Parallel Processors

Session 10

Multivector and SIMD Computers
Vector Processing Principles

• Vector:
  – A set of scalar data items
  – All of the same type
  – Stored in memory

• Stride:
  – Address increments between successive elements of a vector

• Vector Processor:
  – Hardware resources to perform vector operations:
    • Vector registers
    • Functional pipelines
    • Processing elements
    • Register counters

• Vector Processing:
  – Arithmetic or logic operations on vectors

• Vectorization:
  – Conversion from scalar code to vector code
Performance

• Vector processing:
  – Faster
  – More efficient
  – Reduced software overhead
    • Loop control
    • Memory access
  – Matches with pipelining mechanism

• Speedup:
  – Vectorization ratio
  – Speed ration between vector and scalar operations

• Costs:
  – Hardware costs
  – Compiler (vectorizing compiler or vectorizer)
  – Programming skills
Vector Instruction Types

- Vector-Vector Instructions
- Vector-Scalar Instructions
- Vector-Memory Instructions
- Vector Reduction Instructions
- Gather and Scatter Instructions
- Masking Instructions
Vector-Vector Instructions

- $V_i \rightarrow V_j$
- $V_j \times V_k \rightarrow V_i$
- Examples:
  - $V_1 = \sin (V_2)$
  - $V_3 = V_1 + V_2$
Vector-Scalar Instructions

\[ s \times V_i \rightarrow V_j \]
Vector-Memory Instructions

- Vector load:
  - $M \rightarrow V_i$
- Vector store:
  - $V_i \rightarrow M$
Vector Reduction Instructions

- **Mappings:**
  - $V_i \rightarrow S_j$
  - $V_i \times V_j \rightarrow S_k$

- **Examples:**
  - maximum of all elements
  - minimum of all elements
  - sum of all elements
  - mean value of all elements
  - dot product:
    - $s = \sum a_i \times b_i$ for $A = (a_i)$ and $B = (b_i)$
Gather and Scatter Instructions

• Gather:
  - $M \rightarrow V_1 \times V_0$
  - $V_1$ contains the data and $V_0$ is used as an index
  - Fetches from memory the nonzero elements of a sparse vector using indices that themselves are indexed

• Scatter:
  - $V_1 \times V_0 \rightarrow M$
  - $V_1$ contains the data and $V_0$ is used as an index
  - Stores a vector into memory in a sparse vector whose nonzero entries are indexed
Gather:

- \( M \rightarrow V_1 \times V_0 \)
- \( V_1 \) contains the data and \( V_0 \) is used as an index
- Fetches from memory the nonzero elements of a sparse vector using indices that themselves are indexed
Scatter

- Scatter:
  - $V_1 \times V_0 \rightarrow M$
  - $V_1$ contains the data and $V_0$ is used as an index
  - Stores a vector into memory in a sparse vector whose nonzero entries are indexed
Masking Instructions

- A mask vector is used to:
  - Compress a vector to a shorter index vector
  - Expand a vector to a longer index vector

- Mapping:
  - $V_0 \times V_m \rightarrow V_1$
Masking Instructions

- A mask vector is used to:
  - Compress a vector to a shorter index vector
  - Expand a vector to a longer index vector

- Mapping:
  - $V_0 \times V_m \rightarrow V_1$
Vector Operands and Memory Access

- Arbitrary length
- Arbitrary distribution in memory
  - A matrix is either row major or column major
  - Each row or column can be used as a vector
  - Vector elements are not necessarily in contiguous memory locations
    - Row elements are in contiguous locations with stride n (n is the matrix order)
    - Column elements are in locations with stride n
    - Diagonal elements are in locations with stride n+1

- To access a vector in memory:
  - Base address
  - Stride
  - Length

- Fast vector access necessary to match the pipeline rate
- The access path itself is pipelined: access pipe
C-Access Memory Organization

- Vector access scheme from interleaved memory modules
- m-way low-order interleaved memory structure
- Allows m memory words to be accessed **concurrently**
- This is called C-access
S-Access Memory Organization

- Similar to low-order interleaved memory
  - High order bits select modules
  - Words from modules are latched at the same time
  - Low order bits select words from data latches
  - This is done through the multiplexed with higher speeds (minor cycles)
- Allows **simultaneous** access
- This is called S-access
Interleaved Fetch and Access

- If the minor cycle is selected $1/m$
  - $m$ words (one row) is accessed in 2 memory (major) cycles
- If fetch and access to the latches are interleaved
  - $m$ words is accessed in 1 memory cycle
C/S Access

- C-access and S-access are combined
- $n$ access busses with $m$ interleaved memory modules
- The $m$ modules on each bus are $m$-way interleaved to allow C-access
- The $n$ busses operate in parallel to allow S-access
Balanced Vector/Scalar Ratio

- In a supercomputer separate hardware resources are dedicated to concurrent vector and scalar operations.
- Vector processing is needed for regularly structured parallelism in scientific and engineering computations.
- For a better performance these two types of operations must be balanced.
- Vector balance point:
  - Percentage of vector code to achieve equal utilization of vector and scalar hardware.
  - In best case none of the vector and scalar hardware is idle at any time.
Vector Balance Point

• Percentage of vector code to achieve equal utilization of vector and scalar hardware

• Example:
  – System capability:
    • 9 Mflops in vector mode
    • 1 Mflops in scalar mode
  – Equal time will be spent in each mode if the code is:
    • 90% vector code
    • 10% scalar code
  – The vector balance point is 0.9
Compound Vector Processing
CVF

• Compound Vector Function:
  – A composite function of vector operations converted from a looping structure of linked scalar operations
Example

- \( \text{X(I)} \) and \( \text{Y(I)} \) are two source vectors with length \( N \) in memory.
Vectorized Code

\[
\begin{align*}
M(x : x + N - 1) & \rightarrow V1 \\
M(y : y + N - 1) & \rightarrow V2 \\
S \times V1 & \rightarrow V1 \\
V2 + V1 & \rightarrow V2 \\
V2 & \rightarrow M(y : y + N - 1)
\end{align*}
\]
Vector load \quad \text{Vector load} \quad \text{Vector multiply} \quad \text{Vector add} \quad \text{Vector store}

- Expressed as a CVF:

\[
Y(1 : N) = S \times X(1 : N) + Y(1 : N)
\]

\[
Y(I) = S \times X(I) + Y(I)
\]
## Compound Vector Functions

<table>
<thead>
<tr>
<th>One-dimensional compound vector functions</th>
<th>Maximum chaining degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V1(I) = V2(I) + V3(I) \times V4(I) )</td>
<td>2</td>
</tr>
<tr>
<td>( V1(I) = B(I) + C(I) )</td>
<td>3</td>
</tr>
<tr>
<td>( A(I) = V1(I) \times S + B(I) )</td>
<td>4</td>
</tr>
<tr>
<td>( A(I) = V1(I) + B(I) + C(I) )</td>
<td>5</td>
</tr>
<tr>
<td>( A(I) = B(I) + S \times C(I) )</td>
<td>5</td>
</tr>
<tr>
<td>( A(I) = B(I) + C(I) + D(I) )</td>
<td>6</td>
</tr>
<tr>
<td>( A(I) = Q \times V1(I) \times (R \times B(I) + C(I)) )</td>
<td>7</td>
</tr>
<tr>
<td>( A(I) = B(I) \times C(I) + D(I) \times V1(I) )</td>
<td>7</td>
</tr>
<tr>
<td>( A(I) = V1(I) + (1 / A(I) + 1 / B(I)) + \log(V2(I)) )</td>
<td>8</td>
</tr>
<tr>
<td>( A(I) = \sqrt{V2(I)} + \sin(B(I) + C(I)) + V3(I) )</td>
<td>8</td>
</tr>
<tr>
<td>( A(I) = B(I) \times C(I) + D(I) \times E(I) \times S )</td>
<td>9</td>
</tr>
<tr>
<td>( A(I) = (A(I) + B(I) \times C(I) + D(I)) \times (I) )</td>
<td>10</td>
</tr>
</tbody>
</table>

Note: \( V_i(I) \) are vector registers in the processor. \( A(I), B(I), C(I), D(I), \) and \( E(I) \) are vectors in memory. Scalars are indicated as \( Q, R, \) and \( S \) are available from scalar registers in the processor. The chaining degrees include both memory-access and functional pipeline operations.
Strip-Mining

- Segmentation of a long vector in memory
- Fixed length segments
- Loading and processing the segments one segment at a time
- Segment length matches the vector register size
- More flexible if vector register size can be configured
- The vector register used for the vector is not released until all the segments are processed
Vector Loop

• The program construct for processing long vectors is called a vector loop
• Strip-mining is a part of the vector loop
• All is done in hardware
Chaining

• Chaining of multiple pipelines is used for concurrent processing of several vector operations
• A CVF is a candidate for chaining
• Actual implementation depends on the hardware
Functional Units in the Chain

• Linked vector operations must follow a linear data flow pattern
• Functional pipeline units must be independent of each other
• Same unit cannot be assigned to execute more than one instruction in the same chain
• Vector registers must be used as interface between functional pipelines
Examples of Pipeline Chaining

Chaining with only one memory-access pipe compared to chaining with three memory-access pipes
The Vector Registers

• The successive output results of a pipeline are fed into the vector register one element per cycle
• The vector register is then used as an input register for the next pipeline unit in the chain
• The interface registers must be able to pass one vector element per cycle between adjacent pipelines
Timing in Various Chaining Scenarios in the Example

- Sequential execution without chaining

- Chaining with only one memory access pipe

- Chaining with three memory access pipes

Captions:
- $s =$ Memory access latency
- $m, a =$ Multiply/add latencies.
- $n =$ Time to process $n$ elements, one per cycle.
Multipipeline Networking

- Generalization of the idea of linking vector operations (chaining)
- Instead of a linear chain, a pipeline net (pipenet) is built
- Multiple functional pipelines are linked to achieve systolic computation of CVFs
- A systolic array is formed with a network of functional units which are locally connected and operate synchronously
- Unlike a systolic architecture which is fixed, a pipenet can be configured dynamically
Implementation of a Pipenet

(a) A program graph

(b) The pipenet

(c) A crossbar implementation

\[ E(I) = \frac{[A(I) \times B(I) + B(I) \times C(I)]}{[B(I) \times C(I) \times [C(I) + D(I)]]} \]
Generalized Pipenet Model

Diagram showing a Generalized Pipenet Model with components labeled as 'Register File', 'Buffered Crossbar Network with Programmable Delays (BCN1)', 'Buffered Crossbar Network with Programmable Delays (BCN 2)', and 'MPX'. The diagram includes arrows indicating the flow of data or signals through these components.
SIMD Computers
SIMD Computers

• Vector processing can be carried out by SIMD computers
• Vector instruction’s operands must have a fixed length of \( n \) equivalent to the number of PEs
• Two models:
  – Distributed memory model
  – Shared memory model
Distributed-Memory Model

- Spatial parallelism is explored
  - An array of PEs
  - An array control unit
- Program and data are loaded into the control memory through a host unit
- Instructions are sent to the control unit for decoding
- A scalar or program control operation is directly executed by a scalar processor attached to the control unit
- A vector instruction will be broadcast to all PEs for execution
- Partitioned data sets are distributed to all the local memories attached to the PEs through a vector data bus
Distributed-Memory Model

- The PEs are synchronized in hardware by the control unit.
- The same instruction is executed by all the PEs in the same cycle.
- Masking logic is provided to disable any PE from participating in a given instruction cycle.
- The PEs are interconnected by a data-routing network which performs inter-PE data communications.
- The data-routing network is under program control through the control unit.
Distributed-Memory SIMD Model

[Diagram showing a distributed-memory SIMD model with components such as Scalar Processor, Array Control Unit, Control Memory (Program and Data), Host Computer, Mass Storage, and Data Routing Network.]
Shared-Memory SIMD Model