Data-level Parallelism

→ Types of data-level parallelism

Arises when multiple data items are process at the same time. Two different architectures SIMD and MIMD, where SIMD is more energy efficient and the programmer can still think sequentially.

Three variations of SIMD:

- vector architectures: pipelined execution of many data operations.
- multimedia instruction set extensions: parallel execution of data operations found today in most architectures
- GPUs: similar as vector architectures but act as coprocessors.

→ Vector architectures

Grabs sets of data elements scattered about memory and transfer them into large sequential register banks and operate on those through matrix oriented computations.

Primary components of VMIPS instruction architecture are:

- vector registers: each vector register holds a single vector. VMIPS has 8 vector registers holding

64|64 - bit wide elements. (16 read ports, 8 write ports)

- vector functional units: each unit is fully pipelined and may start an operation every cycle.
- vector load/store unit: loads from or stores to memory a data vector. VMIPS moves one word per cycle. Also handles scalar loads and stores.
- scalar registers: also provide input to vector functional units. Hold addresses to pass to the load/store unit. VMIPS has 32 general purpose registers and 32 fp registers, all 64-bit wide.

The independence of elements within a vector instruction set allows scaling the functional units without carrying out additional dependency checks as superscalar require.

Pipeline stalls are required only once per vector instruction rather than once per vector element.

The execution time of a sequence of vector operations depends primarily on three factors:

- length of the operand vectors
- number of structural hazards and their kind
- data dependences among successive operations

Assume that VMIPS has all functional units having a single pipeline with an initiation rate of one element per clock cycle for individual operations, thus the execution time for a single vector operation

is approximately the vector length.

Convoy: set of vector instructions that can be potentially executed together and must not contain any structural hazards.

→ Multiple lanes

VMIPS instruction set has a property to only allow element i of one vector register to take part in operations with element i of another vector register.

Multiple parallel lanes reduces the number of clock cycles. Each lane contains a portion of the vector register bank and one execution pipeline from each vector functional unit.

→ Vector length register: loops size

VL (vector length register) controls the length of any vector operation including loads and stores. VL only works when the real vector length is ≤ to the maximum vector length (MVL). When its larger, a technique known as strip mining is applied.

→ Vector mask register: conditions in loops

The vector mask control uses a boolean vector to control the execution of a vector instruction.

When vector mask register is enabled (VM), any vector instructions operate only on vector elements whose entry in VM is one.

→ Memory banks: supporting bandwidth for load and store

Start-up time for loads and stores is high.

To maintain an initiation rate of one word fetched (stored) per clock cycle, spreading accesses across multiple independent memory banks usually delivers the desired rate, because:

- computer systems contain multiple vector processors that share the same main memory
- load and store words whose addresses are not sequential require independent bank addressing
- simultaneous access requires multiple banks and independent address control to them.

→ Stride: mult. dimension arrays

Stride is the distance separating elements to be gathered in a single register.

Once a vector is loaded into a vector register, it acts as if its successive elements are adjacent, that is, a vector processor can strides greater than one provided that there are load and store instructions with stride capabilities. (VMIPS → LVWS e SVWS).

A bank conflict might occur if:

$$\frac{\text{num. banks}}{gcd(\text{stride}, \text{num. banks})} < \text{bank busy time.}$$

→ Gather-scatter

Primary mechanism for

primary mechanism for supporting sparse matrices is gather-scatter using index vectors.

Gather: operation takes an index vector and fetches the vector whose elements are at the addresses given by adding a base address to the offsets stored in the index vector. ⟶ dense vector

Scatter: operation that sends back to memory the elements operated in this dense form, using the same index vector.