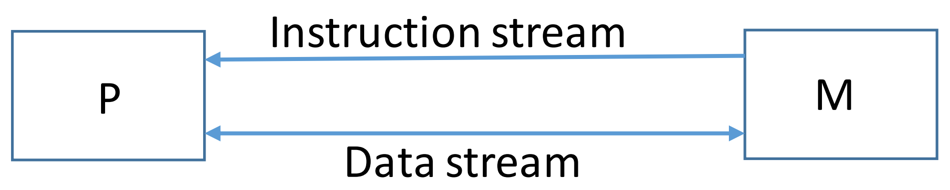
Module-2
CSEN 3104
Lecture 13

Dr. Debranjan Sarkar

SIMD Architecture

Flynn’s classification



m_I : minimum number of active instruction streams
m_D : minimum number of active data streams

- Single Instruction Stream Single Data Stream (SISD)
    - Conventional machines with single CPU
    - Capable of only scalar arithmetic
    - $m_I = m_D = 1$
- Single Instruction Stream Multiple Data Stream (SIMD)
    - Single program control unit and many independent execution units
    - Example: Illiac IV
    - $m_I = 1, m_D > 1$
- Multiple Instruction Stream Single Data Stream (MISD)
    - Several CPUs process the same data using different programs
    - Fault tolerant computers
    - $m_I > 1, m_D = 1$
- Multiple Instruction Stream Multiple Data Stream (MIMD)
    - Computers with more than one CPU with the ability to execute several programs simultaneously
    - Multi Processors containing two or more CPUs that cooperate on common computational tasks
    - $m_I > 1, m_D > 1$

SIMD Array Processor

- A synchronous array of parallel processors
- Consists of multiple processing elements (PEs) under the supervision of one control unit (CU)
- Can handle single instruction and multiple data (SIMD) streams
- Specially designed to perform vector computations over matrices or arrays of data
- Two basic architectural organizations of SIMD computers
    - Array processors using distributed memories
    - Associative processors using content-addressable (or associative) memory
- Two configurations of Array Processors
    - Illiac IV (developed by the Illinois Automatic Computer team of University of Illinois)
    - Burroughs Scientific Processor (BSP)

Configuration of Illiac IV

- Show the block diagram
- N numbers of identical synchronized processing elements (PEs) which can concurrently do the same operation on different data
- All these PEs are under the control of one CU
- Each PE is basically an ALU with working registers and local memory (PEM) for the storage of distributed data
- The CU has its own main memory for the storage of programs
- The system and user programs are executed under the control of CU
- Scalar and control type instructions are executed in the CU
- Vector instructions are broadcast to the PEs for distributed execution
- Thus spatial parallelism is achieved through multiple arithmetic units (PEs)

Configuration of Illiac IV

- All the PEs perform the same function synchronously in a lock-step fashion under the command of the CU
- Vector operands are distributed to the PEMs before parallel execution in the array of PEs
- The distributed data can be into the PEMs from an external source via the system data bus or via the CU in a broadcast mode using the control bus
- Masking schemes are used to enable or disable each PE to participate in the execution of a vector instruction
- Data exchanges among the PEs are done via an interconnection network which performs all necessary data routing and manipulation functions
- This interconnection network is under the control of CU

Configuration of Illiac IV

- The Array Processor is normally interfaced to a host computer through the CU
- The host computer is a general purpose machine which serves as the operating manager of the entire system, consisting of the host and the processor array
- The functions of the host computer include resource management, peripheral and IO supervision
- The control unit of the processor array directly supervises the execution of programs, whereas
- The host machine performs the executive and IO functions with the outside world
- So, an array processor can be considered a back-end attached computer

SIMD Computer

- AN SIMD computer is characterized by a set of parameters:

  C = <N, F, I, M>

- N = Number of processing elements (PEs) in the system. For Illiac IV, N = 64
- F = a set of data routing functions provided by the interconnection network
- I = the set of machine instructions for scalar-vector, data routing, and network manipulation operation
- M = the set of masking schemes, where each mask partitions the set of PEs into two disjoint subsets of enabled PEs and disabled PEs
- This model provides a common basis for evaluating different SIMD machines

Components in a Processing Element (PE_i)

- Show figure
- Each PE_i has
  - its own memory PEM_i
  - A set of working registers and flags (A_i,B_i, C_i and S_i)
  - An Arithmetic and Logic Unit (ALU)
  - A local Index Register (I_i), an address register (D_i) and a data routing register (R_i)
  - The R_i of each PE_i is connected to the R_j of other PEs via the interconnection network
  - During data transfer among PEs, the contents of the R_i registers are transferred
  - The inputs and outputs of the R_i are totally isolated
  - The i^{th} PE is denoted by PE_i where the index i is the address of PE_i
  - Let the total number of PEs be $N = 2^m$, then $m = \log_2 N$ binary digits are needed to encode the address of a PE
  - The address register D_i is used to hold the m-bit address of the PE_i
  - Some array processors may use two routing registers – one for input and the other for output

Masking

- During each instruction cycle, each PE_i is either
  - in the active mode, or
  - in the inactive mode
- In case a PE_i is active, it executes the instruction, broadcast to it by the CU, otherwise it won’t
- The masking schemes are used to specify the status flag S_i of PE_i
- $S_i = 1$ indicates an active PE_i and $S_i = 0$ indicates an inactive PE_i
- In the CU, there is a global
  - index register (I) and
  - N-bit masking register (M)
- The collection of S_i flags for i = 0, 1, 2, …., N-1 forms a status register S for all the PEs
- The bit patterns in registers M and S are exchangeable under the control of CU when masking is to be set

Thank you

Module-2
CSEN 3104
Lecture 14

Dr. Debranjan Sarkar

SIMD Architecture

Use of indexing to address the local memories in parallel at different local addresses

- Consider an array of n X n data elements:

  $$A = \{A(i,j), 0 \le i, j \le (n-1)\}$$

- Elements of jth column of A are stored in n consecutive locations of PEM_j [say from location 200 to location (200+n-1)] (assume $n \le N$)
- We want to access the principal diagonal elements A(j,j) for j=0, 1 , …, (n-1) of the array A
- The CU must generate and broadcast an effective memory address 200
- The local index registers must be set to be $I_j = j$ for j = 0, 1, …, (n-1) in order to convert the global address 200 to local address $200 + I_j = 200 + j$ for each PEM_j
- Within each PE, there is a separate memory address register for holding these local addresses

Data Routing Mechanisms

- Execution of the following vector instruction in an array of N processing elements (PEs)
- The sum S(k) of the first k components in a vector A = $(A_0, A_{1, \ldots\ldots,} A_{n-1})$ is desired for each k from 0 to (n-1)
- We need to compute the following n summations:

$$S(k) = \sum_{i=0}^{k} A_i \qquad \text{for } k = 0, 1, \ldots\ldots, (n-1)$$

Data Routing Mechanisms

- These n vector summations can be computed recursively by going through the following (n-1) iterations:

$$S(0) = A_0$$

$$S(k) = S(k-1) + A_k \qquad \text{for } k = 1, 2, \ldots\ldots, (n-1)$$

- For n = 8, the above recursive summation is implemented in an array processor with N = 8 processing elements (PEs)
- $\text{Log}_2 n = 3$ steps are required
- Both data routing and PE masking are used
- Show diagram
- Initially each A_i, residing in PEM_i is moved to the R_i register in PE_i for i = 0,1,2,…,7

Data Routing Mechanisms

- In the first step, A_i is routed from R_i to R_{i+1} and added to A_{i+1} with the resulting sum $A_i + A_{i+1}$ in R_{i+1} for $i = 0,1,2,\ldots,6$
- In step 2, the intermediate sums in R_i are routed to R_{i+2} for $i = 0$ to 5
- In step 3, the intermediate sums in R_i are routed to R_{i+4} for $i = 0$ to 3
- Thus, the final value of PE_k will be $S(k)$ for $k = 0,1,2,\ldots,7$

Data Routing Mechanisms

- In step 1, PE_7 is not involved in data routing (receiving but not transmitting)
- In step 2, PE_7 and PE_6 are not involved in data routing
- In step 3, PE_7, PE_6, PE_5 and PE_4 are not involved in data routing
- These unwanted PEs are masked off during the corresponding steps
- During the addition operations
  - PE_0 is disabled in step 1
  - PE_0 and PE_1 are made inactive in step 2
  - PE_0, PE_1, PE_2 and PE_3 are masked off in step 3
- The PEs that are masked off in each step depend on the operation (data-routing or addition)
- Thus the masking pattern keep changing in different operation cycles
- Masking and routing operation are much more complicated when the vector length $n > N$

Thank you