Advanced Parallel Architecture

Lesson 2

Annalisa Massini - 2014/2015

Introduction

Architectural Trends

- Advances in technology determine what is possible
- Architecture translates the potential of the technology into performance and capability
- Two ways to improve performance: ***parallelism*** and ***locality*,** **but** these two compete for the same resources:
  - multiple operations performed in parallel
    - reduction of number of cycles to execute the program
    - **but** need for resources supporting simultaneous activities
  - data references performed close to the processor
    - accessing deeper levels of the storage hierarchy avoided
    - **but** need for resources providing local storage

Architectural Trends

- The best performance is obtained by an *intermediate strategy* which devotes resources to exploit a degree of parallelism and a degree of locality
- Indeed, parallelism and locality interact in interesting ways in systems of all scales, from within a chip to across a large parallel machine
- The history of computer architecture is divided into four generations identified by the basic logic technology:
  - tubes
  - transistors
  - integrated circuits
  - VLSI

Generations of Computer

- Vacuum tube - 1946-1957
- Transistor - 1958-1964
- Small scale integration - 1965 on
  - Up to 100 devices on a chip
- Medium scale integration - to 1971
  - 100-3,000 devices on a chip
- Large scale integration - 1971-1977
  - 3,000 - 100,000 devices on a chip
- Very large scale integration - 1978 -1991
  - 100,000 - 100,000,000 devices on a chip
- Ultra large scale integration – 1991 -
  - Over 100,000,000 devices on a chip

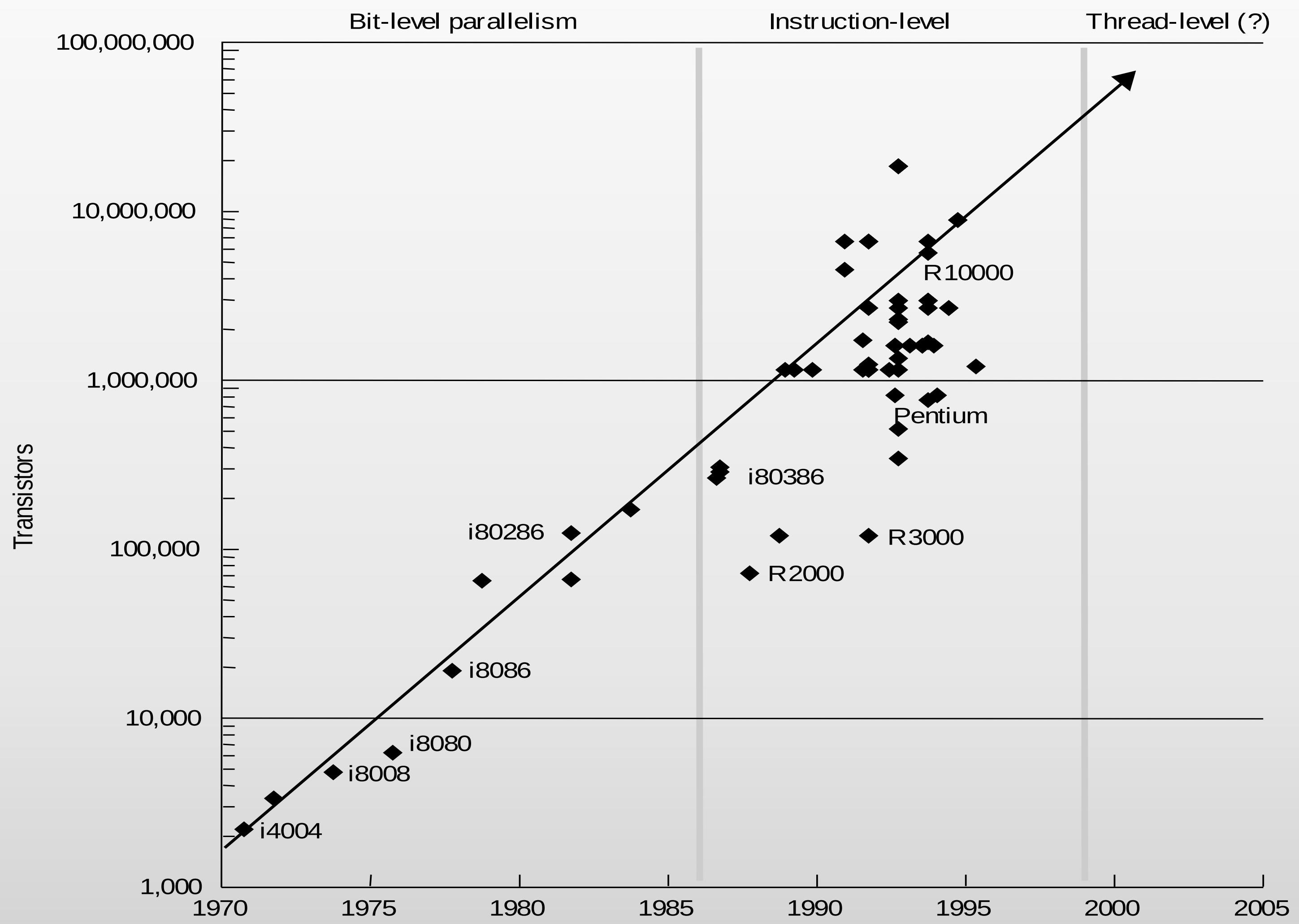
Architectural Trends

- There has been tremendous architectural advance
- The strongest delineation in VLSI generation is the kind of parallelism that is exploited
- The period up to about 1985 is dominated by advancements in ***bit-level parallelism***, with 4-bit microprocessors replaced by 8-bit, 16-bit, and so on
- Doubling the width of the datapath reduces the number of cycles required to perform a full 32-bit operation
- This trend slows once a 32-bit word size is reached in the mid-80s, and the adoption of 64-bit operation is reached decade later

Architectural Trends

- Further increases in word-width will be driven by demands for improved floating-point representation and a larger address space, rather than performance.
- With address space requirements growing by less than one bit per year, the demand for 128-bit operation appears to be well in the future.
- The early microprocessor period was able to reap the benefits of the easiest form of parallelism: bit-level parallelism in every operation.
- ***Inflection*** point in the microprocessor growth curve has been when 32-bit word operation combined with use of cache (late ‘80s).

Phases in “VLSI” Generation



Architectural Trends

- The period from the mid-80s to mid-90s is dominated by advancements in ***instruction-level parallelism***
- The basic steps in instruction processing (instruction decode, integer arithmetic, and address calculation) can be performed in ***a single cycle***
- By using caches, the instruction fetch and data access can be performed in ***a single cycle***, most of the time
- Using the RISC approach - care in the instruction set design - it is straightforward to ***pipeline*** the stages of instruction execution → an instruction is executed almost every cycle, on average

Architectural Trends

- The parallelism in the steps of instruction processing is exploited across a small number of instructions
- In addition, advances in compiler technology made instruction pipelines more effective
- The mid-80s microprocessor-based computers consisted of a set of chips: an integer processing unit, a floating-point unit, a cache controller, and SRAMs for the cache data and tag storage
- These components were coalesced into ***a single chip***, reducing the cost of communicating among them

Architectural Trends

- In addition to pipelining individual instructions, it became very attractive to fetch ***multiple instructions*** at a time and issue them in parallel to distinct function units whenever possible
- This form of instruction level parallelism came to be called ***superscalar*** execution
- More function units were added, more instructions were fetched at time, and more instructions could be issued in each clock cycle to the function units

Architectural Trends

- In order to satisfy the increasing instruction and data bandwidth requirement, *larger and larger caches* were placed on-chip with the processor, further consuming the ever increasing number of transistors

- With the *processor and cache on the same chip*, the path between the two could be made very wide to satisfy the bandwidth requirement of multiple instruction and data accesses per cycle

Architectural Trends

- However, as more instructions are issued each cycle, the performance impact of each control transfer and each cache miss becomes more significant
- A control transfer may have to wait for the depth, or *latency*, of the processor pipeline, until a particular instruction reaches the end of the pipeline and determines which instruction to execute next
- Similarly, instructions which use a value loaded from memory may cause the processor to wait for the latency of a cache miss

Architectural Trends

- Processor designs in the 90s deploy a variety of *complex instruction processing mechanisms* in an effort to reduce the performance degradation in superscalar processors
- Sophisticated *branch prediction techniques* are used to avoid pipeline latency by guessing the direction of control flow before branches are actually resolved
- Larger, *more sophisticated caches* are used to *avoid* the latency of cache misses
- Instructions are scheduled dynamically and allowed to complete *out of order* so if one instruction encounters a miss, other instructions can proceed ahead of it, as long as they do not depend on the result of the instruction

Architectural Trends

- A larger window of instructions that are waiting to issue is maintained within the processor and whenever an instruction produces a new result, several waiting instructions may be issued to the function units
- These complex mechanisms allow the processor to *tolerate* the latency of a cache-miss or pipeline dependence when it does occur
- However, each of these mechanisms place a ***heavy demand*** on chip resources and a very heavy design cost

Architectural Trends

- Given the expected increases in chip density, the instruction level parallelism within a single thread of control was overcome
- The processors and their interconnect are all implemented on a single silicon chip and the new technology is multi-core processor
- The emphasis shifts to ***thread level parallelism***, parallelism available as multiple processes or multiple threads of control within a process

Architectural Trends

- By the early 2000s, CPU designers were thwarted from achieving higher performance from *instruction level parallelism techniques*
- The **growing disparity** between CPU operating frequencies and main memory operating frequencies as well as escalating CPU power dissipation implied more esoteric instruction level parallelism techniques
- CPU designers realize that to aggregate performance of multiple programs was more important than the performance of a single thread or program
- This is evidenced by the proliferation of dual and multiple core CMP (chip-level multiprocessing) designs

Architectural Trends

- Parallel execution at thread level
- Examples:
  - hyper-threading – 2 threads on the same pipeline executed in parallel (up to 30% speedup)
  - multi-core architectures – multiple CPUs on a single chip
  - multiprocessor systems (parallel systems)
  - manycore architectures (GPUs)

Supercomputers

- The development of parallel architecture is driven by the request to achieve absolute maximum performance, or *supercomputing*
- Although commercial and information processing applications are increasingly becoming important drivers of the high end, historically, scientific computing has been a kind of proving ground for innovative architecture
- In the mid 60’s this included pipelined instruction processing and dynamic instruction scheduling, which are commonplace in microprocessors today

Supercomputers

- Starting in the mid 70's, supercomputing was dominated by *vector processors,* which perform operations on sequences of data elements - vectors - rather than individual scalar data
- Vector operations permit more parallelism to be obtained within a single thread of control
- Also, these vector supercomputers were implemented in very fast, expensive, high power circuit technologies
- Within the vector processing approach, the single processor performance improvement is dominated by modest improvements in cycle time and more substantial increases in the vector memory bandwidth

Supercomputers

- In the microprocessor systems, we see the combined effect of increasing clock rate, on-chip pipelined floating-point units, increasing on-chip cache size, increasing off-chip second-level cache size, and increasing use of instruction level parallelism
- The gap in uniprocessor performance is rapidly closing
- Multiprocessor architectures are adopted by both the vector processor and microprocessor designs, but the scale is quite different
- The microprocessor based supercomputers provided initially about a hundred processors, increasing to roughly a thousand from 1990 onward

Supercomputers

- These *massively parallel processors* (MPPs) have tracked the microprocessor advance, with typically a lag of one to two years behind the leading microprocessor-based workstation or personal computer
- The performance advantage of the MPP systems over traditional vector supercomputers is less substantial on more complete applications owing to the relative immaturity of the programming languages, compilers, and algorithms, however, the trend toward the MPPs is still very pronounced

Summary: Why Parallel Architecture?

- Increasingly attractive
  - Economics, technology, architecture, application demand
- Increasingly central and mainstream
- Parallelism exploited at many levels
  - Instruction-level parallelism
  - Multiprocessor servers
  - Large-scale multiprocessors (“MPPs”)
- Same story from memory system perspective
  - Increase bandwidth, reduce average latency with many local memories
- Spectrum of parallel architectures make sense
  - Different cost, performance and scalability

Taxonomy of Computer Architectures

- The idea of obtaining more performance by utilizing multiple resorce is not a new one
- In 1966 Michael Flynn introduced a **taxonomy** of computer architectures that is still the most common way of categorizing systems with parallel processing capability
- Machines are classified based on how many **data** items they can process concurrently and how many different **instructions** they can execute at the same time:
  - Single instruction, single data stream - SISD
  - Single instruction, multiple data stream - SIMD
  - Multiple instruction, single data stream - MISD
  - Multiple instruction, multiple data stream- MIMD

Single Instruction, Single Data Stream - SISD

- Single processor
- Single instruction stream
- Data stored in single memory

- A single processor executes a single instruction at a time operating on data stored in a single memory
- Uniprocessor fall into this category

- The majority of contemporary CPUs is multicore.
- A single core can be considered a SISD machine

Single Instruction, Multiple Data Stream - SIMD

- A ***single machine instruction*** controls the simultaneous execution of a number of processing elements on a lockstep basis
- Each ***processing element*** has an ***associated data memory.***
- ***Each instruction*** is executed on a ***different set of data*** by the ***different processors***
- Vector and array processors were the first SIMD machines
- GPUs follow this design at the level of Streaming multiprocessor

Multiple Instruction, Single Data Stream - MISD

- A ***sequence of data*** is transmitted to a ***set of processors***, each of which ***executes a different instruction*** sequence
- This structure is not commercially implemented
- In fact for most applications, MISD computers are rather awkward to use, but can be useful in applications of a specialised nature
- A typical example of one such specialised application is robot vision
- When fault tolerance is required in a system (military or aerospace application) data can be processed by multiple machines and decisions can be made on a majority principle

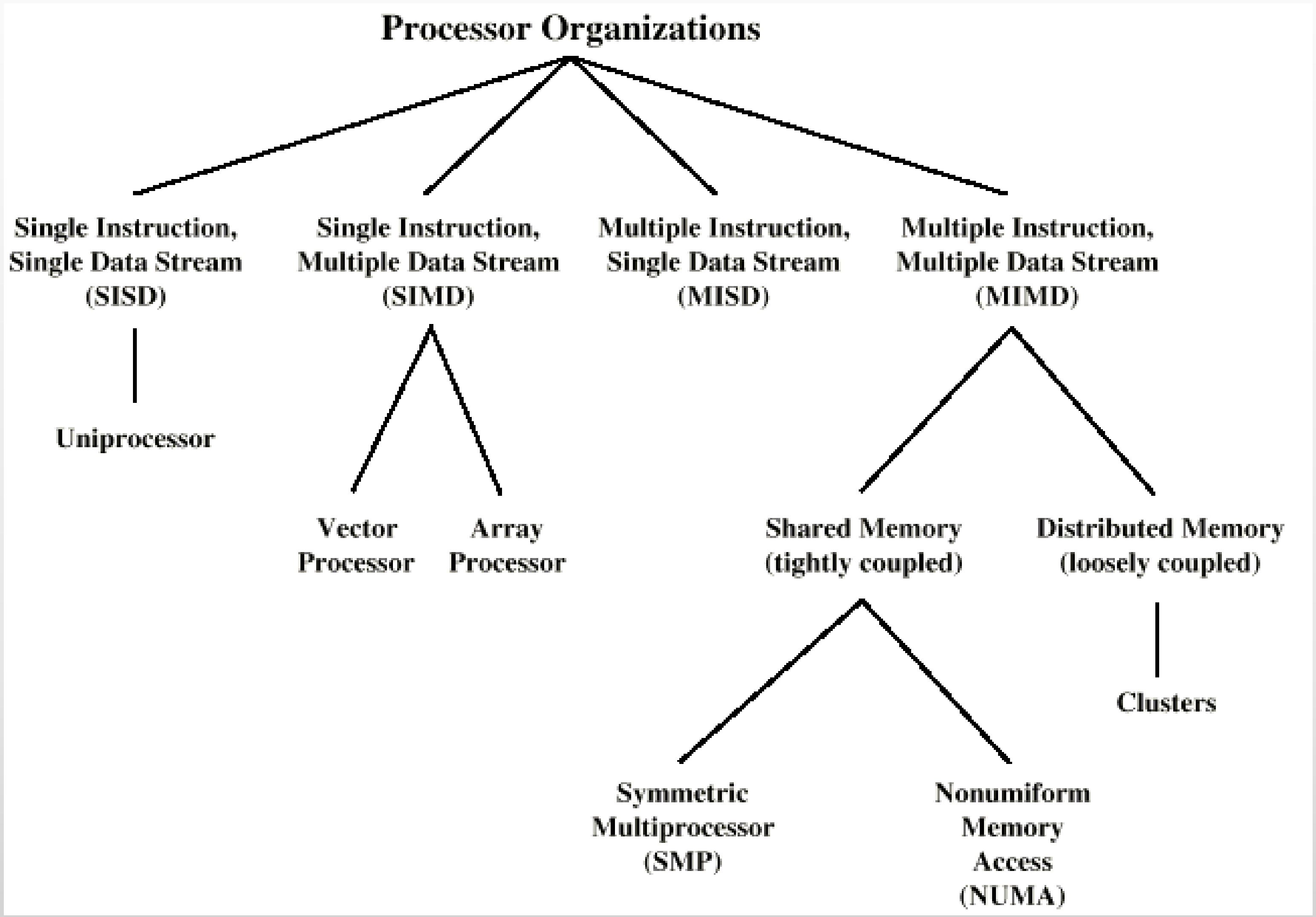
Multiple Instruction, Multiple Data Stream- MIMD

- A **set of processors** simultaneously execute different instruction sequences on different data sets

- This architecture is the most common and widely used form of parallel architectures

- General purpose processors
- Each can process all instructions necessary
- **Further classified** by method of processor communication

Taxonomy of Parallel Processor Architectures


Processor Organizations
Single Instruction, Single Data Stream (SISD)
Single Instruction, Multiple Data Stream (SIMD)
Multiple Instruction, Single Data Stream (MISD)
Multiple Instruction, Multiple Data Stream (MIMD)
Uniprocessor
Vector Processor
Array Processor
Shared Memory (tightly coupled)
Distributed Memory (loosely coupled)
Clusters
Symmetric Multiprocessor (SMP)
Nonumiform Memory Access (NUMA)

Speedup

- A key reference point for both the architect and the application developer is how the use of parallelism improves the performance of the application
- We may define the *speedup* on processors as

$$\text{Speedup (p processors)} = \frac{\textit{Performance (p processors)}}{\textit{Performance (1 processor)}}$$

- For a fixed problem size (input data set), performance = 1/time

- Speedup fixed problem (p processors) = $\dfrac{\textit{Time (1 processor)}}{\textit{Time (p processors)}}$

Communication Performance

- Performance characteristics determine usage of operations at a layer
  - Programmer, compilers etc make choices based on this
- Fundamentally, three characteristics:
  - *Latency*: time taken for an operation
  - *Bandwidth*: rate of performing operations
  - *Cost*: impact on execution time of program
- If processor does one thing at a time:
  - bandwidth (operation per second) is $\propto$ 1/latency
  - cost is simply the latency times the number of operations
- But actually it is more complex in modern systems

Communication Performance

- Modern computer systems do many different operations at once and the relationship between these performance metrics is much more complex
- Characteristics apply to overall operations, as well as individual components of a system

- Since the unique property of parallel computer architecture is **communication**, the operations that we are concerned with most often are **data transfers**

Linear Model of Data Transfer Latency

- The time for a **data transfer operation** is generally described by a linear model:
  - *Transfer time (n)* $= T_0 + n/B$
  - n is the amount of data (e.g. number of bytes),
  - B is the transfer rate of the component moving the data (e.g. bytes per second),
  - the constant term T_0 is the start-up cost

- This is a very convenient model, and it is used to describe a diverse collection of operations: messages, memory accesses, bus transactions, and vector operations

Linear Model of Data Transfer Latency

- It applies in many aspects of traditional computer architecture, as well
  - For memory operations, it is essentially the access time
  - For bus transactions, it reflects the bus arbitration and command phases
  - For any sort of pipelined operation, including pipelined instruction processing or vector operations, it is the time to fill pipeline

Linear Model of Data Transfer Latency

- But linear model not enough:
  - It does not give any indication when the *next such operation can be initiated*
  - It does not indicate whether other *useful work can be performed during the transfer*
  - These other factors depend on how the transfer is performed: *need to know how transfer is performed*

Communication Cost Model

- The data transfer in which we are most interested is the one that occurs **across the network** in parallel machines
- It is initiated by the processor through the communication assist
- The essential components of this operation can be described by the following simple model

Communication Time (n)= Overhead + Network Delay + Occupancy

Communication Cost Model

Communication Time (n)= Overhead + Network Delay + Occupancy

- The ***Overhead*** is the time the processor spends initiating the transfer
- This may be a fixed cost, if the processor simply has to tell the communication assist to start, or it may be linear in *n*, if the processor has to copy the data into the assist
- The key point is that this is time the processor is busy with the communication event; it cannot do other useful work or initiate other communication during this time

Communication Cost Model

Communication Time (n)= Overhead + Network Delay + Occupancy

- The remaining portions of the communication time is considered the ***network latency***; it is the part that can be hidden by other processor operations
- The ***Occupancy*** is the time it takes for the data to pass through the slowest component on the communication path

Communication Cost Model

Communication Time (n)= Overhead + Network Delay + Occupancy

- ***Occupancy*** :
  - The data will occupy other resources, including buffers, switches, and the communication assist
  - Often the communication assist is the bottleneck that determines the occupancy
  - The occupancy limits how frequently communication operations can be initiated.
  - The next data transfer will have to wait until the critical resource is no longer occupied before it can use that same resource

Communication Cost Model

Communication Time (n)= Overhead + Network Delay + Occupancy

- The remaining communication time is lumped into the ***Network Delay***, which includes the time for a bit to be routed across the actual network and many other factors, such as the time to get through the communication assist
  - From the processors viewpoint, the specific hardware components contributing to network delay are indistinguishable
  - What effects the processor is: - how long it must wait before it can use the result of a communication event - how much of this time it can be bust with other activities - and how frequently it can communicate data

Communication Cost Model

- A useful model connecting the program characteristics to the hardware performance is given by

  Communication Cost = frequency * (Comm time - overlap)

- The *frequency of communication*:
  - is defined as the number of communication operations per unit of work in the program
  - it depends on many programming factors and many hardware design factors

Communication Cost Model

Communication Cost = frequency * (Comm time - overlap)

- The *frequency of communication*:
  - Hardware may limit the transfer size and thereby determine the minimum number of messages. It may automatically replicate data or migrate it to where it is used
  - There is a certain amount of communication that is inherent to parallel execution, since data must be shared and processors must coordinate their work
  - A machine can support programs with a high communication frequency if the other parts of the communication cost are small: low overhead, low network delay, and small occupancy