Cap 1
Introduction

Introduction

What is Parallel Architecture?

Why Parallel Architecture?

Evolution and Convergence of Parallel Architectures

Fundamental Design Issues

What is Parallel Architecture?

A parallel computer is a collection of processing elements that cooperate to solve large problems fast

Some broad issues:

- Resource Allocation:
  - how large a collection?
  - how powerful are the elements?
  - how much memory?
- Data access, Communication and Synchronization
  - how do the elements cooperate and communicate?
  - how are data transmitted between processors?
  - what are the abstractions and primitives for cooperation?
- Performance and Scalability
  - how does it all translate into performance?
  - how does it scale? (satura o crescimento ou não)

1.1 Why Study Parallel Architecture?

Role of a computer architect:

> To design and engineer the various levels of a computer system to maximize *performance* and *programmability* within limits of *technology* and *cost*.

Parallelism:

- Provides alternative to faster clock for performance (limitações de tecnologia)
- Applies at all levels of system design (ênfase do livro: pipeline, cache, comunicação, sincronização)
- Is a fascinating perspective from which to view architecture
- Is increasingly central in information processing

Why Study it Today?

History: diverse and innovative organizational structures, often tied to novel programming models

Rapidly maturing under strong technological constraints

- The "killer micro" is ubiquitous
- Laptops and supercomputers are fundamentally similar!
- Technological trends cause diverse approaches to converge

Technological trends make parallel computing inevitable

- In the mainstream

Need to understand fundamental principles and design tradeoffs, not just taxonomies

- Naming, Ordering, Replication (de dados), Communication performance

Inevitability of Parallel Computing

Application demands: Our insatiable need for computing cycles

- *Scientific computing*: CFD (Computational Fluid Dynamics), Biology, Chemistry, Physics, ...
- *General-purpose computing*: Video, Graphics, CAD, Databases, TP...

Technology Trends

- Number of transistors on chip growing rapidly
- Clock rates expected to go up only slowly

Architecture Trends

- Instruction-level parallelism valuable but limited
- Coarser-level parallelism, as in MPs, the most viable approach

Economics

Current trends:

- Today's microprocessors have multiprocessor support
- Servers and workstations becoming MP: Sun, SGI, DEC, COMPAQ!...
- Tomorrow's microprocessors are multiprocessors (hoje multicore)

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

1.1.1 Application Trends

Demand for cycles fuels advances in hardware, and vice-versa

- Cycle drives exponential increase in microprocessor performance
- Drives parallel architecture harder: most demanding applications

Range of performance demands

- Need range of system performance with progressively increasing cost
- Platform pyramid

Goal of applications in using parallel machines: Speedup

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

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

$$Speedup_{\ fixed\ problem}\ (p\ processors) = \frac{Time\ (1\ processor)}{Time\ (p\ processors)}$$

Scientific Computing Demand

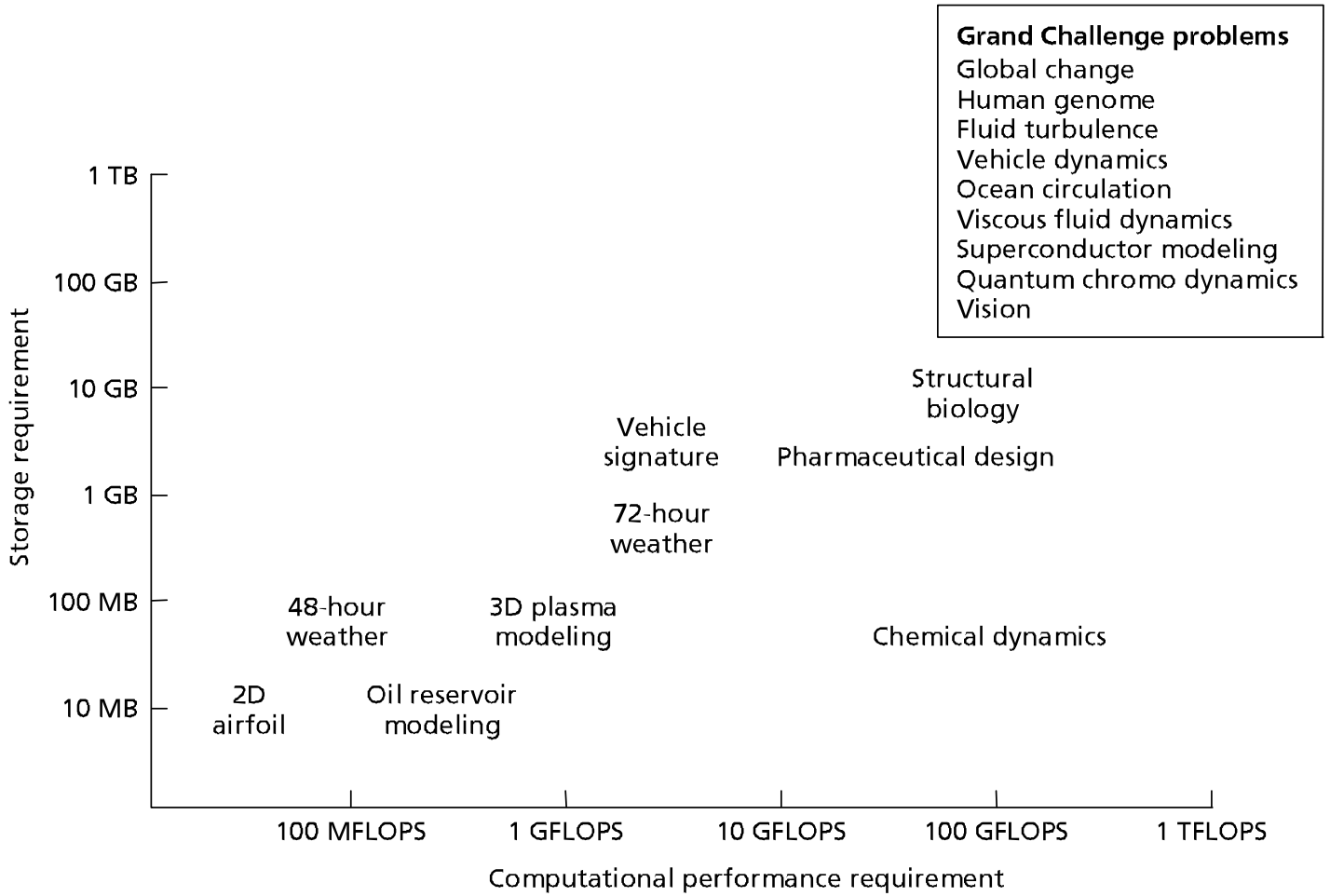


Fig 1.2, pag 7: dados de 1993

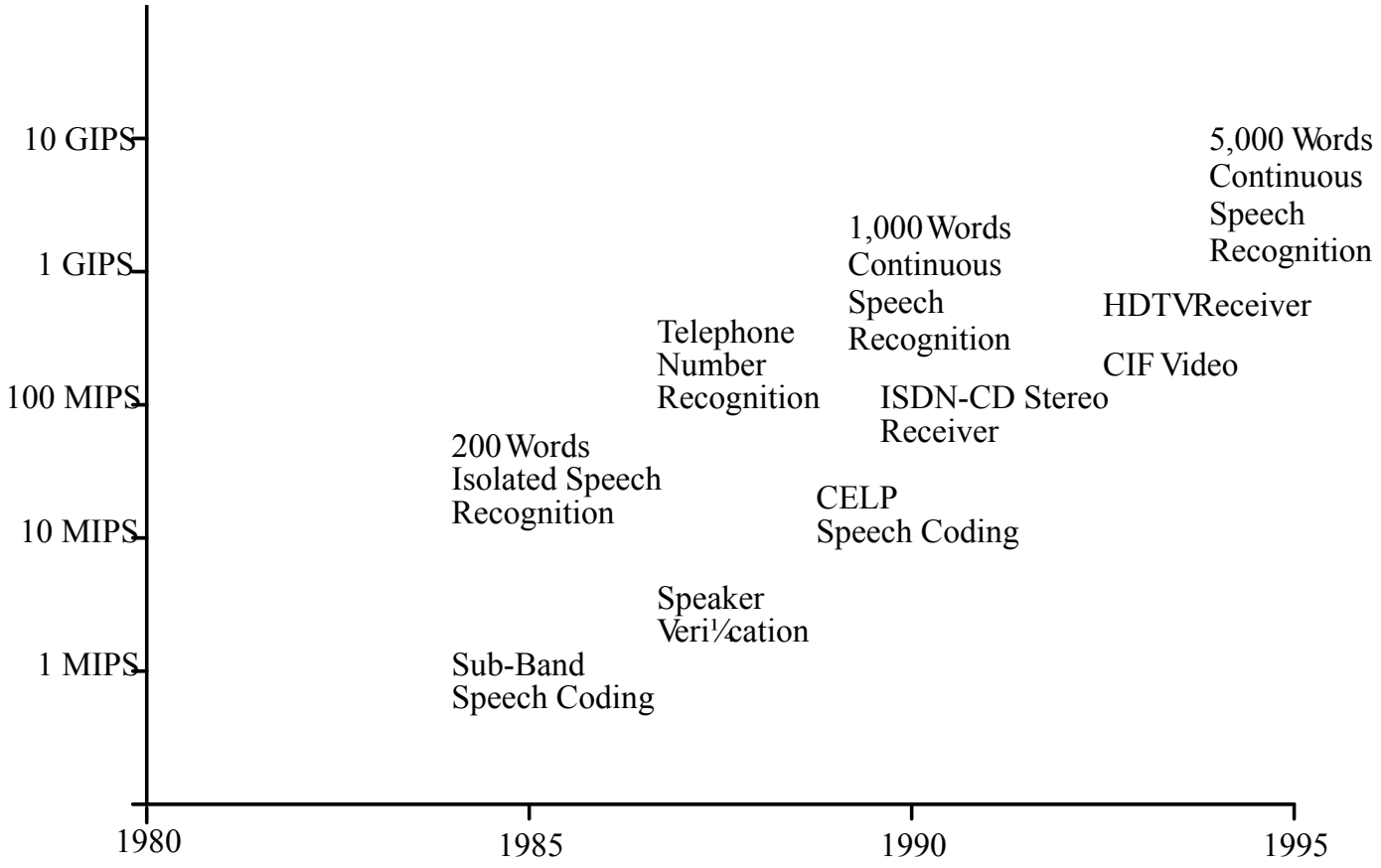
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Engineering Computing Demand

Large parallel machines a mainstay in many industries

- Petroleum (reservoir analysis)
- Automotive (crash simulation, drag analysis, combustion efficiency),
- Aeronautics (airflow analysis, engine efficiency, structural mechanics, electromagnetism),
- Computer-aided design
- Pharmaceuticals (molecular modeling)
- Visualization
  - in all of the above
  - entertainment (films like Toy Story, centenas de SUNs)
  - architecture (walk-throughs and rendering)
- Financial modeling (yield and derivative analysis)
- etc.

Applications: Speech and Image Processing

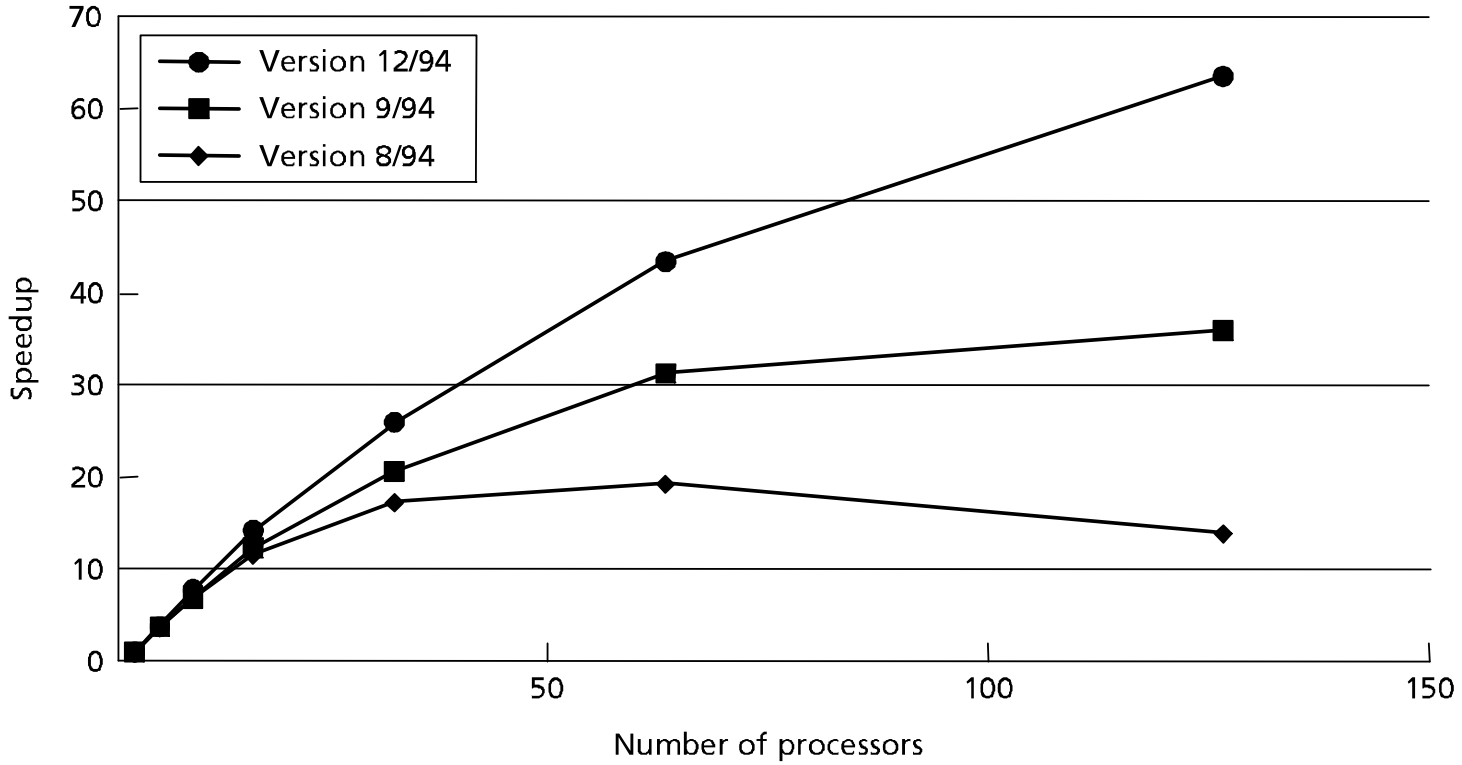


- Also CAD, Databases, . . .
- *100 processors (hoje) gets you 10 years, 1000 (hoje) gets you 20* !

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Learning Curve for Parallel Applications

Intel
Paragon
128 proc



- AMBER (Assisted Model Building through Energy Refinement) molecular dynamics simulation program
- Starting point was vector code for Cray-1 (equiv. à versão 8; versões 9 e 12 incorporaram otimizações no código visando a paralelização)
- 145 MFLOP on Cray90, 406 MFLOP for final version on 128-processor Paragon, 891 MFLOP on 128-processor Cray T3D

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Commercial Computing

Also relies on parallelism for high end

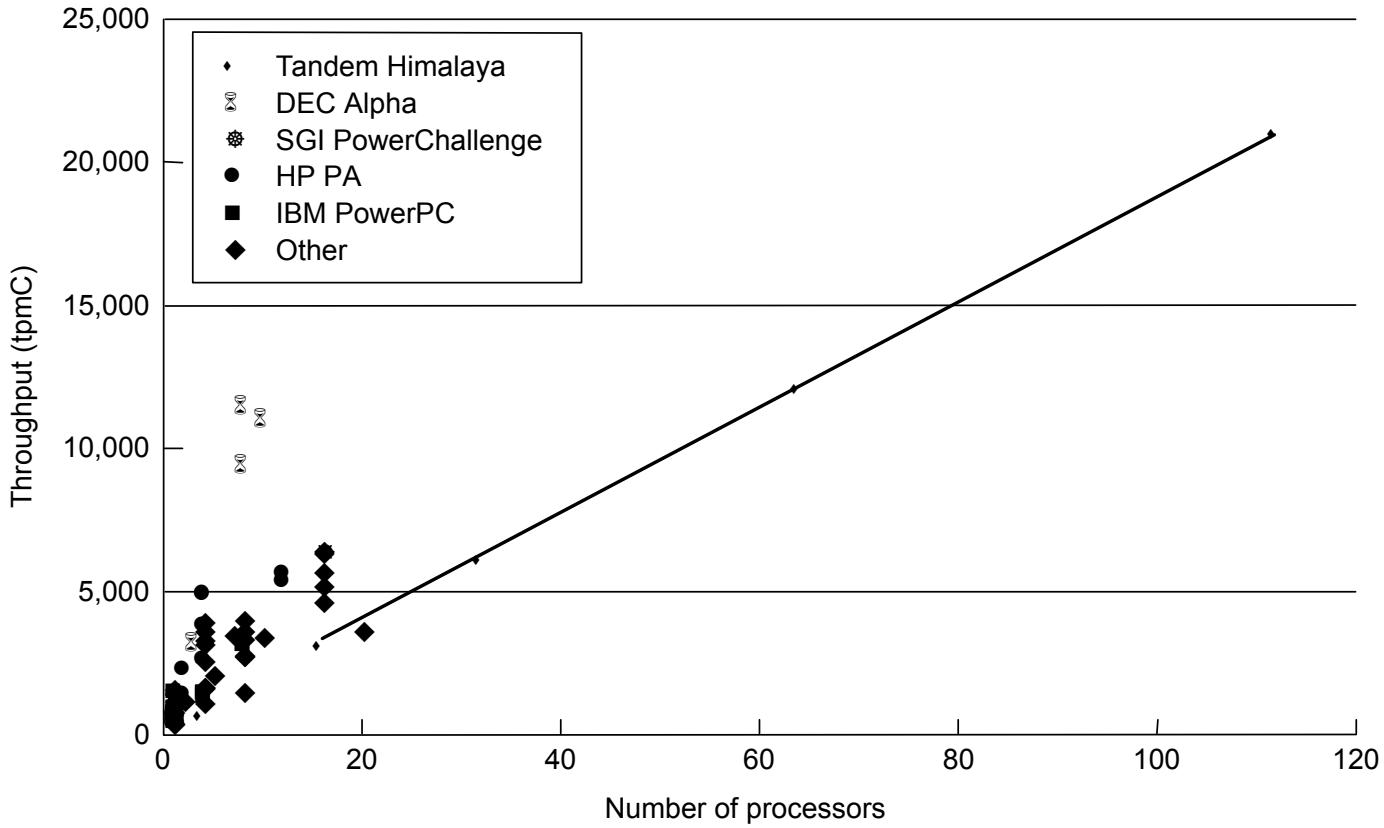
- Scale not so large, but use much more wide-spread
- Computational power determines scale of business that can be handled

Databases, online-transaction processing, decision support, data mining, data warehousing ...

TPC (Transaction Processing Performance Council) benchmarks (TPC-C order entry, TPC-D decision support)

- Explicit scaling criteria provided
- Size of enterprise scales with size of system
- Problem size no longer fixed as *p* increases, so throughput is used as a performance measure (transactions per minute or *tpm*)

TPC-C Results for March 1996



- Parallelism is pervasive
- Small to moderate scale parallelism very important
- Difficult to obtain snapshot to compare across vendor platforms (sistemas mostrados lançados em datas diferentes)
  - Ex: Tandem é mais velho que PowerPC e DEC Alpha

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Summary of Application Trends

Transition to parallel computing has occurred for scientific and engineering computing

In rapid progress in commercial computing

- Database and transactions as well as financial
- Usually smaller-scale, but large-scale systems also used

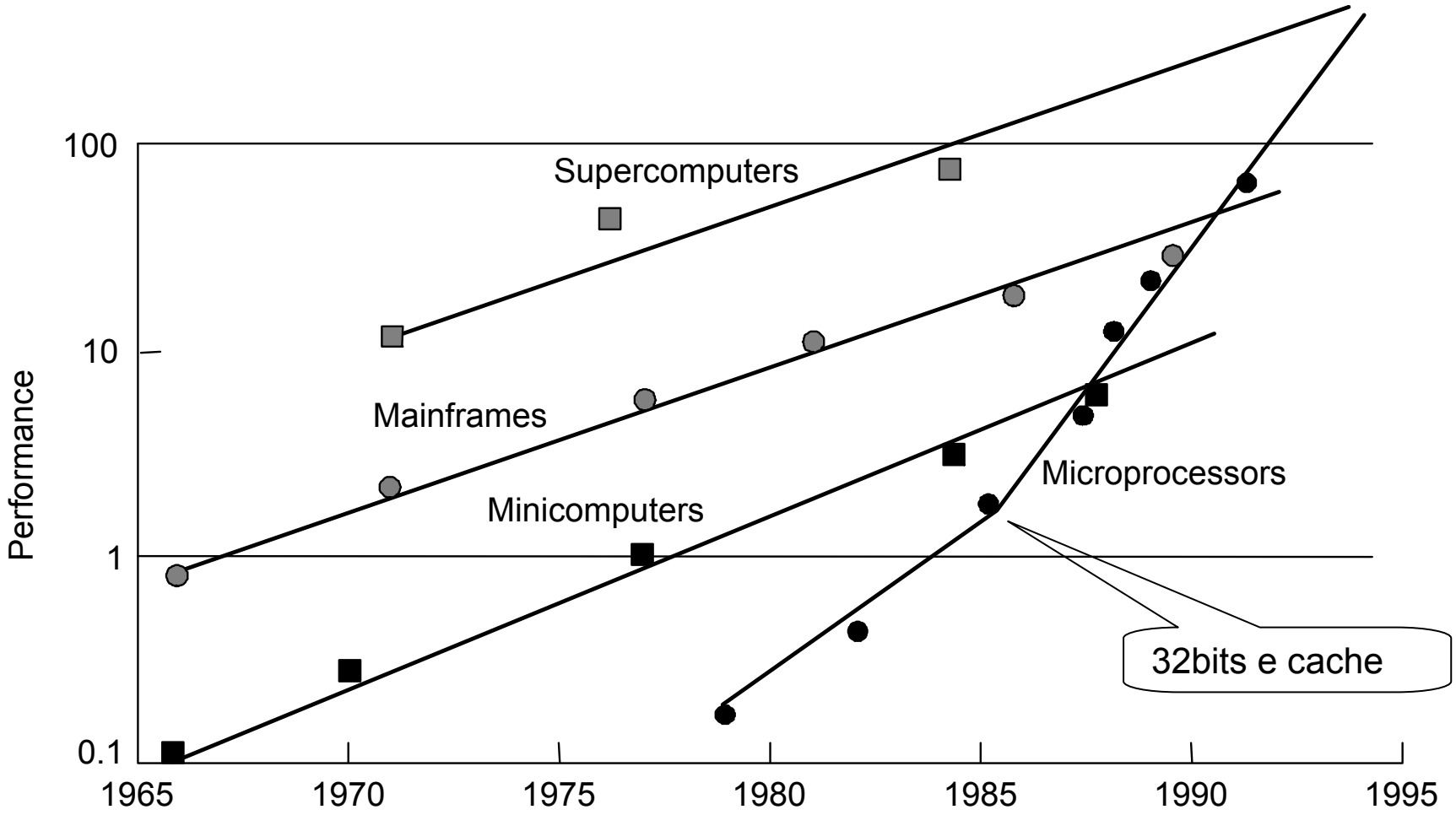
Desktop also uses multithreaded programs, which are a lot like parallel programs

Demand for improving throughput on sequential workloads

- Greatest use of small-scale multiprocessors

Solid application demand exists and will increase

1.1.2 Technology Trends



The natural building block for multiprocessors is now also about the fastest!

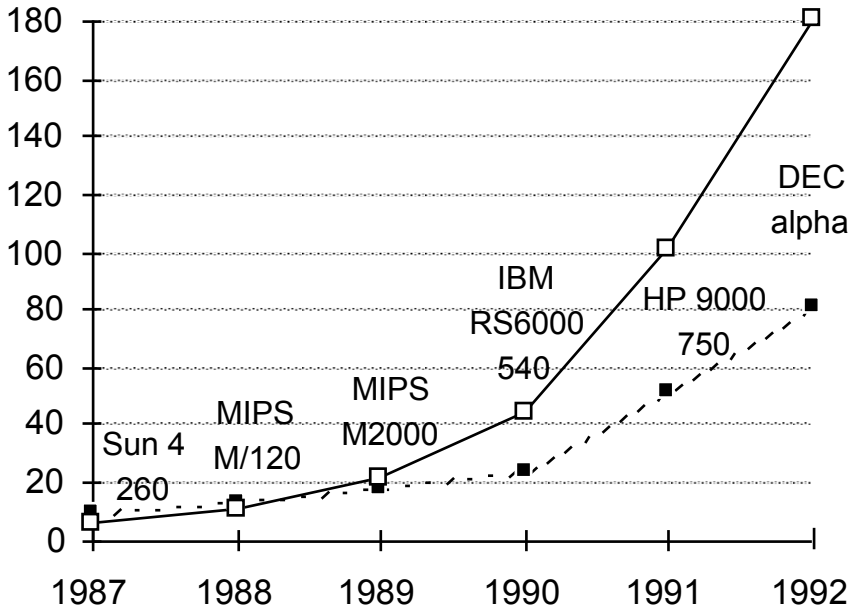
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

pag 15 e 4-5

General Technology Trends

- *Microprocessor performance* increases 50% - 100% per year
- *Transistor count* doubles every 3 years
- *DRAM size* quadruples every 3 years
- Huge investment per generation is carried by huge commodity market

SPEC
Linpack



DRAM de ´80 a ´95

Tamanho 1000x

Tempo de acesso 2x

- Not that single-processor performance is plateauing, but that parallelism is a natural way to improve it.

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Technology: A Closer Look

Basic advance is *decreasing feature size* (λ)

- Circuits become either faster or lower in power

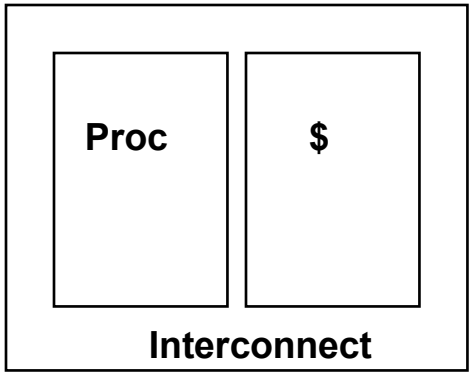
Die size is growing too

- Clock rate improves roughly proportional to improvement in λ
- Number of transistors improves like λ^2 (or faster)

Performance > 100x per decade; clock rate 10x, rest transistor count

How to use more transistors?

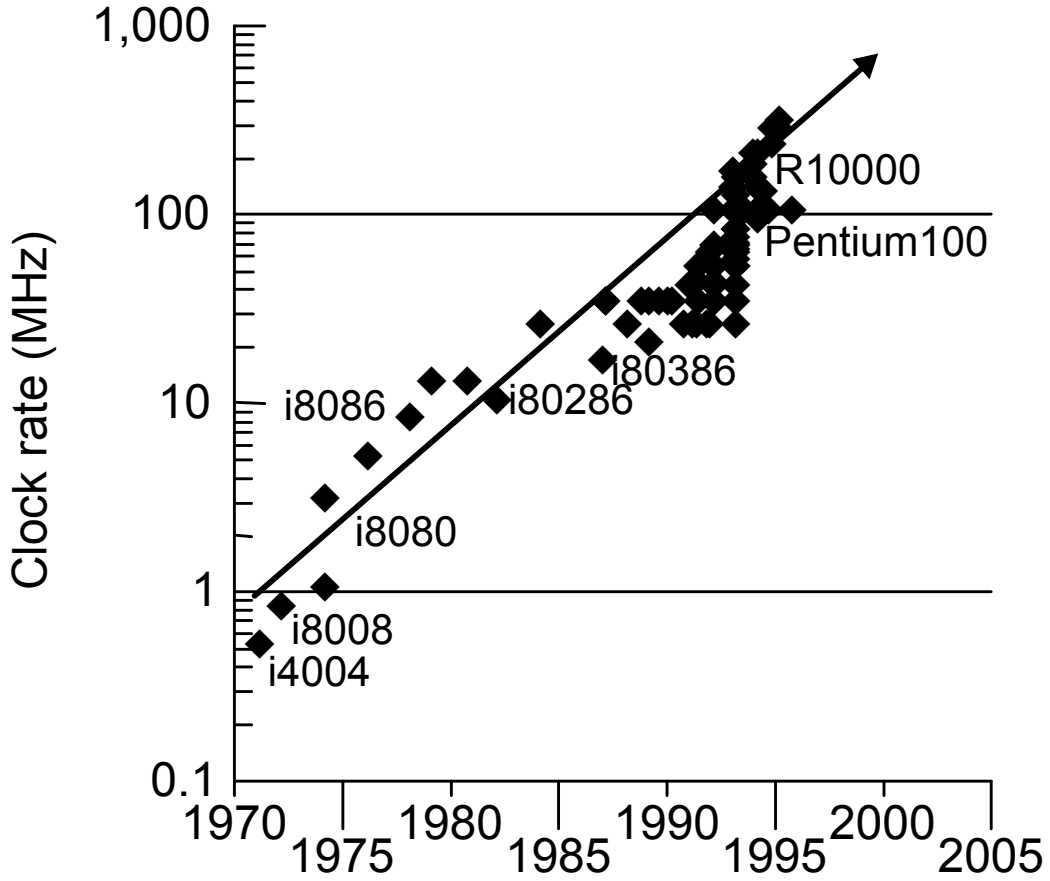
- Parallelism in processing
  - multiple operations per cycle reduces CPI
- Locality in data access
  - avoids latency and reduces CPI
  - also improves processor utilization
- Both need resources, so tradeoff



Fundamental issue is resource distribution, as in uniprocessors

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

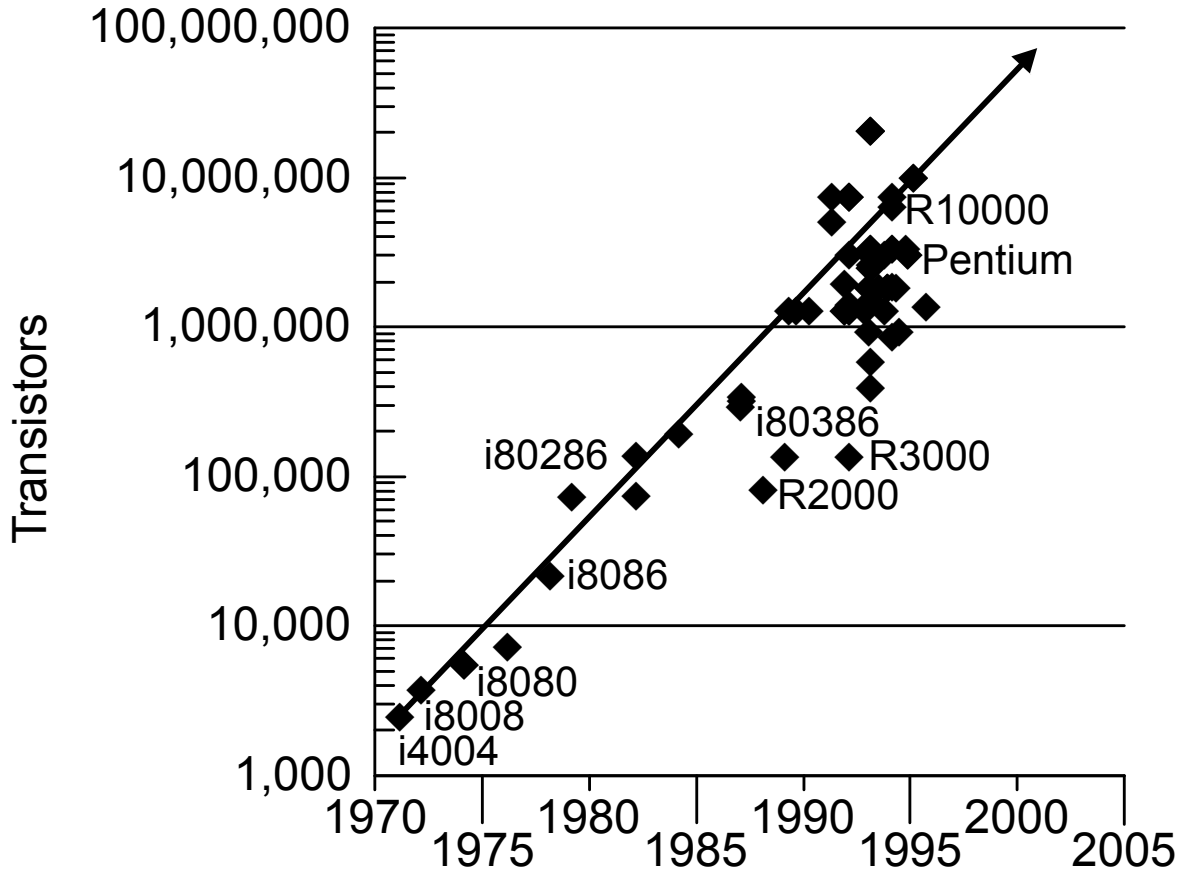
Clock Frequency Growth Rate



- 30% per year

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Transistor Count Growth Rate



- 100 million transistors on chip by early 2000's A.D. (2006 > 1 bilhão - Montecito)
- Transistor count grows much faster than clock rate
  - 40% per year, order of magnitude more contribution in 2 decades

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Similar Story for Storage

Divergence between memory capacity and speed more pronounced

- Capacity increased by 1000x from 1980-95, speed only 2x
- Gigabit DRAM by c. 2000, but gap with processor speed much greater

Larger memories are slower, while processors get faster

- Need to transfer more data in parallel
- Need deeper cache hierarchies
- How to organize caches?

Parallelism increases effective size of each level of hierarchy, without increasing access time

Parallelism and locality within memory systems too

- New designs fetch many bits within memory chip; follow with fast pipelined transfer across narrower interface
- Buffer caches most recently accessed data

Disks too: Parallel disks plus caching

1.1.3 Architectural Trends

Architecture translates technology's gifts to performance and capability

Resolves the tradeoff between parallelism and locality

- Current microprocessor: 1/3 compute, 1/3 cache, 1/3 off-chip connect
- Tradeoffs may change with scale and technology advances

Understanding microprocessor architectural trends

- Helps build intuition about design issues or parallel machines
- Shows fundamental role of parallelism even in "sequential" computers

Four generations of architectural history: tube, transistor, IC, VLSI

- Here focus only on VLSI generation

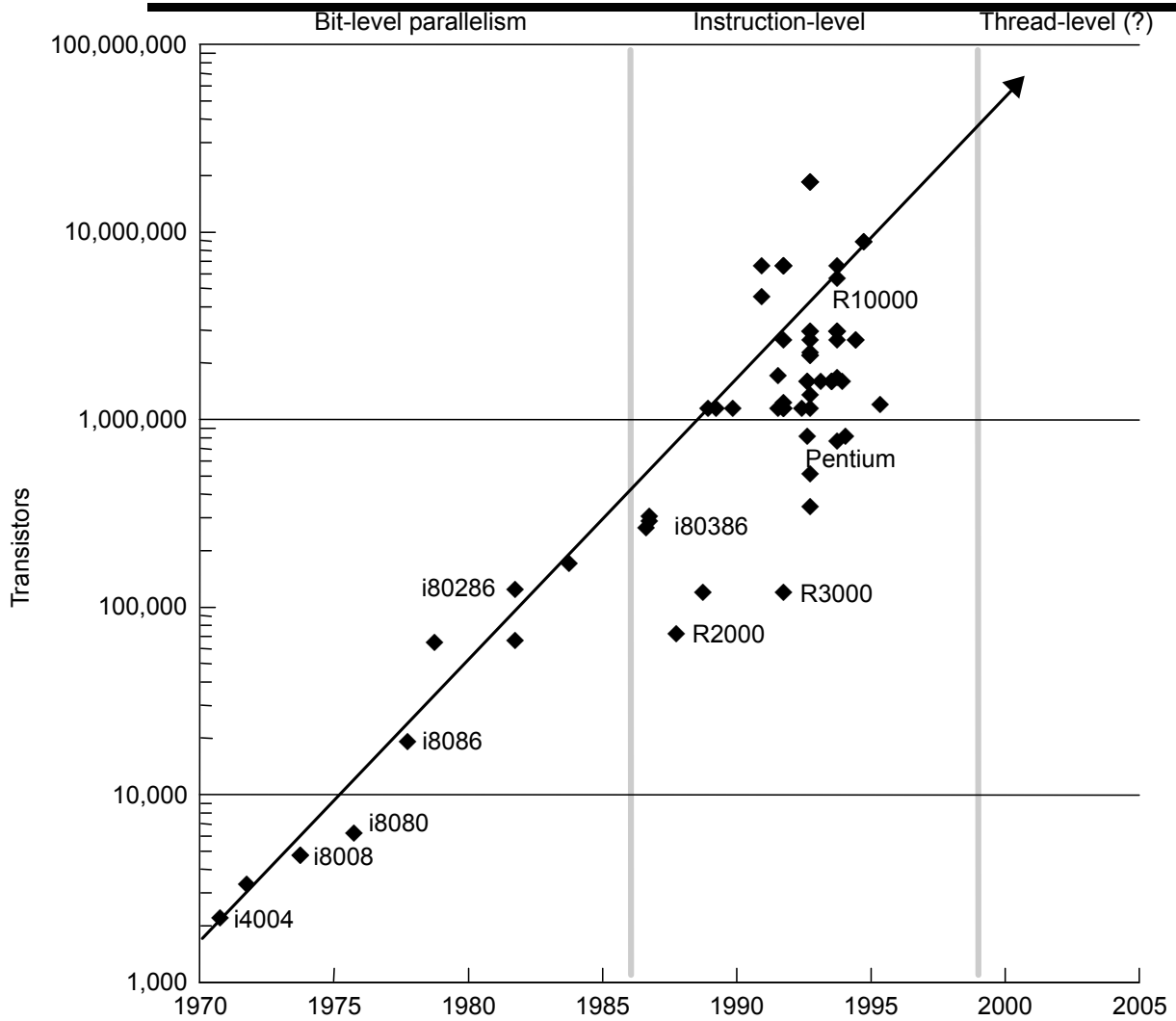
Greatest delineation in VLSI has been in type of parallelism exploited

Architectural Trends

Greatest trend in VLSI generation is increase in parallelism

- Up to 1985: bit level parallelism: 4-bit -> 8 bit -> 16-bit
  - slows after 32 bit
  - adoption of 64-bit now under way, 128-bit far (not performance issue)
  - great inflection point when 32-bit micro and cache fit on a chip (ver fig 1.1)
- Mid 80s to mid 90s: instruction level parallelism
  - pipelining and simple instruction sets, + compiler advances (RISC)
  - on-chip caches and functional units => superscalar execution
  - greater sophistication: out of order execution, speculation, prediction
    - to deal with control transfer and latency problems
- Next step: thread level parallelism

Phases in VLSI Generation



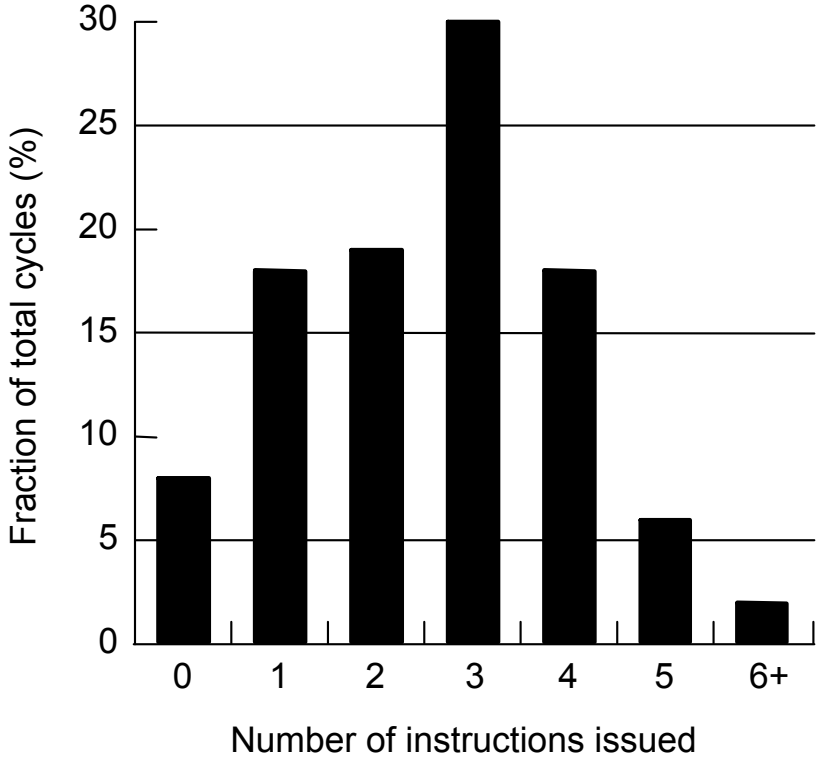
- How good is instruction-level parallelism?
- Thread-level needed in microprocessors?

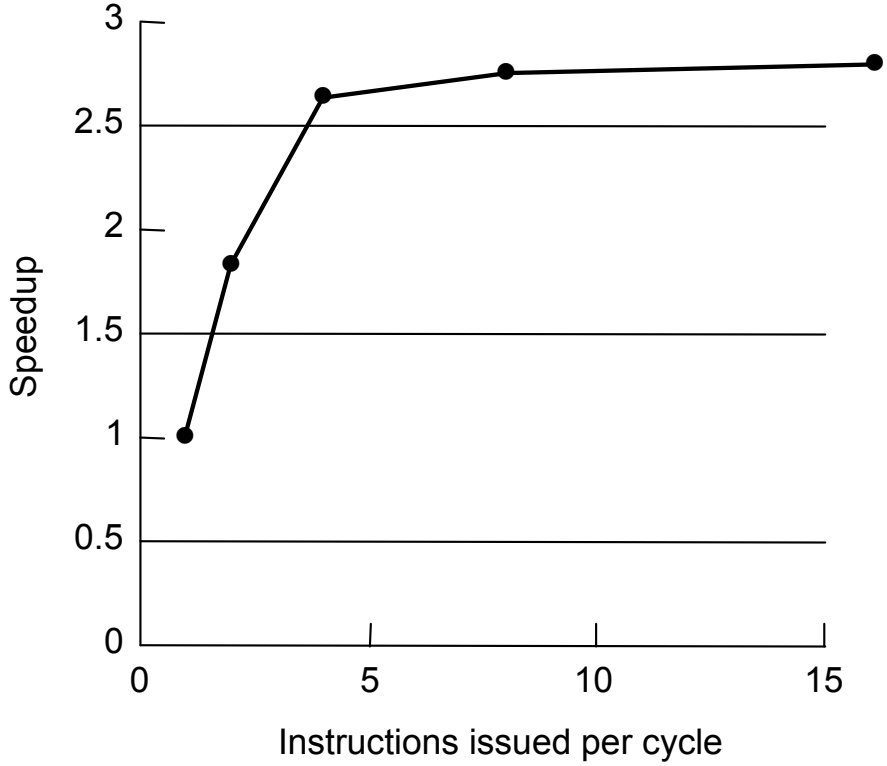
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Architectural Trends: ILP

- Reported speedups for superscalar processors
  - Horst, Harris, and Jardine [1990] 1.37
  - Wang and Wu [1988] 1.70
  - Smith, Johnson, and Horowitz [1989] 2.30
  - Murakami et al. [1989] 2.55
  - Chang et al. [1991] 2.90
  - Jouppi and Wall [1989] 3.20
  - Lee, Kwok, and Briggs [1991] 3.50
  - Wall [1991] 5
  - Melvin and Patt [1991] 8
  - Butler et al. [1991] ..... 17+
- Large variance due to difference in
  - application domain investigated (numerical versus non-numerical)
  - capabilities of processor modeled

ILP Ideal Potential



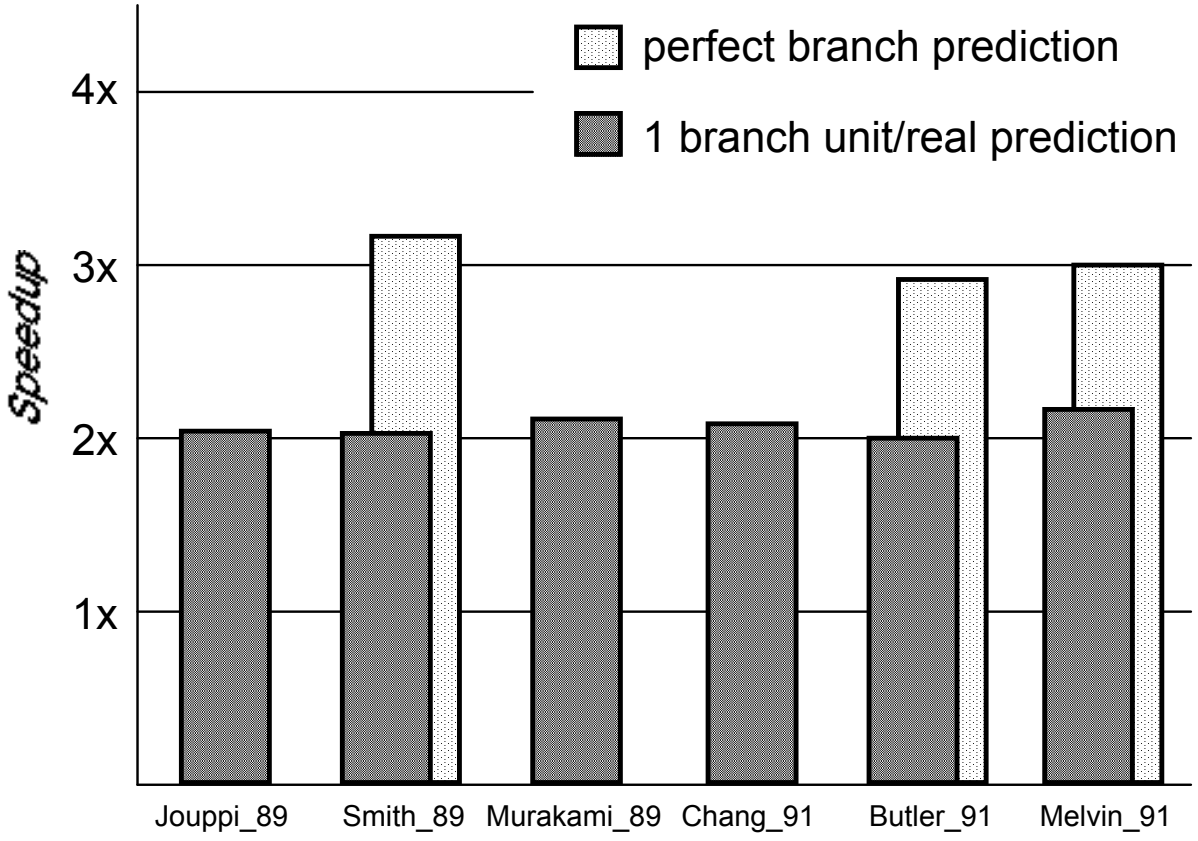


- Infinite resources and fetch bandwidth, perfect branch prediction and renaming
  - real caches and non-zero miss latencies
- Recursos ilimitados; única restrição é dependência de dados

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Results of ILP Studies

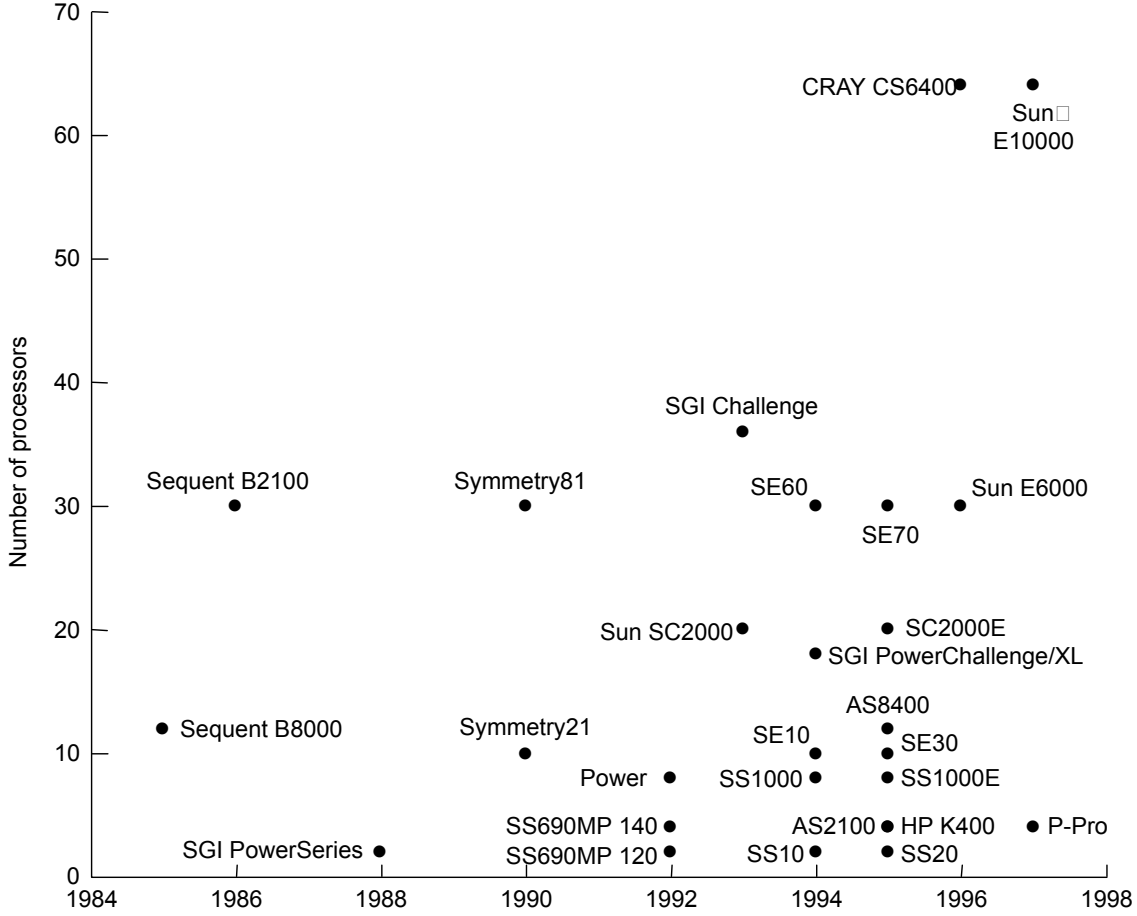
- Concentrate on parallelism for 4-issue machines



- Realistic studies show only 2-fold speedup
  - Recent studies show that more ILP needs to look across threads

Architectural Trends: Bus-based MPs

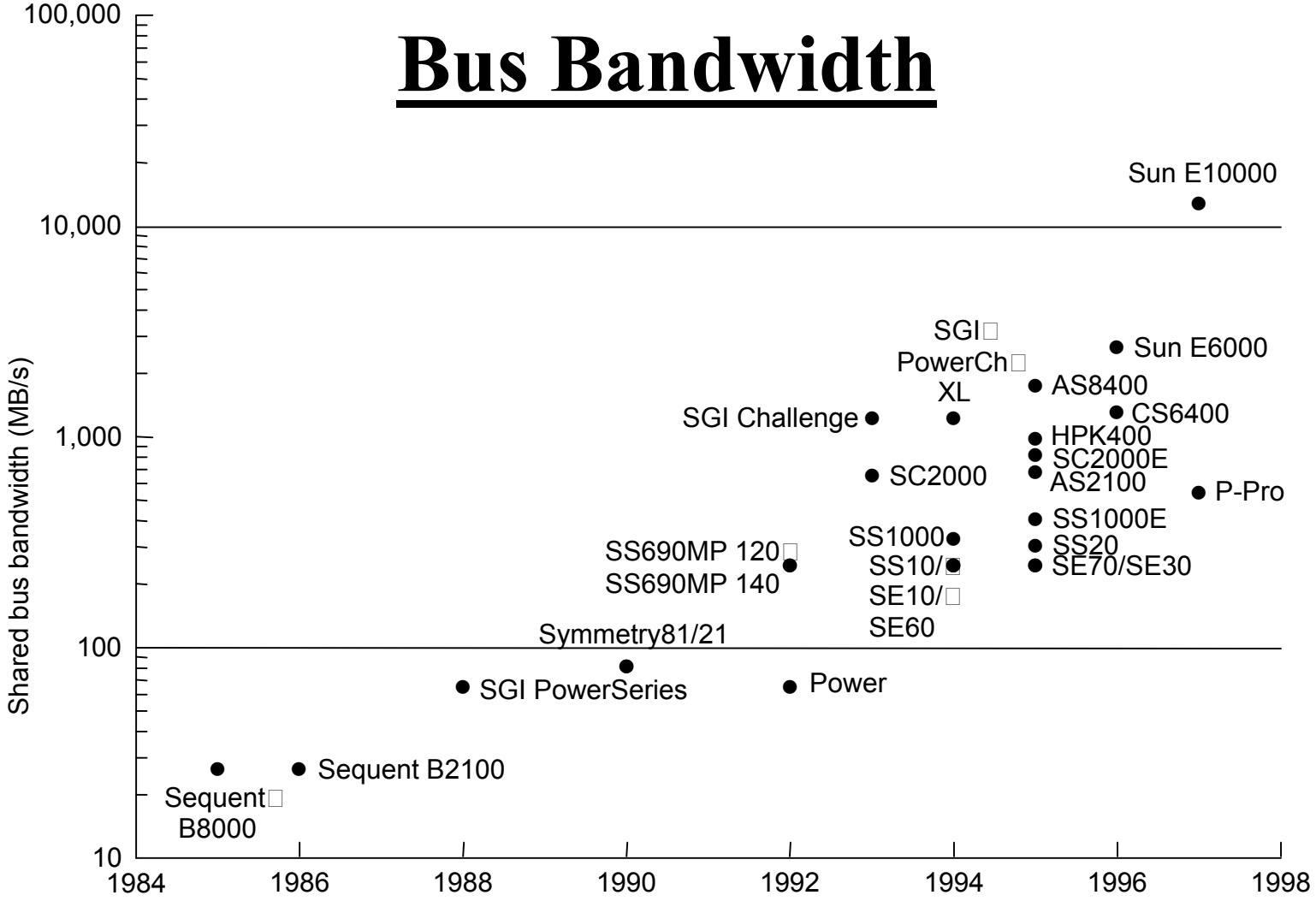
- Micro on a chip makes it natural to connect many to shared memory
  - dominates server and enterprise market, moving down to desktop
- Faster processors began to saturate bus, then bus technology advanced
  - today, range of sizes for bus-based systems, desktop to large servers



No. of processors in fully configured commercial shared-memory systems

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Bus Bandwidth



• muitos processadores já vêm com suporte para multiprocessador (Pentium Pro: ligar 4 processadores a um barramento único sem glue logic)

• multiprocessamento de pequena escala tornou-se commodity

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Economics

Commodity microprocessors not only fast but CHEAP

- Development cost is tens of millions of dollars (5-100 typical)
- BUT, many more are sold compared to supercomputers
- Crucial to take advantage of the investment, and use the commodity building block
- Exotic parallel architectures no more than special-purpose

Multiprocessors being pushed by software vendors (e.g. database) as well as hardware vendors

Standardization by Intel makes small, bus-based SMPs commodity

Desktop: few smaller processors versus one larger one?

- Multiprocessor on a chip

1.1.4 Scientific Supercomputing

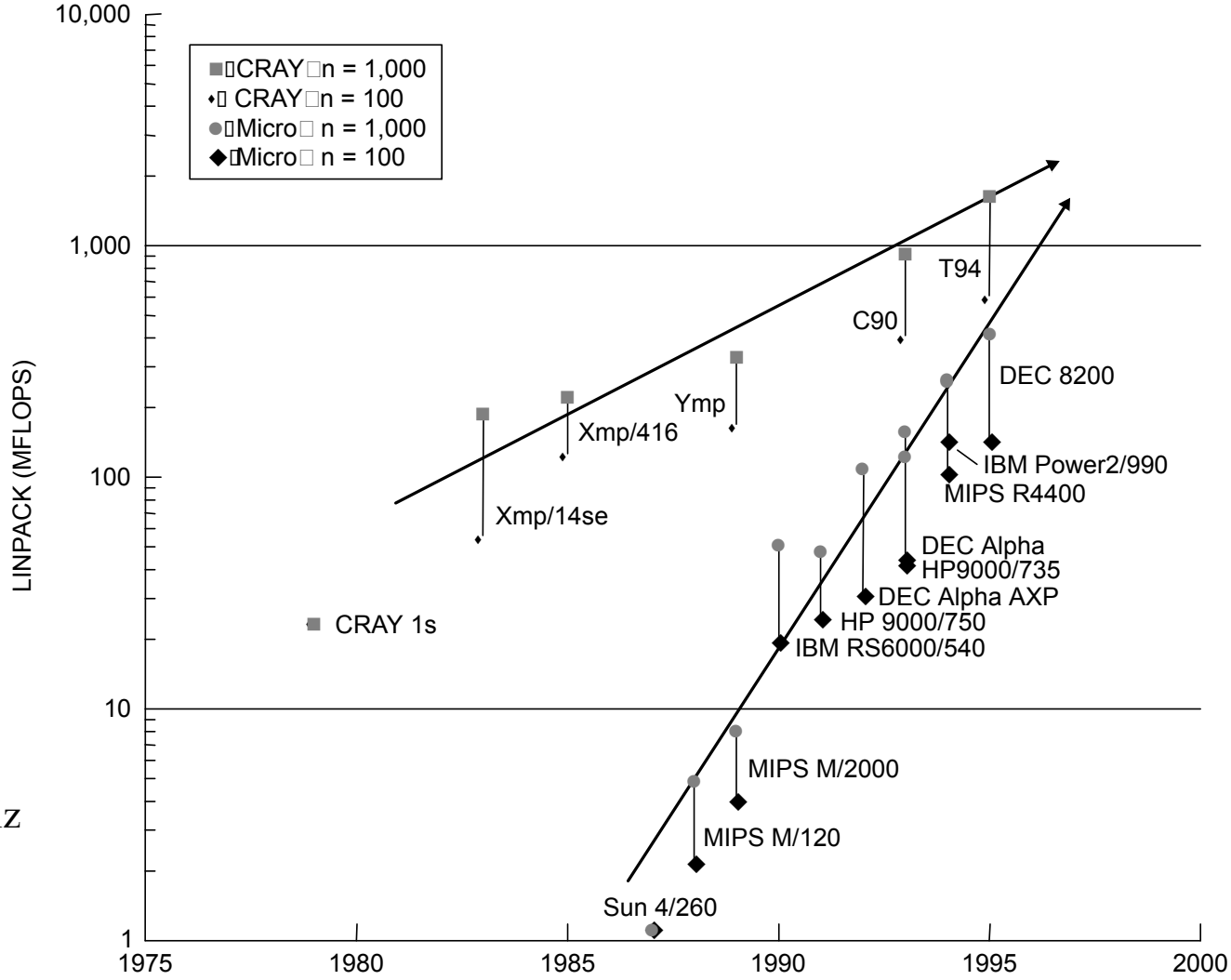
Proving ground and driver for innovative architecture and techniques

- Market smaller relative to commercial as MPs become mainstream
- Dominated by vector machines starting in 70s
- Microprocessors have made huge gains in floating-point performance
  - high clock rates
  - pipelined floating point units (e.g., multiply-add every cycle)
  - instruction-level parallelism
  - effective use of caches (e.g., automatic blocking)
- Plus economics

Large-scale multiprocessors replace vector supercomputers

- Well under way already

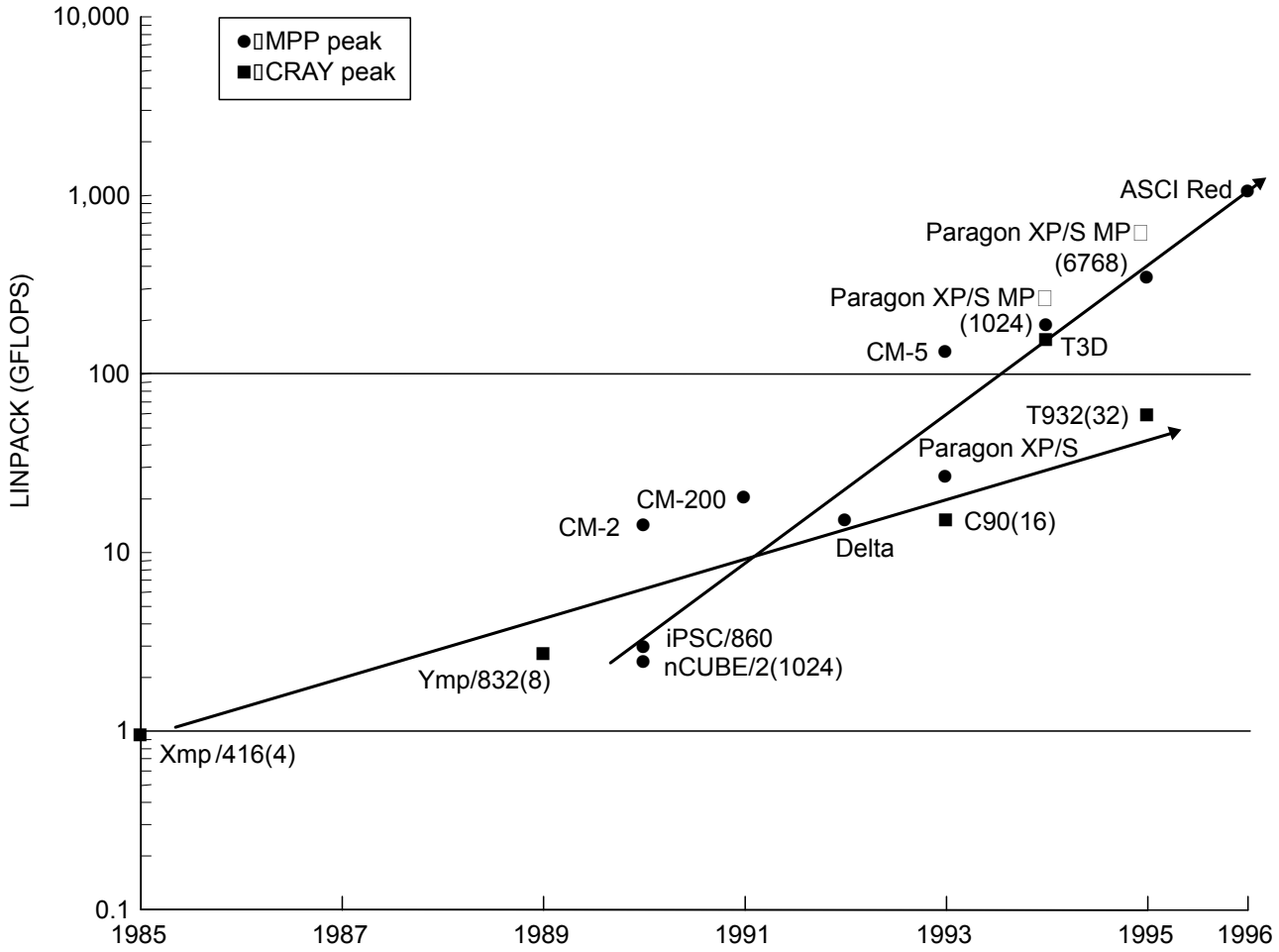
Raw Uniprocessor Performance: LINPACK



2 pontos: matriz
100 x 100 e
1000 x 1000

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

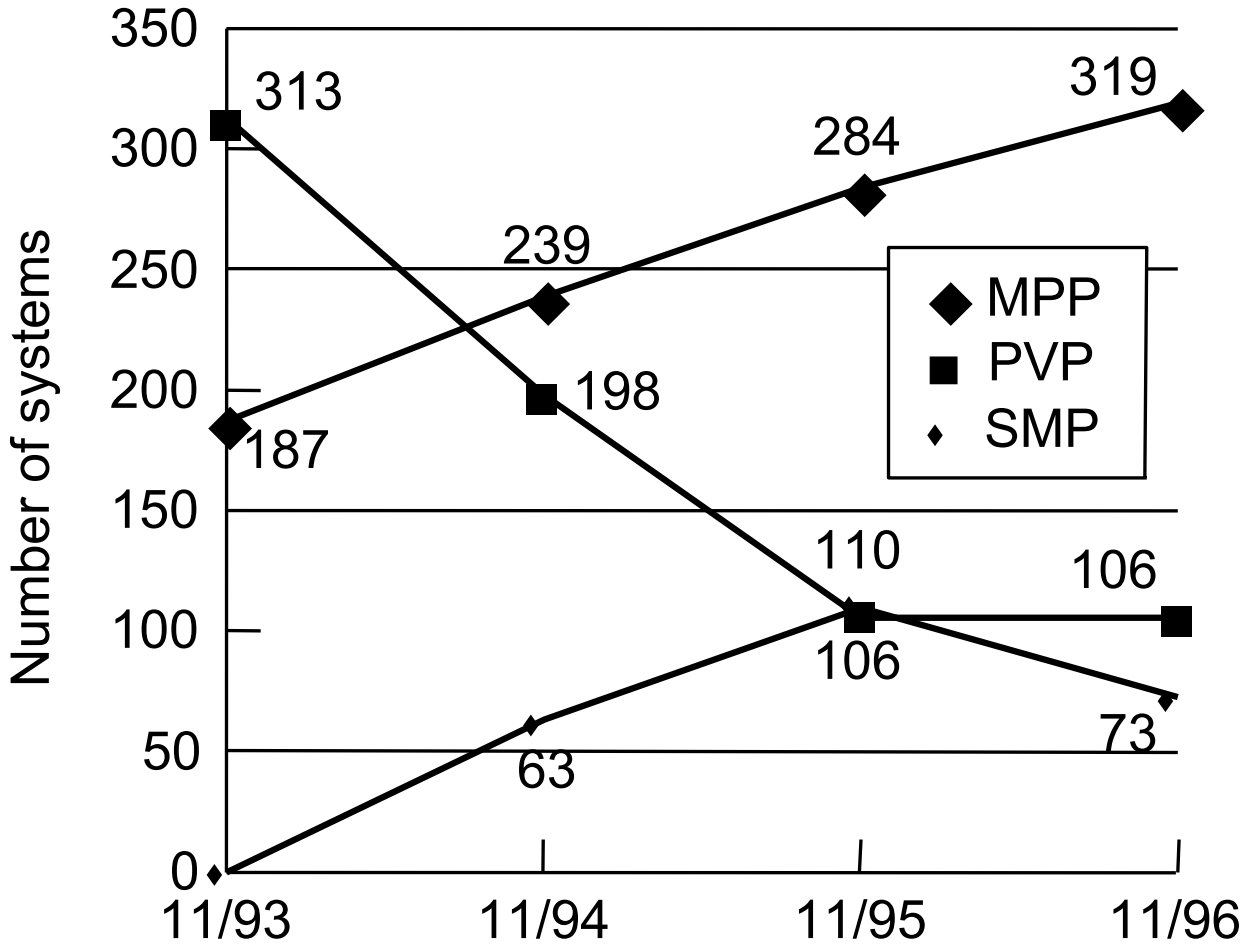
Raw Parallel Performance: LINPACK



- Even vector Crays became parallel: X-MP (2-4) Y-MP (8), C-90 (16), T94 (32)
- Since 1993, Cray produces MPPs (Massively Parallel Processors) too (T3D, T3E)

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

500 Fastest Computers



MPP: Massively Parallel Processors
PVP: Parallel Vector Processors
SMP: Symmetric Shared Memory Multiprocessors

Ver http://www.top500.org/

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

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")

Focus of this class: multiprocessor level of parallelism

Same story from memory system perspective

- Increase bandwidth, reduce average latency with many local memories

Wide range of parallel architectures make sense

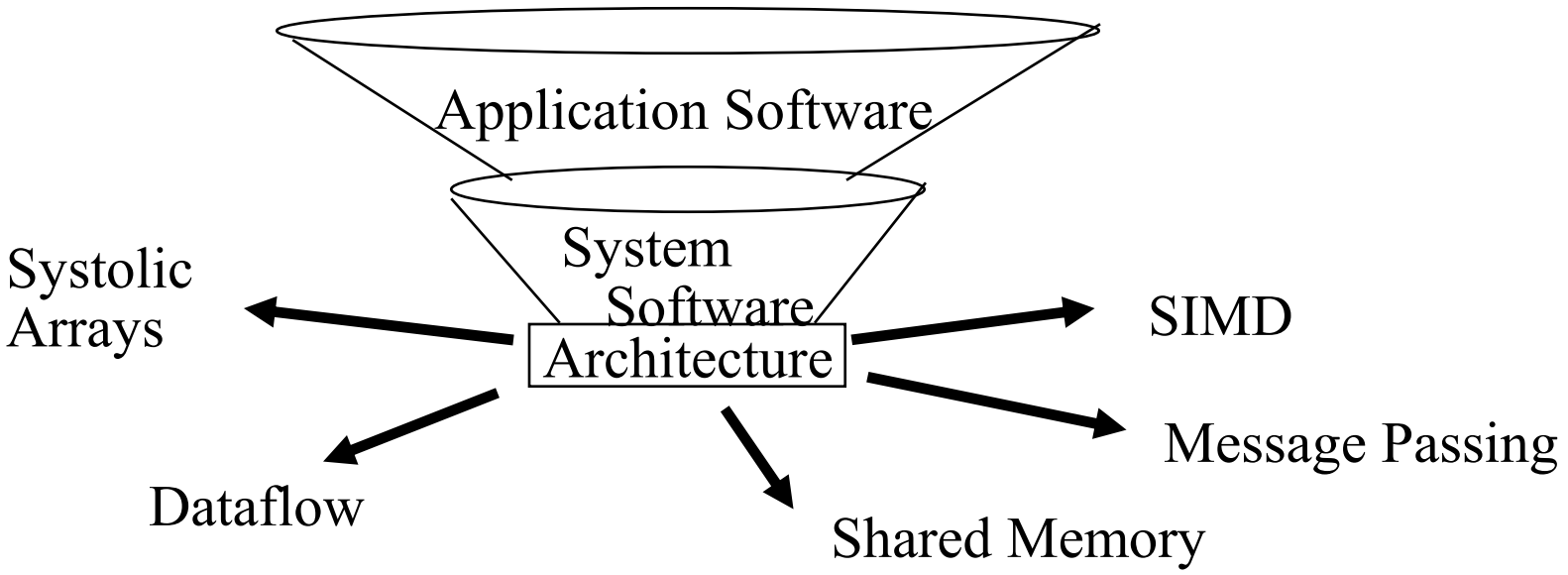
- Different cost, performance and scalability

1.2 Convergence of Parallel Architectures

History

Historically, parallel architectures tied to programming models

- Divergent architectures, with no predictable pattern of growth.



- Uncertainty of direction paralyzed parallel software development!

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

1.2.1 Today

Extension of "computer architecture" to support communication and cooperation

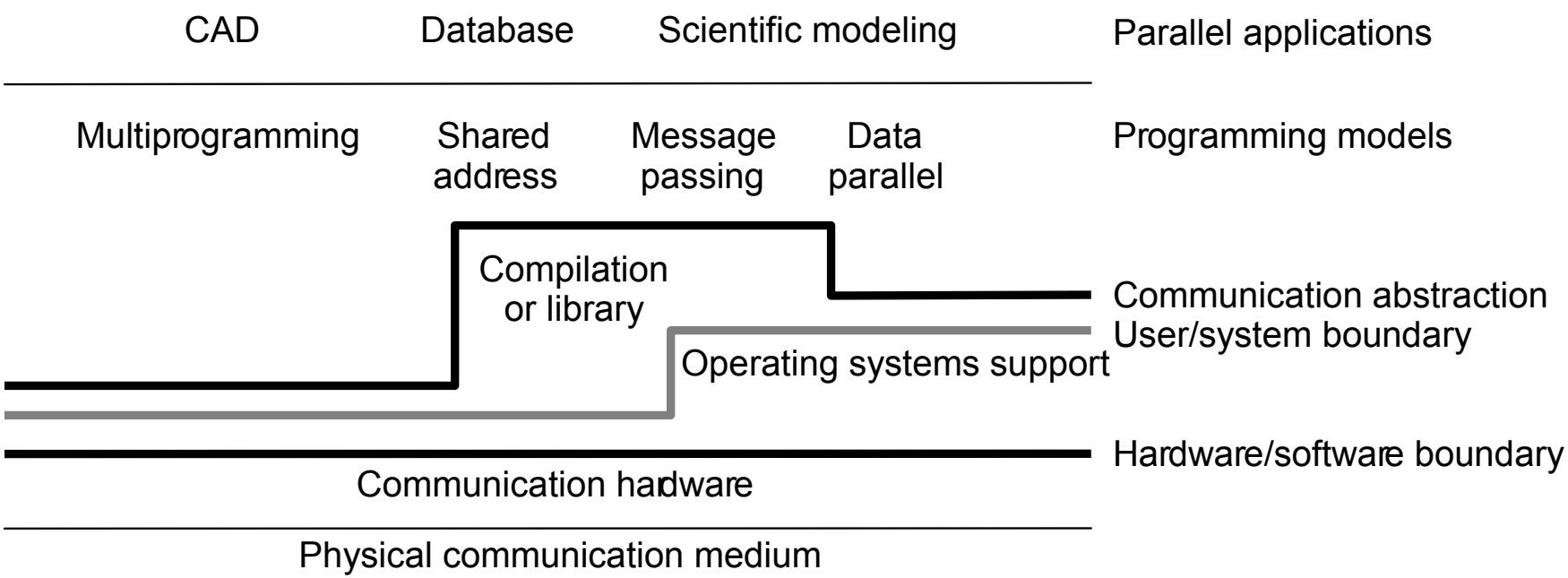
- OLD: Instruction Set Architecture
- NEW: *Communication Architecture*

Defines

- Critical abstractions, boundaries (HW/SW e user/system), and primitives (interfaces)
- Organizational structures that implement interfaces (hw or sw)

Compilers, libraries and OS are important bridges today

Modern Layered Framework



- Distância entre um nível e o próximo indicam se o mapeamento é simples ou não
  - ex: acesso a uma variável
    - SAS: simplesmente ld ou st
    - Message passing: envolve library ou system call

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Programming Model

What programmer uses in coding applications

Specifies communication and synchronization

Examples:

- Multiprogramming: no communication or synch. at program level
- *Shared address space*: like bulletin board
- *Message passing*: like letters or phone calls, explicit point to point
- *Data parallel*: more regimented, global actions on data
  - Implemented with shared address space or message passing

Communication Abstraction

User level communication primitives provided

- Realizes the programming model
- Mapping exists between language primitives of programming model and these primitives

Supported directly by hw, or via OS, or via user sw

Lot of debate about what to support in sw and gap between layers

Today:

- Hw/sw interface tends to be flat, i.e. complexity roughly uniform
- Compilers and software play important roles as bridges today
- Technology trends exert strong influence

Result is convergence in organizational structure

- Relatively simple, general purpose communication primitives

Communication Architecture

= ***User/System Interface + Implementation***

User/System Interface:

- Comm. primitives exposed to user-level by hw and system-level sw

Implementation:

- Organizational structures that implement the primitives: hw or OS
- How optimized are they? How integrated into processing node?
- Structure of network

Goals:

- Performance
- Broad applicability
- Programmability
- Scalability
- Low Cost

Evolution of Architectural Models

Historically machines tailored to programming models

- Prog. model, comm. abstraction, and machine organization lumped together as the “architecture”

Evolution helps understand convergence

- Identify core concepts

- Shared Address Space
- Message Passing
- Data Parallel

Others:

- Dataflow
- Systolic Arrays

Examine programming model, motivation, intended applications, and contributions to convergence

1.2.2 Shared Address Space Architectures

Any processor can directly reference any memory location

- Communication occurs implicitly as result of loads and stores

Convenient:

- Location transparency
- Similar programming model to time-sharing on uniprocessors
  - Except processes run on different processors
  - Good throughput on multiprogrammed workloads

Naturally provided on wide range of platforms

- History dates at least to precursors of mainframes in early 60s
- Wide range of scale: few to hundreds of processors

Popularly known as *shared memory* machines or model

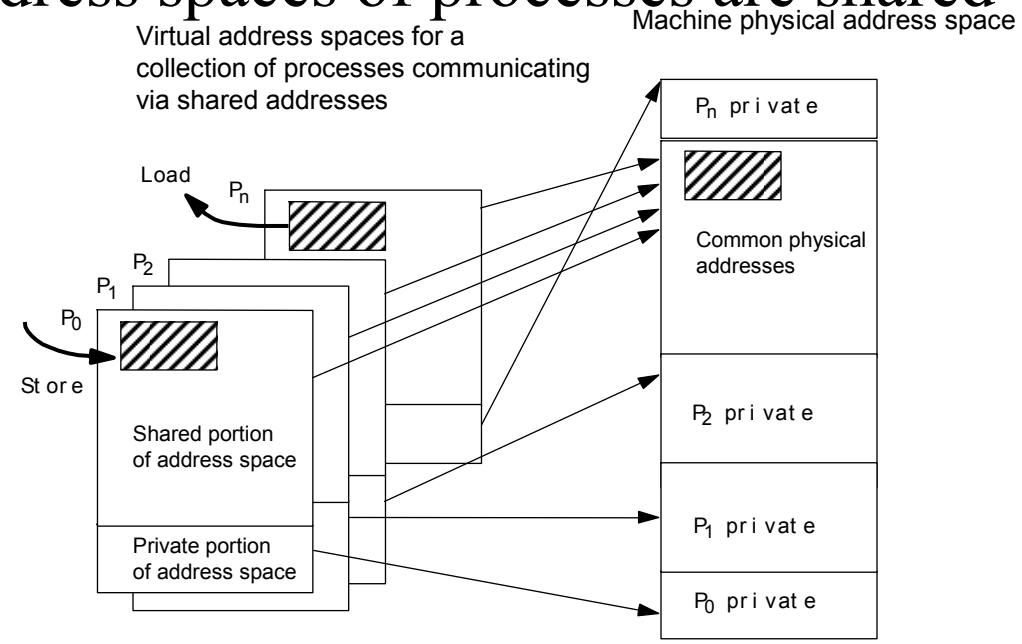
- Ambiguous: memory may be physically distributed among processors

SMP: shared memory multiprocessor

Shared Address Space Model

Process: virtual address space plus one or more threads of control

Portions of address spaces of processes are shared



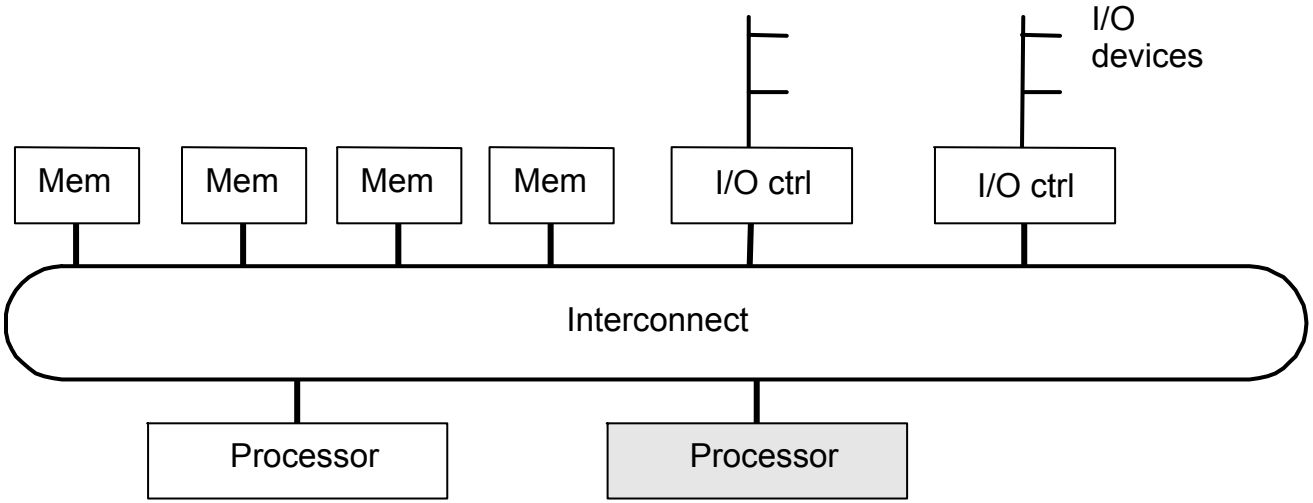
- Writes to shared address visible to other threads (in other processes too)
- Natural extension of uniprocessors model: conventional memory operations for comm.; special atomic operations for synchronization
- OS uses shared memory to coordinate processes

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Communication Hardware

Also natural extension of uniprocessor (estrutura apenas aumentada)

Already have processor, one or more memory modules and I/O controllers connected by hardware interconnect of some sort



Memory capacity increased by adding modules, I/O by controllers

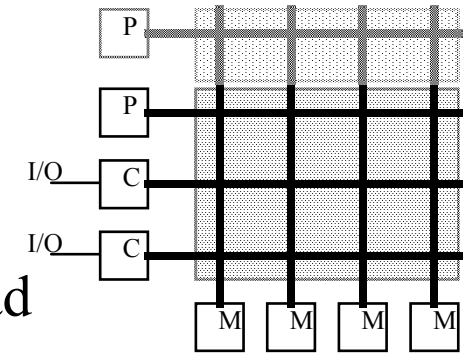
- Add processors for processing!
- For higher-throughput multiprogramming, or parallel programs

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

History

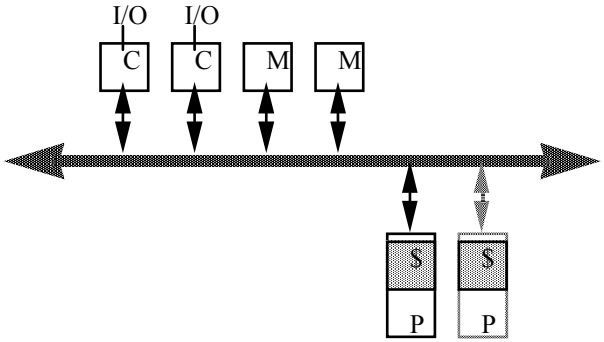
"Mainframe" approach

- Motivated by multiprogramming
- Extends crossbar used for mem bw and I/O
- Originally processor cost limited to small
  - later, cost of crossbar
- Bandwidth scales with *p*
- High incremental cost; use multistage instead



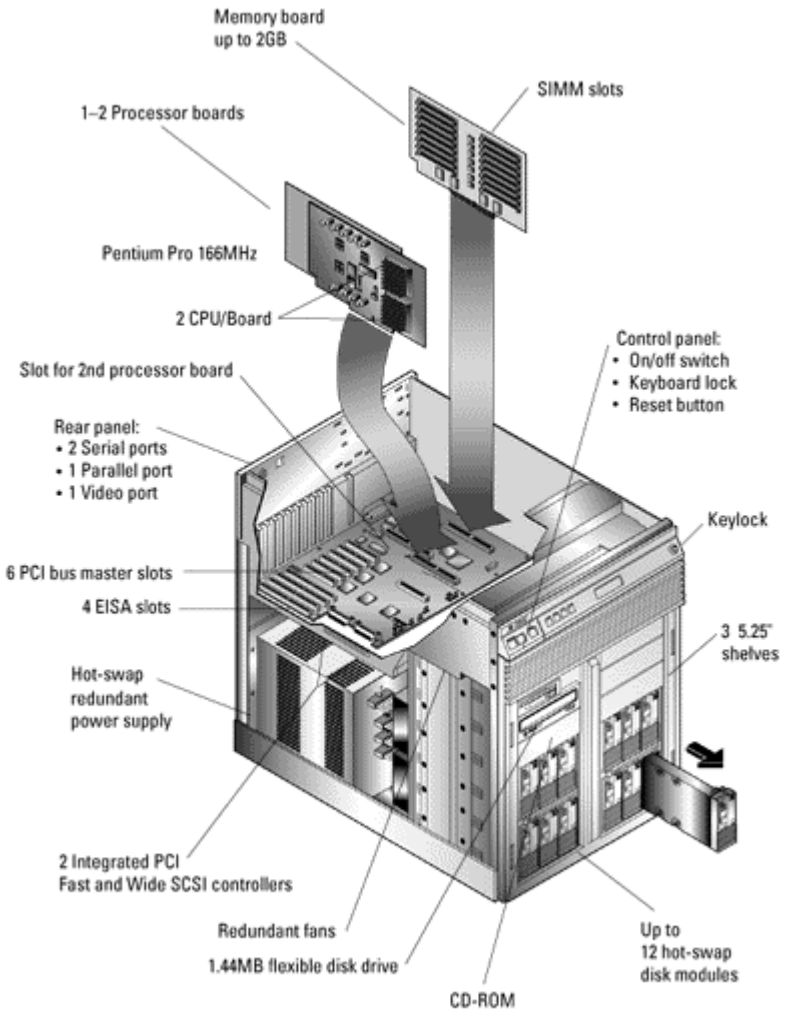
"Minicomputer" approach

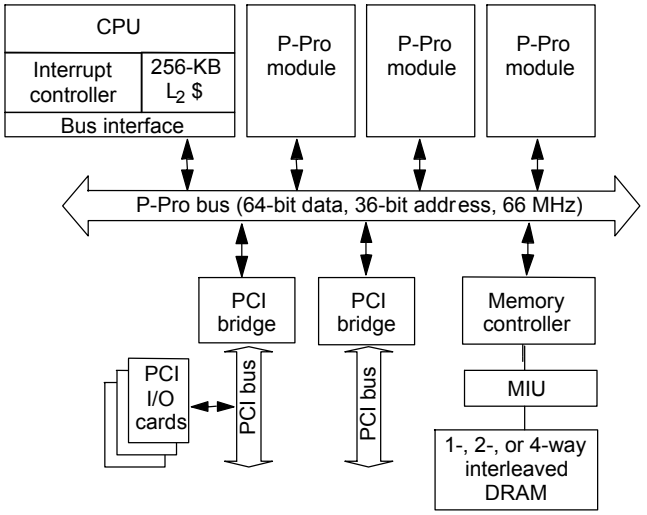
- Almost all microprocessor systems have bus
- Motivated by multiprogramming, TP
- Used heavily for parallel computing
- Called symmetric multiprocessor (SMP)
- Latency larger than for uniprocessor
- Bus is bandwidth bottleneck
  - caching is key: *coherence* problem
- Low incremental cost



(Ver fig. 1.16)

Example: Intel Pentium Pro Quad

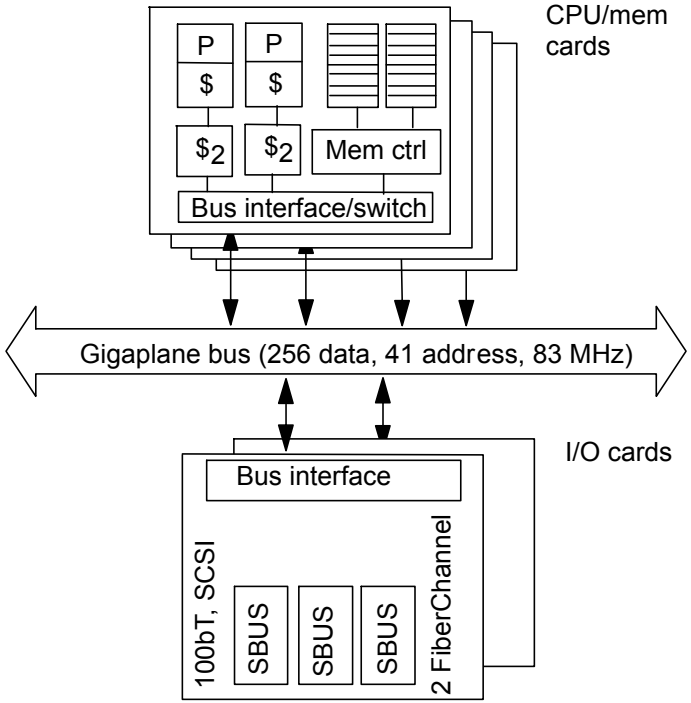




- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

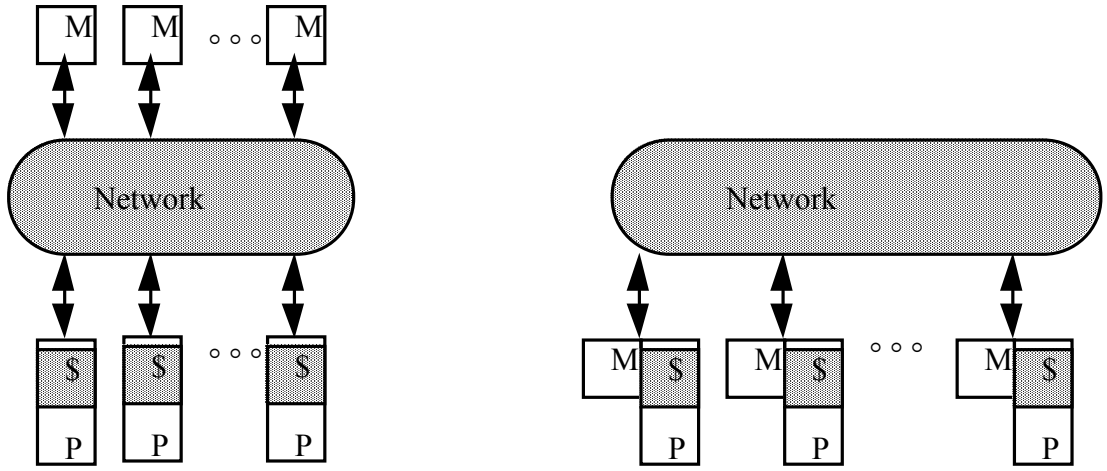
Example: SUN Enterprise



- 16 cards of either type: processors (UltraSparc) + memory, or I/O
- All memory accessed over bus, so symmetric
- Higher bandwidth, higher latency bus

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

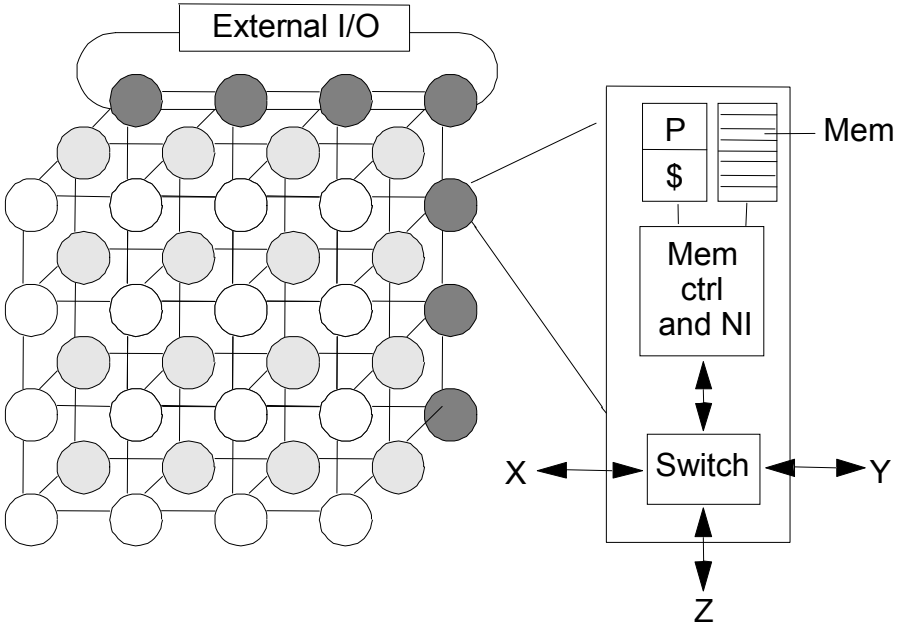
Scaling Up



"Dance hall" Distributed memory

- Problem is interconnect: cost (crossbar) or bandwidth (bus)
- Dance-hall: bandwidth still scalable, but lower cost than crossbar
  - latencies to memory uniform, but uniformly large
- Distributed memory or non-uniform memory access (NUMA)
  - Construct shared address space out of simple message transactions across a general-purpose network (e.g. read-request, read-response)
- Caching shared (particularly nonlocal) data?

Example (NUMA): Cray T3E



- Scale up to 1024 processors (Alpha, 6 vizinhos), 480MB/s links
- Memory controller generates comm. request for nonlocal references
- No hardware mechanism for coherence (SGI Origin etc. provide this)

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

1.2.3 Message Passing Architectures

Complete computer as building block, including I/O

- Communication via explicit I/O operations (e não via operações de memória)

Programming model: directly access only private address space (local memory), comm. via explicit messages (send/receive)

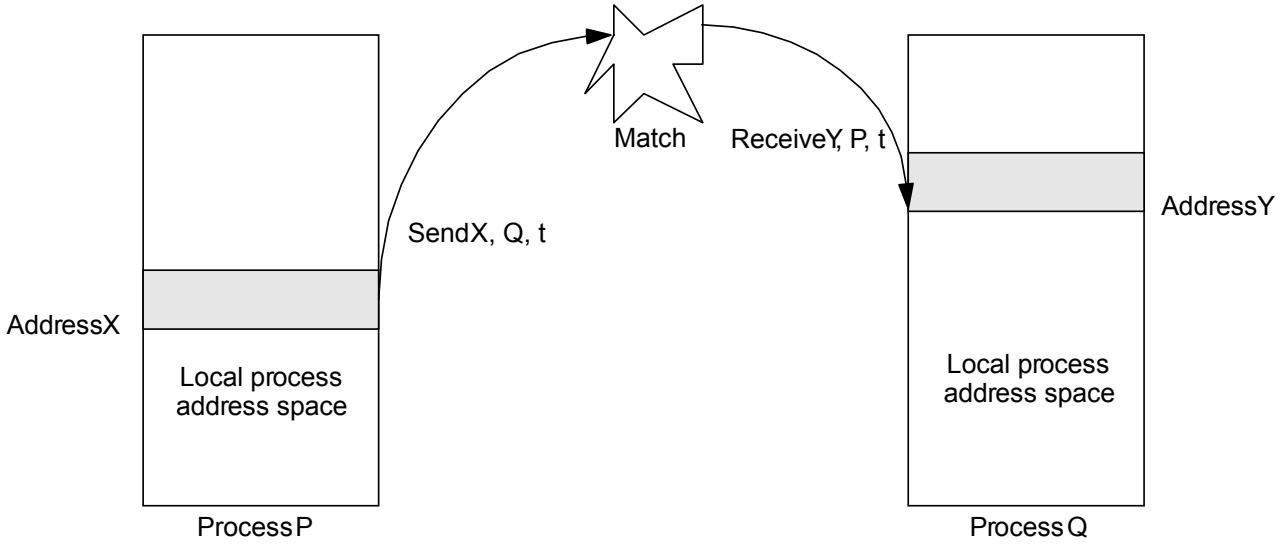
High-level block diagram similar to distributed-memory SAS

- But comm. integrated at IO level, needn't be into memory system
- Like networks of workstations (clusters), but tighter integration (não há monitor/teclado por nó)
- Easier to build than scalable SAS

Programming model more removed (mais distante) from basic hardware operations

- Library or OS intervention

Message-Passing Abstraction



- Send specifies (local) buffer to be transmitted and receiving process
- Recv specifies sending process and application storage to receive into
- Memory to memory copy, but need to name processes
- Optional tag on send and matching rule on receive
- User process names local data and entities in process/tag space too
- In simplest form, the send/recv match achieves pairwise synch event
  - Other variants too
- Many overheads: copying, buffer management, protection

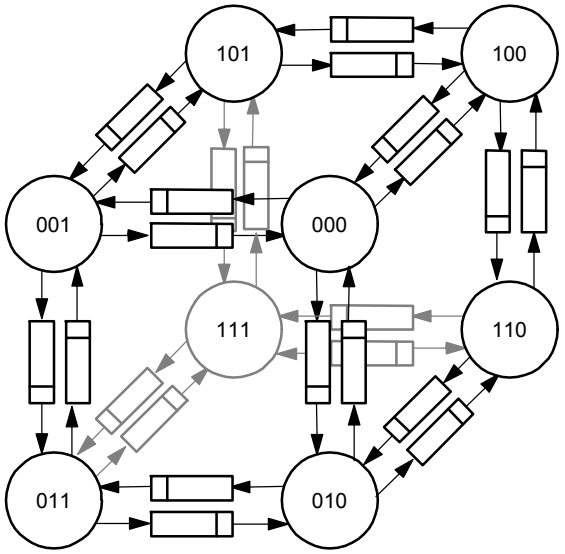
Evolution of Message-Passing Machines

Early machines (´85): FIFO on each link

- Hw close to prog. Model; synchronous ops
- Replaced by DMA, enabling non-blocking ops
  - Buffered by system at destination until recv

Diminishing role of topology

- No início, topologia importante (só nomear processador vizinho)
- Store&forward routing: topology important
- Introduction of pipelined routing made it less so
- Cost is in node-network interface
- Simplifies programming



Topologias típicas:

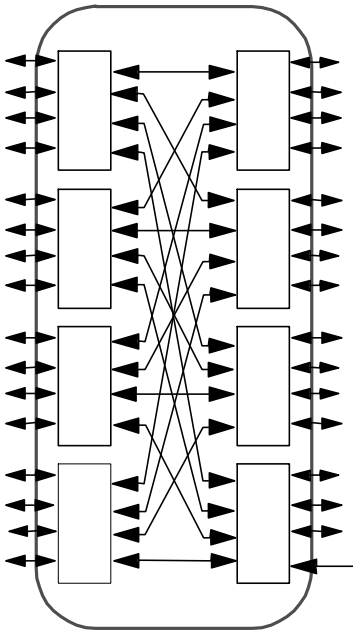
- hipercubo
- mesh

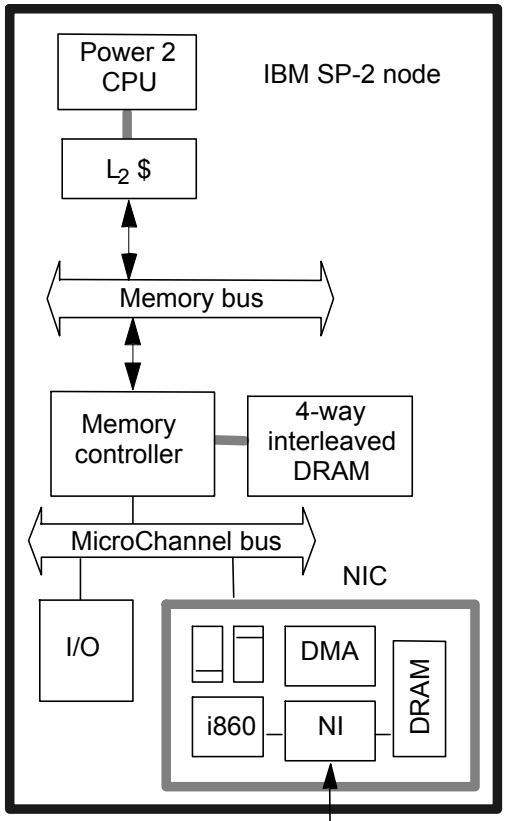
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Example: IBM SP-2

- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bw limited by I/O bus)

General interconnection network formed from 8-port switches

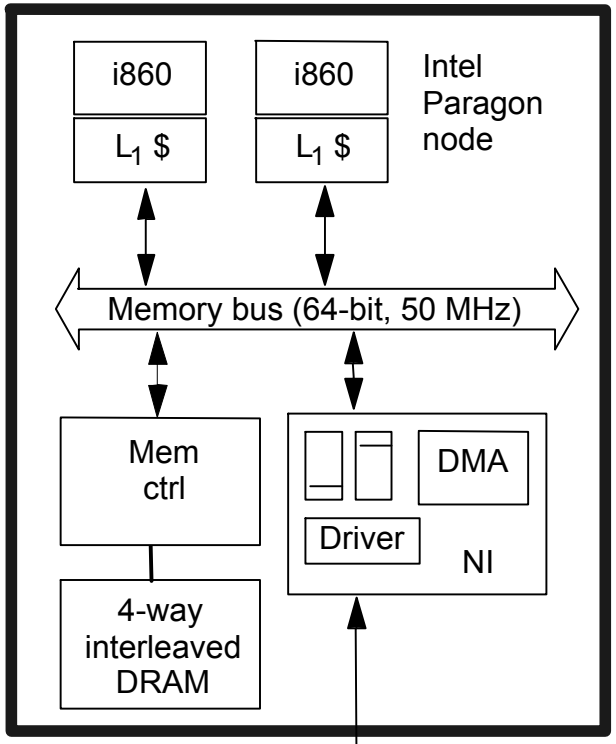


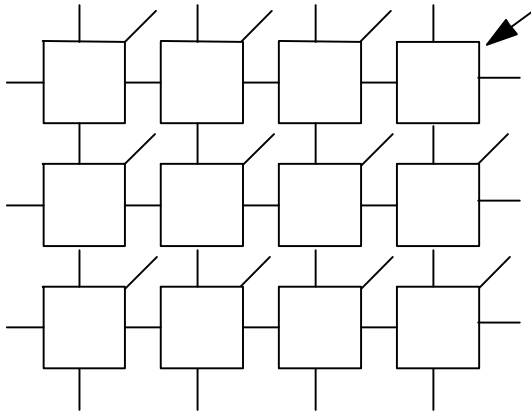


Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Example Intel Paragon

Sandia' s Intel Paragon XP/S-based Supercomputer





2D grid network
with processing node
attached to every switch

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

1.2.4 Toward Architectural Convergence

Evolution and role of software have blurred boundary (SAS x MP)

- Send/recv supported on SAS machines via buffers
- Can construct global address space on MP using hashing
- Page-based (or finer-grained) shared virtual memory

Hardware organization converging too

- Tighter NI integration even for MP (low-latency, high-bandwidth)
- At lower level, even hardware SAS passes hardware messages

Even clusters of workstations/SMPs are parallel systems (the network is the computer

- Emergence of fast system area networks (SAN)

Programming models distinct, but organizations converging

- Nodes connected by general network and communication assists
- Implementations also converging, at least in high-end machines

1.2.5 Data Parallel Systems

Outros nomes: processor array ou SIMD

Programming model

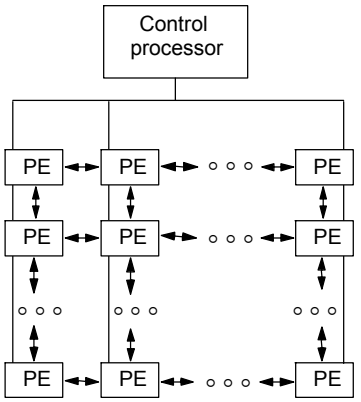
- Operations performed in parallel on each element of data structure (array ou vetor)
- Logically single thread of control, performs sequential or parallel steps
- Conceptually, a processor associated with each data element

Architectural model

- Array of many simple, cheap processors with little memory each
  - Processors don't sequence through instructions
- Attached to a control processor that issues instructions
- Specialized and general communication, cheap global synchronization

Original motivations

- Matches simple differential equation solvers
- Centralize high cost of instruction fetch/sequencing (que era grande)



Application of Data Parallelism

- Each PE contains an employee record with his/her salary

```
If salary > 100K then
   salary = salary *1.05
else
   salary = salary *1.10
```

- Logically, the whole operation is a single step
- Some processors enabled for arithmetic operation, others disabled

Other examples:

- Finite differences, linear algebra, ...
- Document searching, graphics, image processing, ...

Some recent machines:

- Thinking Machines CM-1, CM-2 (and CM-5) (ver fig 1.25)
- Maspar MP-1 and MP-2,

Evolution and Convergence

Rigid control structure (SIMD in Flynn taxonomy)

- SISD = uniprocessor, MIMD = multiprocessor

Popular when cost savings of centralized sequencer high

- 60s when CPU was a cabinet
- Replaced by vectors in mid-70s (grande simplificação)
  - More flexible w.r.t. memory layout and easier to manage
- Revived in mid-80s when 32-bit datapath slices just fit on chip (32 processadores de 1 bit em um único chip)
- No longer true with modern microprocessors

Other reasons for demise

- Simple, regular applications have good locality, can do well anyway (cache é mais genérica e funciona tão bem como)
- Loss of applicability due to hardwiring data parallelism
  - MIMD machines as effective for data parallelism and more general

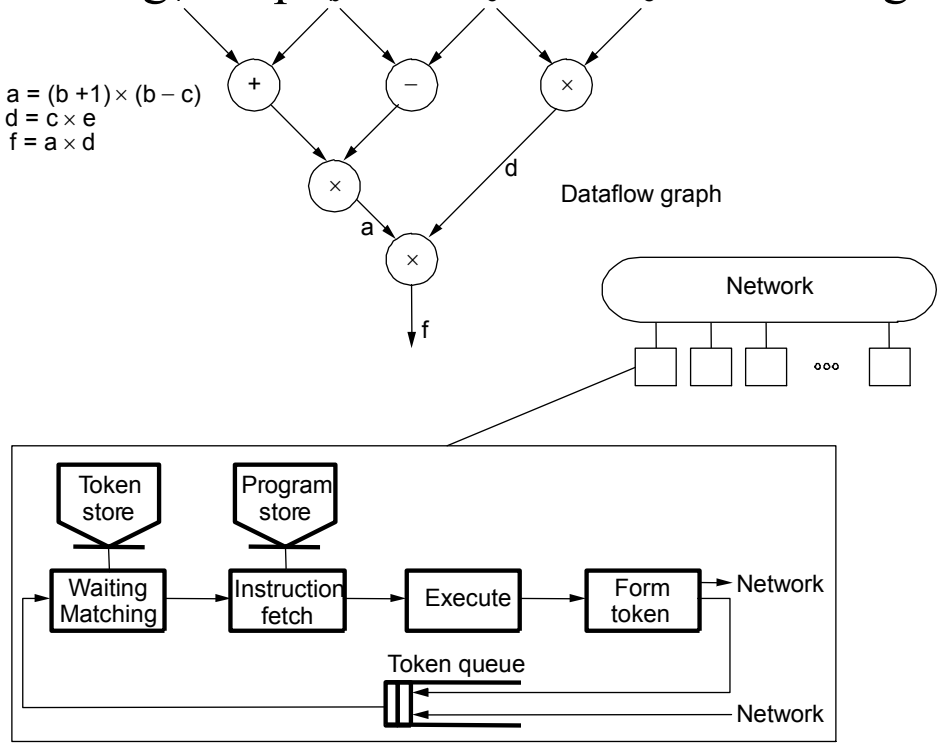
Prog. model converges with SPMD (single program multiple data)

- Contributes need for fast global synchronization
- Structured global address space, implemented with either SAS or MP

1.2.6 (1) Dataflow Architectures

Represent computation as a graph of essential dependences

- Logical processor at each node, activated by availability of operands
- Message (tokens) carrying tag of next instruction sent to next processor (message token = tag (address) + data)
- Tag$_1$ compared with others$_e$ in matching store; match fires execution



Busca instrução (token) do que fazer na rede; se "match" executa e passa resultado adiante

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Evolution and Convergence

Dataflow
- Estático: cada nó representa uma operação primitiva
- Dinâmico: função complexa executada pelo nó

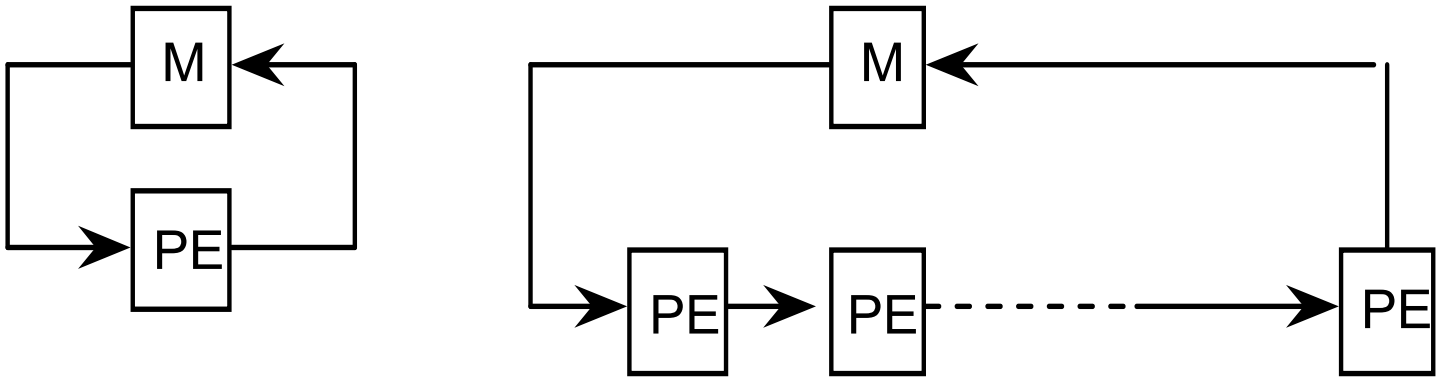
Key characteristics
- Ability to name operations, synchronization, dynamic scheduling

Converged to use conventional processors and memory
- Support for large, dynamic set of threads to map to processors
- Typically shared address space as well
- But separation of progr. model from hardware (like data-parallel)

Lasting contributions:
- Integration of communication with thread (handler) generation
- Tightly integrated communication and fine-grained synchronization
- Remained useful concept for software (compilers etc.)

1.2.6 (2) Systolic Architectures

- Replace single processor with array of regular processing elements
- Orchestrate data flow for high throughput with less memory access



Different from pipelining

- Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory

Different from SIMD: each PE may do something different

Initial motivation: VLSI enables inexpensive **special-purpose** chips

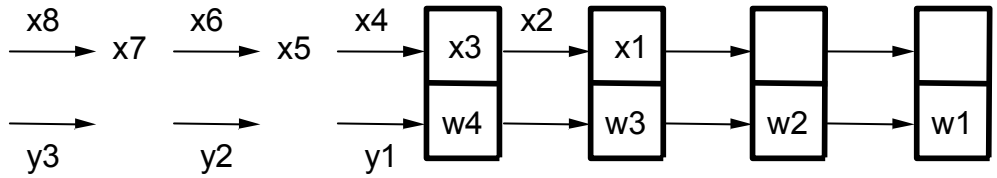
Represent algorithms directly by chips connected in regular pattern

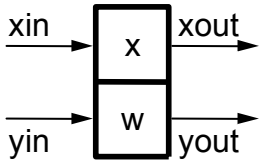
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Systolic Arrays (contd.)

Example: Systolic array for 1-D convolution

$$y(i) = w1 \times x(i) + w2 \times x(i + 1) + w3 \times x(i + 2) + w4 \times x(i + 3)$$





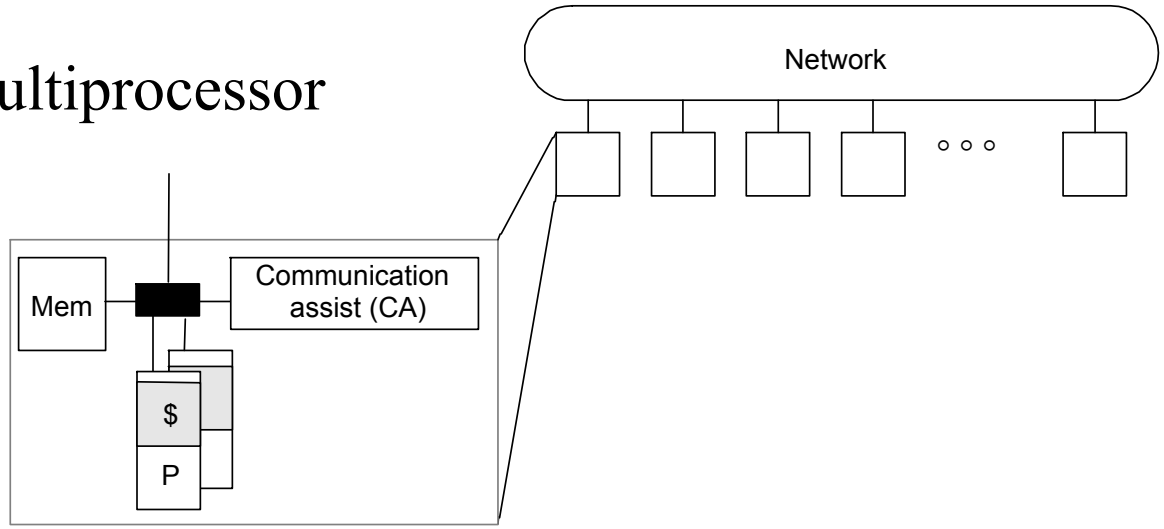
xout = x
x = xin
yout = yin + w × xin

- Practical realizations (e.g. iWARP) use quite general processors
  - Enable variety of algorithms on same hardware
- But dedicated interconnect channels
  - Data transfer directly from register to register across channel
- Specialized, and same problems as SIMD
  - General purpose systems work well for same algorithms (locality etc.)

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

1.2.7 Convergence: Generic Parallel Architecture

A generic modern multiprocessor



Node: processor(s), memory system, plus *communication assist*

- Network interface and communication controller

- Scalable network
- Convergence allows lots of innovation, now within framework
  - Integration of assist with node, what operations, how efficiently...
- Modelo de programação -> efeito no Communication Assist
  - Ver efeito para SAS, MP, Data Parallel e Systolic Array

1.3 Fundamental Design Issues

Understanding Parallel Architecture

Traditional taxonomies not very useful (SIMD/MIMD) (porque multiple general purpose processors are dominant)

Focusing on programming models not enough, nor hardware structures

- Same one can be supported by radically different architectures

Foco deve ser em: Architectural distinctions that affect software

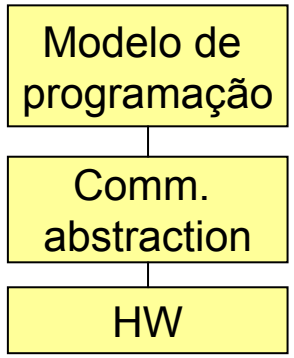
- Compilers, libraries, programs

Design of user/system and hardware/software interface (Decisões)

- Constrained from above by progr. models and below by technology

Guiding principles provided by layers

- What primitives are provided at communication abstraction
- How programming models map to these
- How they are mapped to hardware



Communication Abstraction: interface entre o modelo de programação e a implem. do sistema: importância equivalente ao conjunto de instruções em computadores convencionais

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Fundamental Design Issues

At any layer, interface (contrato entre HW e SW) aspect and performance aspects (deve permitir melhoria individual)

Data *named* by threads; *operations* performed on named data; *ordering* among operations

- *Naming*: How are logically shared data and/or processes referenced?
- *Operations*: What operations are provided on these data
- *Ordering*: How are accesses to data ordered and coordinated?
- Replication: How are data replicated to reduce communication?
- Communication Cost: Latency, bandwidth, overhead, occupancy

Understand at programming model first, since that sets requirements

Other issues

- Node Granularity: How to split between processors and memory?

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Sequential Programming Model

Contract

- Naming: Can name any variable in virtual address space (exemplo em uniprocessadores)
  - Hardware (and perhaps compilers) does translation to physical addresses
- Operations: Loads and Stores
- Ordering: Sequential program order

Performance (sequential programming model)

- Rely on dependences on single location (mostly): *dependence order*
- Compilers and hardware violate other orders without getting caught
- Compiler: reordering and register allocation
- Hardware: out of order, pipeline bypassing, write buffers
- Transparent replication in caches

SAS Programming Model

Naming: Any process can name any variable in shared space

Operations: loads and stores, plus those needed for ordering

Simplest Ordering Model:

- Within a process/thread: sequential program order
- Across threads: some interleaving (as in time-sharing)
- Additional orders through synchronization
- Again, compilers/hardware can violate orders without getting caught
  - Different, more subtle ordering models also possible (discussed later)

Synchronization

Mutual exclusion (locks)

- Ensure certain operations on certain data can be performed by only one process at a time
- Room that only one person can enter at a time
- No ordering guarantees (ordem não interessa; o importante é que apenas um tenha acesso por vez)

Event synchronization

- Ordering of events to preserve dependences
  - Passagem de bastão
  - e.g. producer —> consumer of data
- 3 main types:
  - point-to-point
  - global
  - group

Message Passing Programming Model

Naming: Processes can name private data directly (or can name other processes) (private data space <-> global process space)

- No shared address space

Operations: Explicit communication through *send* and *receive*

- Send transfers data from private address space to another process
- Receive copies data from process to private address space
- Must be able to name processes

Ordering:

- Program order within a process
- Send and receive can provide pt to pt synch between processes
- Mutual exclusion inherent

Can construct global address space:

- Process number + address within process address space
- But no direct operations on these names

Design Issues Apply at All Layers

Prog. model's position provides constraints/goals for system

In fact, each interface between layers supports or takes a position on:

- Naming model
- Set of operations on names
- Ordering model
- Replication
- Communication performance

Any set of positions can be mapped to any other by software

Let's see issues across layers

- How lower layers can support contracts of programming models
- Performance issues

Naming and Operations

Naming and operations in programming model can be directly supported by lower levels (uniforme em todos os níveis de abstração), or translated by compiler, libraries or OS

Example: **Shared virtual address** space in programming model

Alt1: Hardware interface supports *shared (global) physical address space*

- Direct support by hardware through v-to-p mappings (comum para todos os processadores), no software layers

Alt2: Hardware supports independent physical address spaces (cada processador pode acessar áreas físicas distintas)

- Can provide SAS through OS, so in system/user interface
  - v-to-p mappings only for data that are local
  - remote data accesses incur page faults; brought in via page fault handlers
  - same programming model, different hardware requirements and cost model

Naming and Operations (contd)

Example: Implementing **Message Passing**

Alt1: Direct support at hardware interface

- But match and buffering benefit from more flexibility

Alt2: Support at sys/user interface or above in software (almost always)

- Hardware interface provides basic data transport (well suited)
- Send/receive built in sw for flexibility (protection, buffering)
- Choices at user/system interface:
  - Alt2.1: OS each time: expensive
  - Alt2.2: OS sets up once/infrequently, then little sw involvement each time (setup com OS e execução com HW)
- Alt2.3: Or lower interfaces provide SAS (virtual), and send/receive built on top with buffers and loads/stores (leitura/escrita em buffers + sincronização)

Need to examine the issues and tradeoffs at every layer

- Frequencies and types of operations, costs

Ordering

Message passing: no assumptions on orders across processes except those imposed by send/receive pairs

SAS: How processes see the order of other processes' references defines semantics of SAS

- Ordering very important and subtle
- Uniprocessors play tricks with orders to gain parallelism or locality
- These are more important in multiprocessors
- Need to understand which old tricks are valid, and learn new ones
- How programs behave, what they rely on, and hardware implications

1.3.3 Replication

Very important for reducing data transfer/communication

Again, depends on naming model

Uniprocessor: caches do it automatically

- Reduce communication with memory

Message Passing naming model at an interface

- A receive replicates, giving a new name; subsequently use new name
- Replication is explicit in software above that interface

SAS naming model at an interface

- A load brings in data transparently, so can replicate transparently
- Hardware caches do this, e.g. in shared physical address space
- OS can do it at page level in shared virtual address space, or objects
- No explicit renaming, many copies for same name: *coherence* problem
  - in uniprocessors, "coherence" of copies is natural in memory hierarchy

Obs: communication = entre processos (não equivalente a data transfer)

1.3.4 Communication Performance

Performance characteristics determine usage of operations at a layer

- Programmer, compilers etc make choices based on this (evitam operações custosas)

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 $\propto$ 1/latency (custo = latência * nº de operações)

- But actually more complex in modern systems

Characteristics apply to overall operations, as well as individual components of a system, however small

We'll focus on communication or data transfer across nodes

Simple Example (expl 1.2)

Component performs an operation in 100ns (latência)

(portanto) Simple bandwidth: 10 Mops

Internally pipeline depth 10 => bandwidth 100 Mops

- Rate determined by slowest stage of pipeline, not overall latency (se operação executada a cada 200ns -> bandwitdh = 5Mops ->pipeline não efetivo)

Delivered bandwidth on application depends on initiation frequency (quantas vezes sequência é executada)

Suppose application performs 100 M operations. What is cost?

- op count * op latency gives 10 sec (upper bound) (100E6*100E-9=10) (se não é possível usar pipeline)
- op count / peak op rate gives 1 sec (lower bound) (se for possível uso completo do pipeline -> 10x)
  - assumes full overlap of latency with useful work, so just issue cost
- if application can do 50 ns of useful work (em média) before depending on result of op, cost to application is the other 50ns of latency (100E6*50E-9=5)

Linear Model of Data Transfer Latency

Transfer time (n) = T_0 + n/B

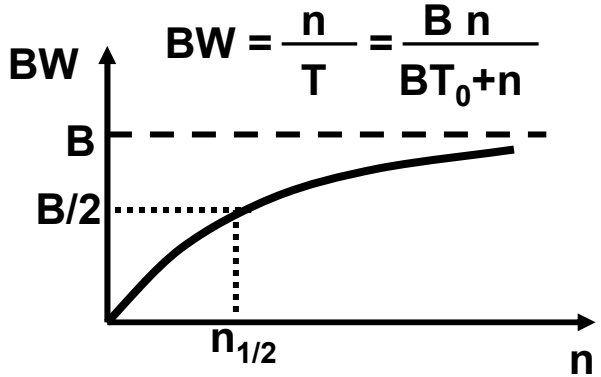
- T_0 = startup; n= bytes; B= bandwidth
- Model useful for message passing (T_0= latência 1ºbit), memory access (T_0= tempo de acesso) , bus (T_0= arbitration), pipeline (T_0= encher pipeline) vector ops etc

As *n* increases, bandwidth approaches asymptotic rate *B*

How quickly it approaches depends on T_0

Size needed for half bandwidth (half-power point):

$$n_{1/2} = T_0 * B \quad \textit{(ver errata no livro texto)}$$



But linear model not enough

- When can next transfer be initiated? Can cost be overlapped?
- Need to know how transfer is performed

Communication Cost Model

Comm Time per message= Overhead + Assist Occupancy +
Network Delay + Size/Bandwidth + Contention

$$= o_v + o_c + l + n/B + T_c$$

Overhead and assist occupancy may be *f(n)* or not

Each component along the way has occupancy and delay

- Overall delay is sum of delays
- Overall occupancy (1/bandwidth) is biggest of occupancies (gargalo)
- Próxima transferência de dados só pode começar se recursos críticos estão livres (assumindo que não há buffers no caminho)

Comm Cost = frequency * (Comm time - overlap)

General model for data transfer: applies to cache misses too

Summary of Design Issues

Functional and performance issues apply at all layers

Functional: Naming, operations and ordering

Performance: Organization, latency, bandwidth, overhead, occupancy

Replication and communication are deeply related

- Management depends on naming model

Goal of architects: design against frequency and type of operations that occur at communication abstraction, constrained by tradeoffs from above or below

- Hardware/software tradeoffs

Recap

Parallel architecture is important thread in evolution of architecture

- At all levels
- Multiple processor level now in mainstream of computing

Exotic designs have contributed much, but given way to convergence

- Push of technology, cost and application performance
- Basic processor-memory architecture is the same
- Key architectural issue is in communication architecture
  - How communication is integrated into memory and I/O system on node

Fundamental design issues

- Functional: naming, operations, ordering
- Performance: organization, replication, performance characteristics

Design decisions driven by workload-driven evaluation

- Integral part of the engineering focus

Outline for Rest of Class

Understanding parallel programs as workloads

- Much more variation, less consensus and greater impact than in sequential

- What they look like in major programming models (Ch. 2)
- Programming for performance: interactions with architecture (Ch. 3)
- Methodologies for workload-driven architectural evaluation (Ch. 4)

Cache-coherent multiprocessors with centralized shared memory

- Basic logical design, tradeoffs, implications for software (Ch 5)
- Physical design, deeper logical design issues, case studies (Ch 6)

Scalable systems

- Design for scalability and realizing programming models (Ch 7)
- Hardware cache coherence with distributed memory (Ch 8)
- Hardware-software tradeoffs for scalable coherent SAS (Ch 9)

Outline (contd.)

Interconnection networks (Ch 10)

Latency tolerance (Ch 11)

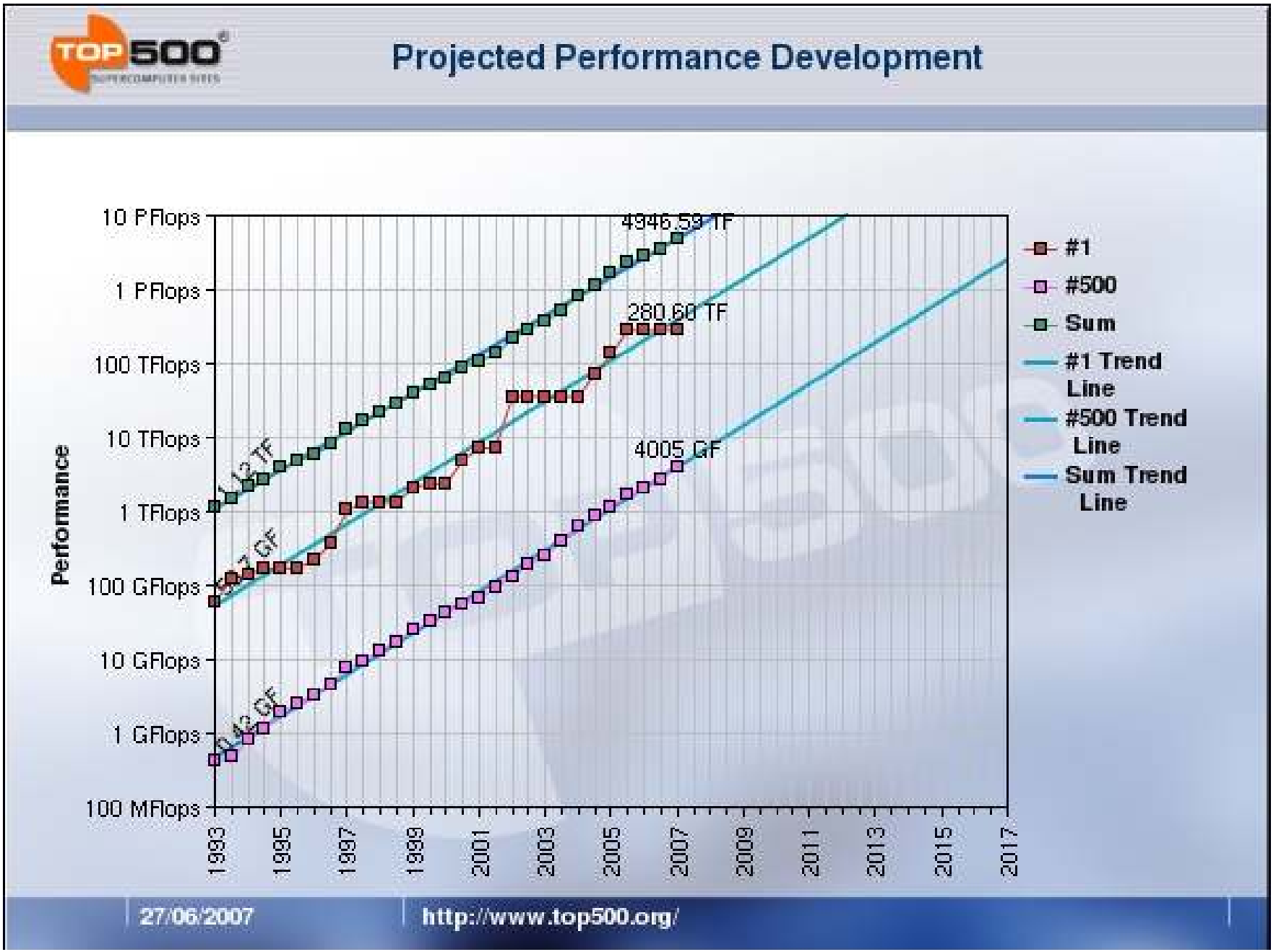
Future directions (Ch 12)

Overall: conceptual foundations and engineering issues across broad range of scales of design, all of which are important

Top 500 em jun/08 (5 primeiros)

- The new No. 1 system, built by IBM for the U.S. Department of Energy's Los Alamos National Laboratory and and named "Roadrunner," by LANL after the state bird of New Mexico achieved performance of 1.026 petaflop/s—becoming the first supercomputer ever to reach this milestone. At the same time, Roadrunner is also one of the most energy efficient systems on the TOP500
- Blue Gene/L, with a performance of 478.2 teraflop/s at DOE's Lawrence Livermore National Laboratory
- IBM BlueGene/P (450.3 teraflop/s) at DOE's Argonne National Laboratory,
- Sun SunBlade x6420 "Ranger" system (326 teraflop/s) at the Texas Advanced Computing Center at the University of Texas – Austin
- The upgraded Cray XT4 "Jaguar" (205 teraflop/s) at DOE's Oak Ridge National Laboratory

Top 500 em jul/07:projeções



Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top 500 em jul/08

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top 500 em jun/09 (10 primeiros)

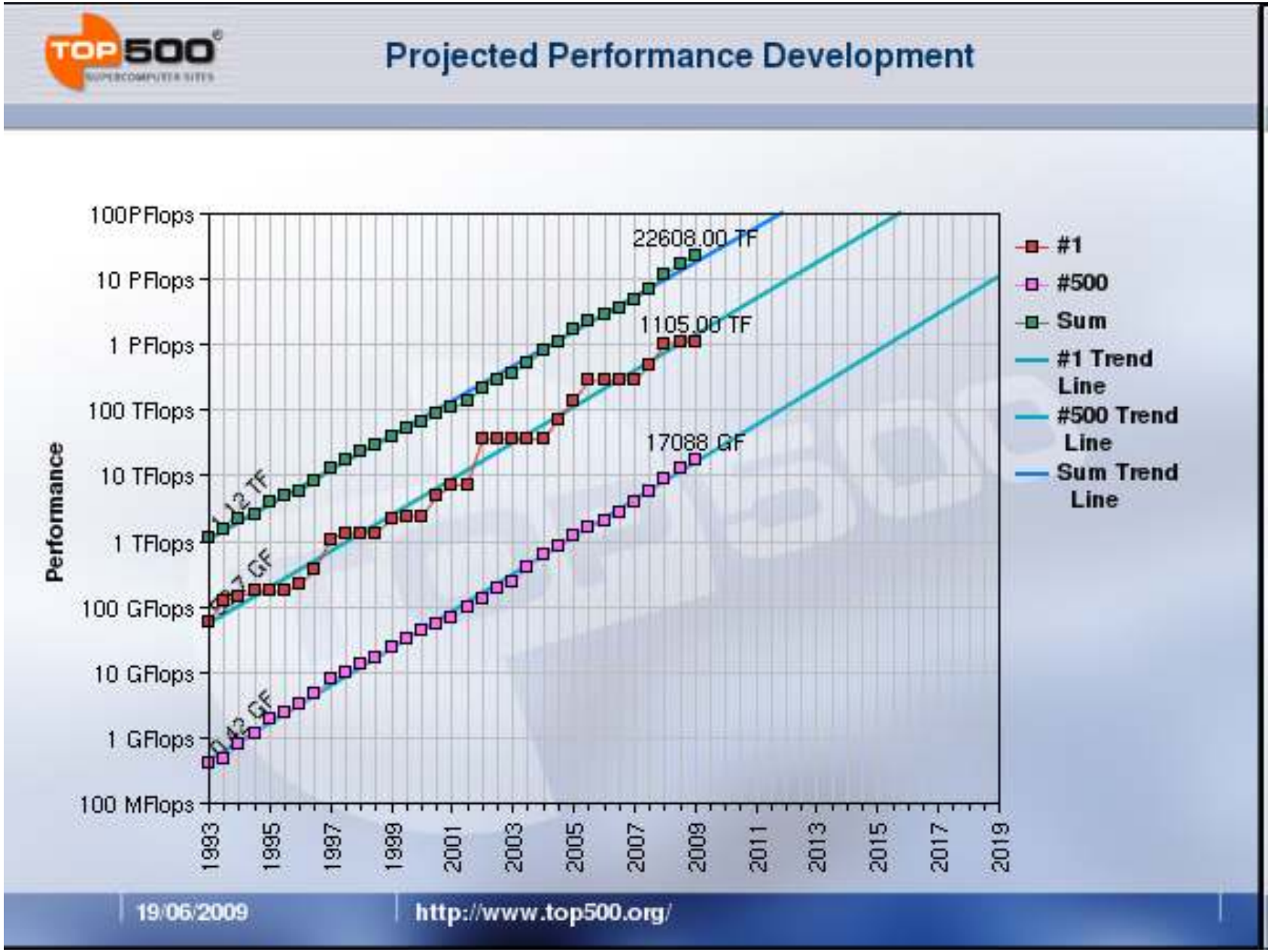
1	DOE/NNSA/LANL	United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband	IBM
2	Oak Ridge National Laboratory	United States	Jaguar - Cray XT5 QC 2.3 GHz	Cray Inc.
3	Forschungszentrum Juelich (FZJ)	Germany	JUGENE - Blue Gene/P Solution	IBM
4	NASA/Ames Research Center/NAS	United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz	SGI
5	DOE/NNSA/LLNL	United States	BlueGene/L - eServer Blue Gene Solution	IBM
6	National Institute for Computational Sciences/University of Tennessee	United States	Kraken XT5 - Cray XT5 QC 2.3 GHz	Cray Inc.
7	Argonne National Laboratory	United States	Blue Gene/P Solution	IBM
8	Texas Advanced Computing Center/Univ. of Texas	United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband	Sun Microsystems
9	DOE/NNSA/LLNL	United States	Dawn - Blue Gene/P Solution	IBM
10	Forschungszentrum Juelich (FZJ)	Germany	JUROPA - Sun Constellation, NovaScale R422-E2, Intel Xeon X5570, 2.93 GHz, Sun M9/Mellanox QDR Infiniband/Partec Parastation	Bull SA

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top 500 em jun/09

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Projeções em jun/09



Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top em jun/2010

1 Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz

2 Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU (China)

3 Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband

4 Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz

5 JUGENE - Blue Gene/P Solution

6 Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon Westmere 2.93 Ghz, Infiniband

7 Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband

8 BlueGene/L - eServer Blue Gene Solution

9 Intrepid - Blue Gene/P Solution

10 Red Sky - Sun Blade x6275, Xeon X55xx 2.93 Ghz, Infiniband

Top 500 em jun/2010

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top em jun/2011

	Site	Computer
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu InterConnect Fujitsu
2	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C NUDT
3	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz Cray Inc.
4	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU Dawning
5	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows NEC/HP
6	DOE/NNSA/LANL/SNL United States	Cielo - Cray XE6 8-core 2.4 GHz Cray Inc.
7	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband SGI
8	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz Cray Inc.
9	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 Bull SA
10	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband IBM

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Top 500 em jun/2011

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Some Other Highlights from the newest List

The two Chinese systems at No. 2 and No. 4 and the Japanese Tsubame 2.0 system at No. 5 are all using NVIDIA GPUs to accelerate computation, and a total of 19 systems on the list are using GPU technology.

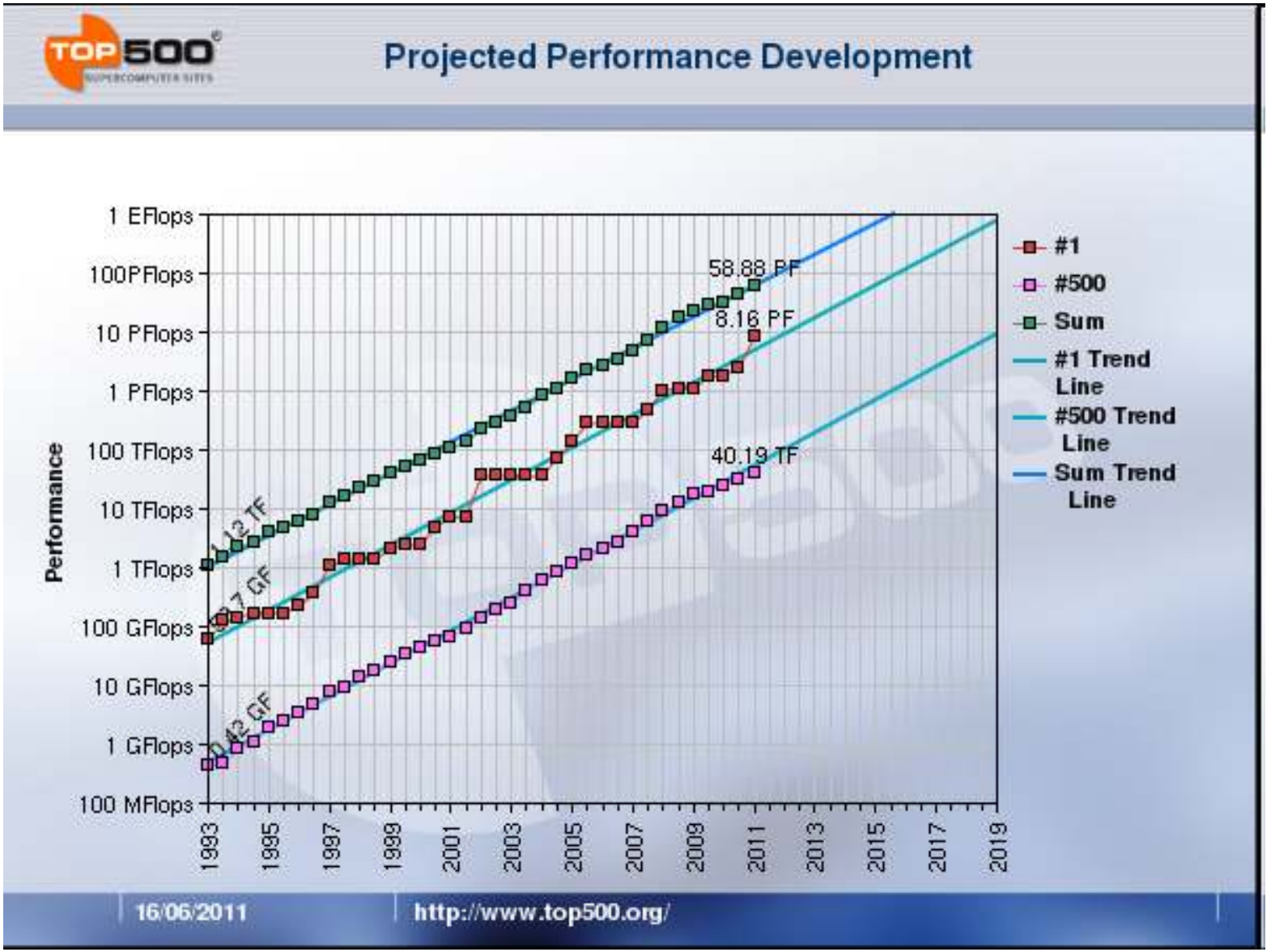
China keeps increasing its number of systems and is now up to 62, making it clearly the No. 2 country as a user of HPC, ahead of Germany, UK, Japan and France.

Intel continues to provide the processors for the largest share (77.4 percent) of TOP500 systems. Intel's Westmere processors increased their presence in the list strongly with 169 systems, compared with 56 in the last list.

Quad-core processors are used in 46.2 percent of the systems, while already 42.4 percent of the systems use processors with six or more cores.

Cray defended the No. 2 spot in market share by total against Fujitsu, but IBM stays well ahead of either. Cray's XT system series remains very popular for big research customers, with three systems in the TOP 10 (one new and two previously listed).

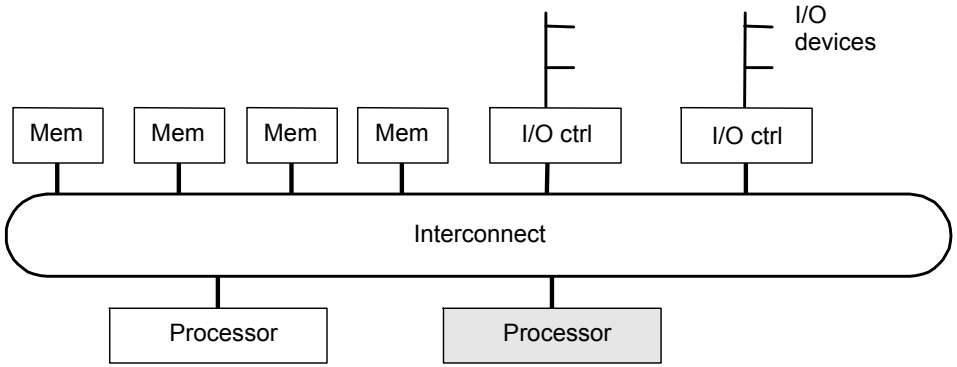
Projected Performance @ 2011



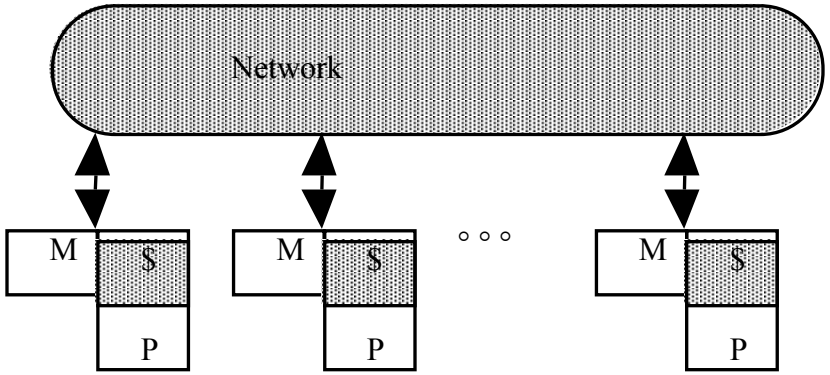
Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Models of Shared-Memory Multiprocessors

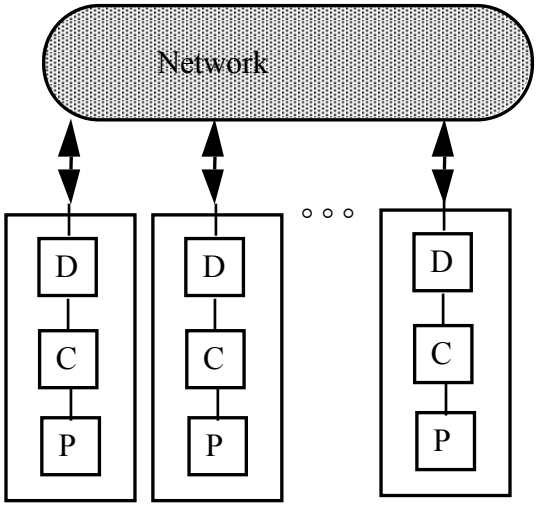
Uniform Memory Access (UMA) Model
or Symmetric Memory Processors (SMPs).



Interconnect:
Bus, Crossbar, Multistage network
P: Processor
M: Memory
C: Cache
D: Cache directory



Distributed memory or
Non-uniform Memory Access (NUMA) Model



Cache-Only Memory Architecture (COMA)

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Performance flatlining

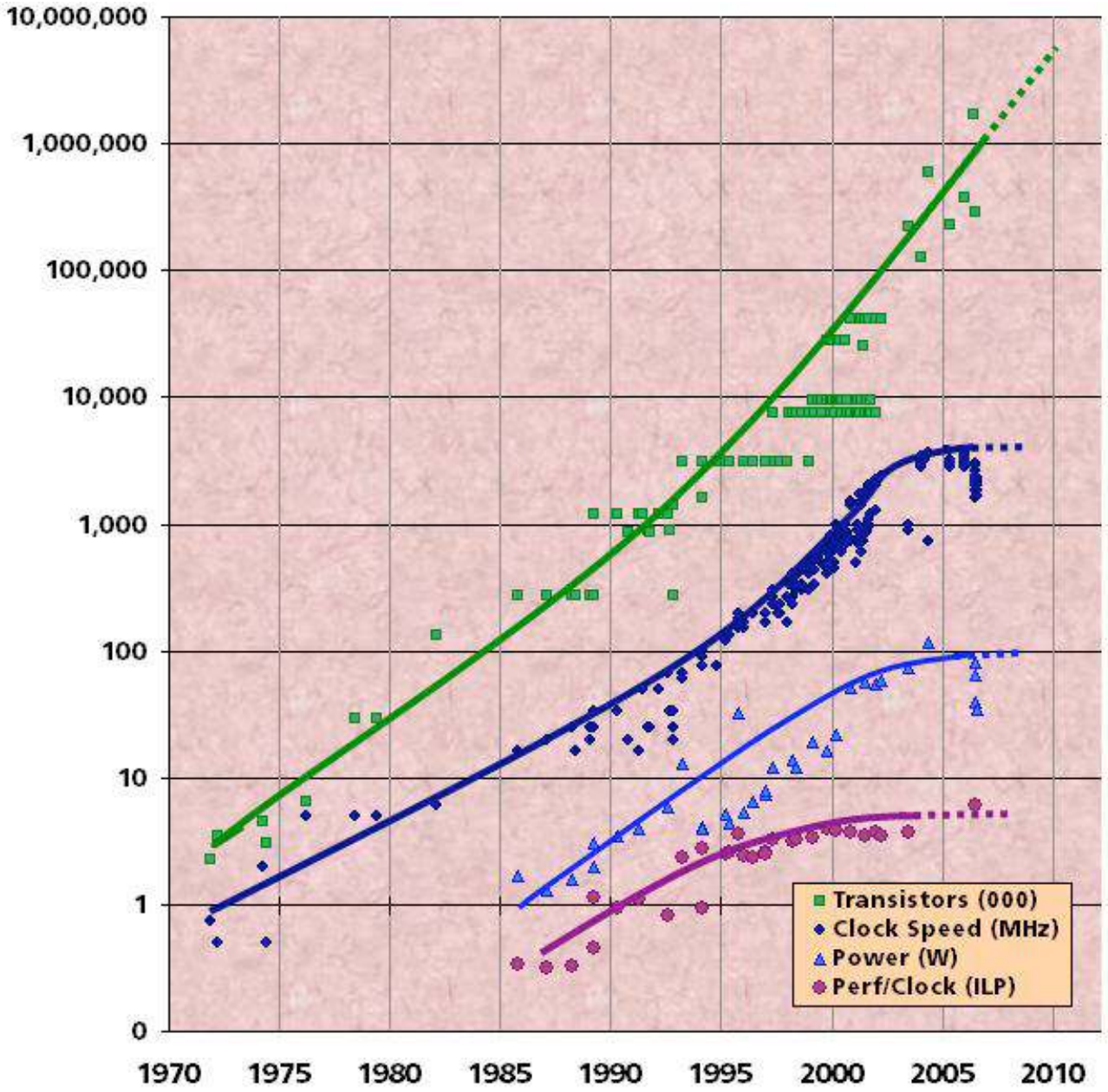


Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Previsão de crescimento do clock

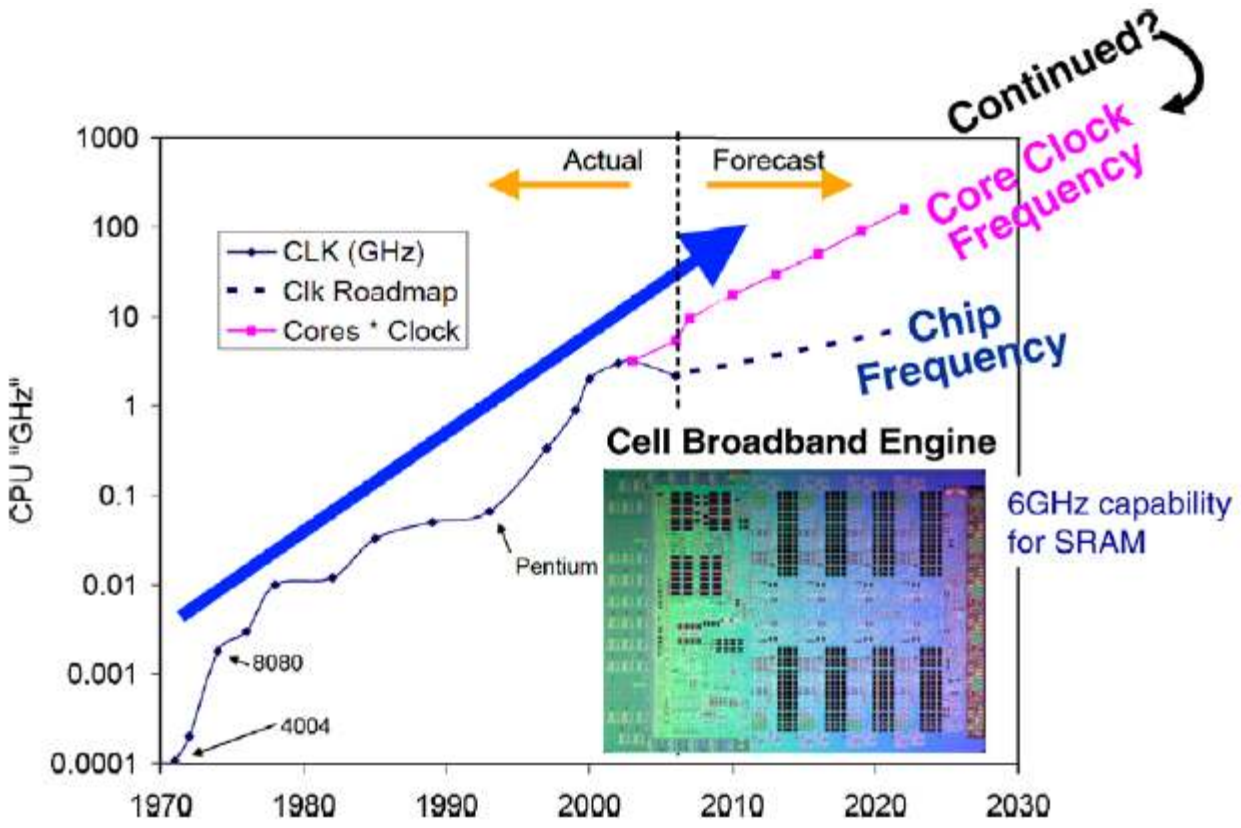


Fig. 7. Clock frequency trend in ITRS 2007 for local on-chip (Core) and entire chip.

H. Iwai, Microelectron. Eng. (2009), doi:10.1016/j.mee.2009.03.129

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp

Changing Conventional Wisdom

Old Conventional Wisdom: Power is free, Transistors expensive

New Conventional Wisdom: "Power wall" Power expensive, Xtors free
(Can put more on chip than can afford to turn on)

Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)

New CW: "ILP wall" law of diminishing returns on more HW for ILP

Old CW: Multiplies are slow, Memory access is fast

New CW: "Memory wall" Memory slow, multiplies fast
(200 clock cycles to DRAM memory, 4 clocks for multiply)

Old CW: Uniprocessor performance 2X / 1.5 yrs

New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall

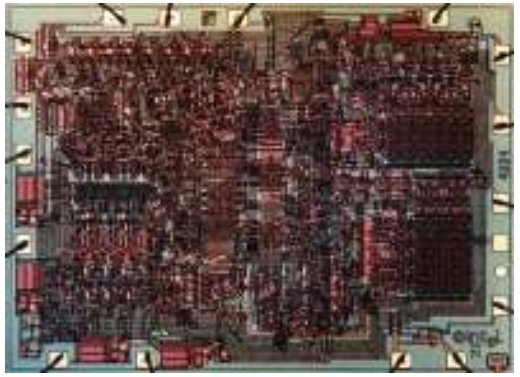
- Uniprocessor performance now 2X / 5(?) yrs

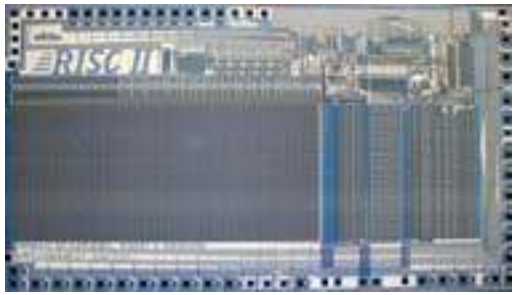
⇒ Sea change in chip design: multiple "cores"
(2X processors per chip / ~ 2 years)

- More simpler processors are more power efficient

Fonte: Doug L Hoffman, 2008

Sea Change in Chip Design





Intel 4004 (1971): 4-bit processor,
2312 transistors, 0.4 MHz,
10 micron PMOS, 11 mm^2 chip

- RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip

- 125 mm^2 chip, 0.065 micron CMOS = 2312 RISC II+FPU+Icache+Dcache
  - RISC II shrinks to ~ 0.02 mm^2 at 65 nm
  - Caches via DRAM or 1 transistor SRAM (www.t-ram.com) ?
  - Proximity Communication via capacitive coupling at > 1 TB/s ? (Ivan Sutherland @ Sun / Berkeley)

- **Processor is the new transistor?**

Adaptado dos slides da editora por Mario Côrtes – IC/Unicamp