Lecture 12: System Organization 1

CS 5515
Computer Architecture

VA Tech
High Performance Computer Architecture

• Computer system performance is defined by three factors
  – time to execute a program is a function of the number of instructions to execute
  – the average number of clock cycles required per instruction
  – and the clock cycle time: \( T = n_i \times CPI \times t_c \)

• Lowering the clock cycle time is primarily a matter of engineering
  – use of more advanced materials or production techniques for the construction of smaller, faster, and more efficient circuits

• Parallelism is a common to all techniques for designing architectures that improve the other two factors
  – replicating basic components in the system
  – e.g. using four adder/multiplier units vice one inside the CPU, or connect two or more memories to the CPU in order to increase bandwidth, or connect two or more processors to one memory in order to increase the number of instructions executed per unit time, or even replicate the entire computer in a network
Parallelism has existed in the minds of computer architects since Charles Babbage.

The granularity of parallel tasks is related to the level of parallelism:
  - “Large grain parallel system” is one in which the operations that run in parallel are fairly large, on the order of entire programs.
  - “Small grain parallel systems” divide programs into very small pieces, in some cases only a few instructions.

Parallelism may be classified in distinct levels:

- **Job level parallelism**
  - at the multi-user level, have more user systems so more jobs are running at any one time (although any one user's job will not run faster)
  - distinction between throughput (number of jobs per day) and latency (the time to execute a program).
— at the single computer level, jobs are treated as a collection of independent tasks with several jobs may reside in memory at the same time with only one in execution at any given time

  • when a job requires I/O services, the operation is initiated, the job requiring the service is suspended, and another job is put into execution state
  • after the I/O operation completes, control passes back to the original job and execution continues
  • CPU and I/O system are functioning in parallel

• **Program level parallelism**
  — when a single program is broken down into constituent parts
    • e.g. processing separate quadrants of a matrix
  — generally manifested in two ways:
    • independent sections of a given program
    • individual iterations of a loop where there are no dependencies between iterations
  — may be exploited by multiple processors or multiple functional units
  — example: code segment for calculating sums:
    DO 10 I=1,N
    A(I) = B(I) + C(I)
    10 CONTINUE
    • the sums are independent, i.e. the calculation of $b_i + c_i$ does not depend on $b_j + c_j$ for any $j < i$
    • sums can be done in any order
    • a machine with $n$ processors could do them all at the same time
Instruction level parallelism

- mostly invisible to users, i.e., it is below the level of the architecture and in the domain of computer organization
  - compilers reorganize programs to exploit this form of parallelism
  - quality of compilers varies greatly and one may have to structure the code to help the compiler make maximum use of the hardware
- Pipelines are the most common way of implementing this type of parallelism
- individual instructions may be overlapped
  - e.g., it is common to find a load instruction, which copies a value from memory to an internal CPU register, overlapped with an arithmetic instruction
- or a given instruction may be decomposed into suboperations with the suboperations overlapped
  - e.g., pipeline for arithmetic processing

Designers must decide to use a relatively small number of powerful processors or a large number of simple processors to achieve the desired performance
- the latter approach is “massively parallel”.
- Most reserve the term for systems with 1000 or more individual processors
High Performance Computer Architecture 5

• Extremes
  – systems built by Cray Research Inc. that consist of two to sixteen very powerful vector processors
  – arrays of tens of thousands of very simple processors, exemplified by the CM-1 from Thinking Machines Corporation, which has up to 65,536 single-bit processors

• A small number of powerful processors are simpler to interconnect and lend themselves to an implementation of memory organizations that make the systems relatively easy to program
  – however, such processors are very expensive to build, power and cool

• Other architects have used commodity microprocessors
  – offer great economies of scale at the expense of more complex interconnection strategies
  – the rapid increase in the power of microprocessors, allows arrays of a few hundred processors to have the same theoretical peak performance of the fastest machines offered by Cray Research Inc.
• **Arithmetic and bit level parallelism**
  - lowest level, mainly of concern to designers of ALUs
  - e.g. a 64-bit sum can be computed by adding all 64 bits at once (the carry into the most significant bits can be predicted and computed almost as fast as the sum of any two bits),
  - or the operation can be broken into 4-bit pieces and the entire sum computed in 16 cycles
Flynn's Taxonomy

• The most popular taxonomy of the different types of parallel systems was defined by Flynn in 1966
  – classification is based on the notion of a stream of information
  – two types of information flow into a processor: instructions and data
  – can be separated into two independent streams, whether or not the information actually arrives on a different set of wires
  – a processor has a "Harvard architecture" if it has two separate memory channels, one for instructions and one for data

• Flynn's taxonomy classifies machines according to whether they have one stream or more than one stream of each type

• Four combinations are:
  – *SISD* (single instruction stream, single data stream)
  – *SIMD* (single instruction stream, multiple data streams)
  – *MISD* (multiple instruction streams, single data stream)
  – *MIMD* (multiple instruction streams, multiple data streams)
Flynn's Taxonomy 2

**Single Data Stream**

- **SISD**: a single processor fetches instructions and performs all data processing operations

**Multiple Data Streams**

- **SIMD**: a single instruction processor (K) fetches instructions, broadcasts orders to processing elements (P). Typically each PE has its own data memory $M_d$

**Single Instruction Stream**

- **MISD**: a single data stream is operated on by several processors, each with an instruction stream from its own instruction memory $M_i$

**Multiple Instruction Streams**

- **MIMD**: processors independently fetch instructions and operate on data. Processors communicate directly (as shown) or through shared memory.
SISD Computers

• Conventional single processor computers are classified as SISD systems
  – each arithmetic instruction initiates an operation on a data item taken from a single stream of data elements
    • most contemporary microprocessors fit into this category

• Vector processors such as the Cray-1 and its descendants are often classified as SIMD machines, although they are more properly regarded as SISD machines
  – Vector processors pass successive vector elements through separate pieces of hardware dedicated to independent phases of a complex operation
  – e.g., to add two numbers such as $3.4 \times 2^3$ and $1.6 \times 2^2$, the numbers must have the same exponent
    • the processor must shift the mantissa (and decrement the exponent) of one number until its exponent matches the exponent of the other number
    • in this example $3.4 \times 2^3$ is adjusted to $6.8 \times 2^2$ and the sum is $8.4 \times 2^2$
    • a vector processor is specially constructed to feed a data stream into the processor at a high rate
      – while part of the processor is adding the mantissas in the pair $(a_i, b_i)$ another part of the processor is adjusting the exponents in $(a_i, b_i)$
SISD Computers 2

- The ambiguity over the classification of vector machines depends on how one views the flow of data
  - a static `"snapshot" of the processor during the processing of a vector shows several pieces of data being operated on at one time
    - therefore one instruction (a vector add) initiates several data operations (adjust exponents, add mantissas, etc.) and the machine might be classified SIMD
  - a more dynamic view shows that there is just one stream of data, and elements of this stream are passed sequentially through a single pipeline (which implements addition in this example)
SIMD Computers

- SIMD machines have one instruction processing unit, sometimes called a controller
  - indicated by a K in the PMS notation
  - the control unit is responsible for fetching and interpreting instructions
  - when it encounters an arithmetic or other data processing instruction, it broadcasts the instruction to all PEs, which then all perform the same operation
    - e.g., for the instruction $\text{add } R3,R0$, Each PE would add the contents of its own internal register R3 to its own R0.

- SIMD machines have several data processing units, generally called \textit{D-units} or \textit{processing elements} (PEs)
  - a PE can be deactivated to allow for needed flexibility in implementing algorithms
    - thus on each instruction, a PE is either idle, in which case it does nothing, or it is active, in which case it performs the same operation as all other active PEs
  - each PE has its own memory for storing data
    - a memory reference instruction, e.g., $\text{load } R0,100$ directs each PE to load its internal register with the contents of memory location 100

- Thinking Machines CM-1 is a recent example of a SIMD machine
SIMD Computers 2

- One advantage of SIMD machines is a savings in the amount of logic:
  - anywhere from 20% to 50% of the logic on a typical processor is devoted to control (fetching, decoding, and scheduling instructions).
  - the remainder is used for on-chip storage (registers and cache) and the logic required to implement the data processing (adders, multipliers, etc.).
  - in an SIMD machine, only one control unit fetches and processes instructions, so more logic can be dedicated to arithmetic circuits and registers.

- Vector processing is performed on an SIMD machine by distributing elements of vectors across all data memories:
  - e.g., with two vectors, a and b, and a machine with 1024 PEs.
    - store $a_i$ in location 0 of memory $i$ and $b_i$ in location 1 of memory $i$.
    - to add $a$ and $b$, the machine would tell each PE to load the contents of location 0 into one register, the contents of location 1 into another register, add the two registers, and write the result.
  - as long as the number of PEs is greater than the length of the vectors, vector processing on an SIMD machine is done in constant time.
    - i.e. it does not depend on the length of the vectors.
  - Vector operations on a pipelined SISD processor, take time that is a linear function of vector length.
MISD Computers

• Few machines in this category
  – none that have been commercially successful or had any impact on computational science

• One type of system that fits the description of an MISD computer is a systolic array
  – a network of small computing elements connected in a regular grid
  – all the elements are controlled by a global clock
  – on each cycle, an element will read a piece of data from one of its neighbors, perform a simple operation (e.g. add the incoming element to a stored value), and prepare a value to be written to a neighbor on the next step

• Could make a case for pipelined vector processors fitting in this category since each step of the pipeline corresponds to a different operation being performed to the data as it flows past that stage in the pipe
  – there have been pipe lined processors with programmable stages, i.e. the function that is applied at each location in the pipeline could vary
  – since the pipeline stage did not fetch its operation from a local memory so it would be difficult to classify it as a``processor"
MIMD Computers

• The most diverse classification in Flynn's taxonomy, includes a wide variety of designs:
  – machines with processors and memory units specifically designed to be components of a parallel architecture
  – large scale parallel machines built from "off the shelf" microprocessors
  – small scale multiprocessors made by connecting four vector processors together

• Continued improvement in network communication and the development of software packages that allow programs running on one machine to communicate with programs on other machines
  – can use local networks of workstations as MIMD systems

• Computer systems with two or more independent processors have been available commercially for a long time
  – Burroughs sold dual processor versions of its B6700 systems in the 1970s.
    • these were rarely used to work on the same job
MIMD Computers 2

- Multiprocessors of this era were intended to be used for job level parallelism, i.e. each would run a separate program.
- Parallel processing, in the sense of using more than one processor in the execution of a single program:
  - an active area in corporate and academic research labs since the early 1970s
  - wide commercial availability in the mid 1980s
  - began to approach top-of-the-line vector processors in computing power by the early 1990s these system
  - the trend for future high performance computing
Other Taxonomies

• Weaknesses of the Flynn taxonomy
  – the empty MISD category
  – the difficulty in classifying vector processors
  – in the MIMD category, all arrays of processors are lumped together regardless of how they are connected and how they view memory
    • these characteristics can have a dramatic effect on performance

• In attempting to expand or modify Flynn’s taxonomy, various authors have
  – expanded the SIMD category to four subcategories
  – subdivided the MIMD category into shared memory systems, distributed memory systems and reconfigurable systems
  – divided the MIMD category into systems with shared memory and those without shared memory
Other Taxonomies 2

- One addition to the Flynn taxonomy that has become very popular is Single Program / Multiple Data stream (SPMD)
  - represents a style of computing rather than an architecture
  - physically the system is an MIMD multiprocessor because there are several independent processors, each with its own data set and program memory
  - however, the same program is executed by each processor, and the processors are synchronized periodically
    - much simpler way to approach an MIMD system than to have to manage many individual instruction streams
    - provides more flexibility than the SIMD system because different processors may be at different parts of the program at any time

- One ambitious attempt at a taxonomy treats pipelined vector processors as a distinct architecture and differentiates among the many multiprocessor possibilities
  - uses a complex notation systems resembling chemical notation for organic compounds
  - provides a unique identifier for all of the systems that have been proposed or manufactured