, the architecture is designed at the RT level from the SystemC model and the performance is evaluated on the RTL platform. The RTL architecture is verified by FPGA prototyping.

RCM Template Architecture

RCM specification starts from the template architecture similar to Morphosys. Whereas the memory structure (frame buffer and configuration cache) of Morphosys support only the SIMD model, we support both SIMD and pipelining by modifying the memory structure.

Types of memory:-

Frame Buffer

Frame buffer (FB) of Morphosys does not support concurrency between the load of two operands and the store of result in a same column. It is not needed in SIMD mapping. However, in the case of loop pipelining, concurrent load and store operations can happen between mapped loop iterations. So we modified the FB and bit-width of data bus is specified in the DSE flow. We simply added a bank to each set. Therefore, a bank can be connected to the

write bus while the other two banks are connected to the read buses. Any combination of one-to-one mapping between the three banks and the three buses is possible.

Configuration Cache

Context memory of Morphosys is designed for broadcast configuration. So RCs in the same row or column share the same context word for SIMD operation. However, in the case of loop pipelining, each RC can be configured by different context word. So we modified the context memory and designated it as Configuration Cache. Configuration cache is composed of 64 Cache Elements(CE) and Cache Control Unit(CCU) for controlling each CE. Each CE has enough layers that enable dynamic reconfiguration and the number of layers is specified in the DSE flow. CCU supports 4 configuration modes(three broadcast modes and one individual mode) for efficient data assignment.

RC Array Execution Control Unit

If the main processor directly controls the RC array execution through AMBA AHB, it will cause high overhead in the main processor. In addition, the latency of the control will degrade the performance of the whole system, especially when dynamic reconfiguration is used. So we implement a control unit to control the execution of the RC array every cycle. The RC Array Execution Control Unit (RCECU) receives the encoded data for controlling RC execution from the main processor. The encoded data includes execution cycles, chip select, read/write mode, and addresses of FB and CCU for guaranteeing correct operations of the RC array.

RCM Specification

From profiling result, we find that ME and DCT functions occupymost of the execution time – ME takes about 70% and DCT takes about 7. 40%. Specifically, Sum of Absolute Differences (SAD) function called by ME takes about 47. 7%. Furthermore, the two functions have regular loops that fit well with the RC array. We determine the RCM structure by analyzing the DCT and ME functions. The structure is similar to Morphosys but the bit-widthof the data bus is extended to 16 and some interconnects between RCs are added for the DCT function. In the case of Morphosys, horizontal and vertical express lanes exist to guarantee connectivity between quadrants but express lanes don’t support concurrent data exchange between symmetrical RCs in the same row or column. Therefore the interconnects are added for removing data arrangement cycles . We do not expect much increase in the area with this modification but need quantitative analysis to see the effect.