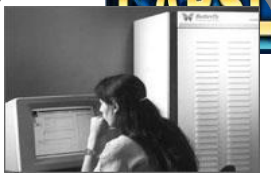
Introduction of Parallel Program Execution and Architecture Models

Guang R. Gao

ACM Fellow and IEEE Fellow
Endowed Distinguished Professor
Electrical & Computer Engineering
University of Delaware
ggao@capsl.udel.edu

Corporations Vanishing (1985 – 2005)

Myrias
1991



BBN
1997

Multiflow
1990

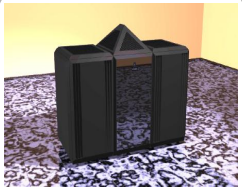
ESCD
1990

Convex
Computer
1994

Kendall Square
Resarch
1996

MasPar
1996

Cray Research
1996

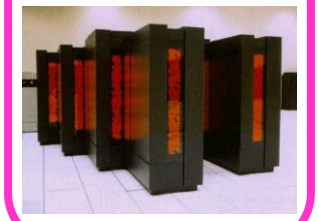


nCube
2005

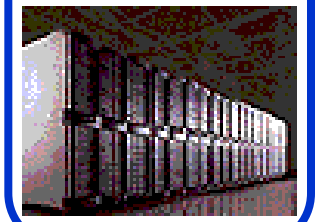
1989
ETA

1992
Meiko Scientific

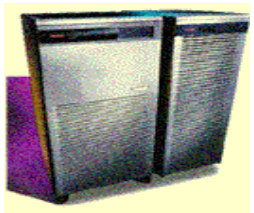
1994
Thinking Machines



1995
Pyramid



1998
DEC



1999
Sequent

From Gao: IPDPS 2005 Keynote

Outline

- A short history of parallel computing systems
- Program execution models (PXMs)
- Examples of parallel program execution models
  - SIMD and Vector processing model
  - MIMD model
  - Message-passing (e.g. MPI-style) execution model
  - Shared memory (e.g. OpenMP) execution model
  - ILP (instruction level parallelism) model
    - Superscalar model
    - VLIW model
- Evolution of (dynamic) fine-grain multithreaded program execution models.
- Summary

Recent Advances in Technologies

Memory

DDR2 SDRAM 2004

1.8V vs. 2.5V for DDR, Up to 8,533MB/s

DDR3 2007

1.5V or 1.35V for DDR3L, 6,400-17,066MB/s

DDR4 2012

1.05-1.2V, 2133-4266MT/s

Solid State Drives

Significant reduction in price per GB since mid-2000

More expensive than HDDs of comparable capacity

2007: *320GB 100k IOPS*

2009: *1TB SSD with 654MB/s write and 712MB/s read BW*

2009: *First SSD with 6Gbps SATA interface*

2011: *2.2TB, 2.7GB/s read bandwidth*

2012: *99.9% of 4KB random accesses within 0.5ms*

InfiniBand

Scalable switched fabric communications technology

High throughput and low latency

Point-to-point bidirectional serial links

Quality of service and failover features

2.5Gbps signaling rate in each direction per link in SDR speed

Since June '12 InfiniBand is the dominant class of interconnects in TOP500

Cray Interconnects

Seastar (Red Storm, Cray XT3), 2004

3-D mesh topology with link throughput of 2.5GB/s in each direction

SeaStar2 (Cray XT4), 2006

Peak bidirectional bandwidth per link: 7.6GB/s,

SeaStar2+ (Cray XT5), 2009

9.6GB/s peak bidirectional bandwidth per link

Gemini (Cray XE6, XK6, XK7), 2010

4 links per X and Z direction, 2 links per Y (10 total per NIC)

Aries/Dragonfly (Cray XC30), 2012

High radix tiled router (48 tiles), 8 processor tiles, 4 NICs

Ethernet

Gigabit Ethernet is the prevalent during last decade

Gained the lead in TOP500 system count in June 2005

Reaches the peak of 56.6% of all TOP500 systems in June 2008

The dominance continues until June 2012

Fast Ethernet 100Mbps

Gigabit Ethernet 1000Mbps

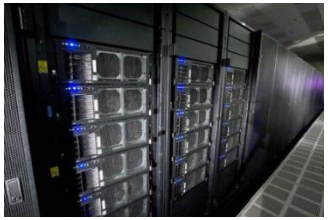
10Gigabit Ethernet 10Gbps

Courtesy of Prof. Thomas Sterling

World's Fastest Supercomputers

Japan, 2002: Earth Simulator
35.86 TFLOPS LINPACK

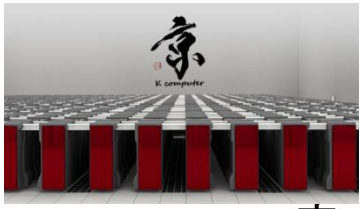
USA, 2004: Blue Gene/L
70.72 TFLOPS Linpack



USA, 2008: *Roadrunner*
1026 TFLOPS Linpack

USA, 2009: Jaguar
1759 TFLOPS Linpack

China 2010: Tianhe-1a
2566 TFLOPS Linpack



Japan 2011: K (京)
10510 TFLOPS Linpack

USA 2012: Sequoia
16324 TFLOPS Linpack

USA 2012: Titan
17590 TFLOPS Linpack

China 2013: Tianhe-2
33860 TFLOPS in HPL

Courtesy of Prof. T. Sterling

Tianhe-1A 2.566 Petaflops

Technology and Historical Perspective:

A peek of the microprocessor Evolution

Intel Pentium 4 Northwood

Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.

uOp Schedulers

FP Move Scheduler: (8x8 dependency matrix)

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

Load / Store uOp Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Integer Execution Core

(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle

(2) Integer Renamed Register File 128 entries of 32 bit + 6 status flags 12 read ports and six write ports

(3) Databus switch & Bypasses to and from the Integer Register File.

(4) Flags, Write Back

(5) Double Pumped ALU 0

(6) Double Pumped ALU 1

(7) Load Address Generator Unit

(8) Store Address Generator Unit

(9) Load Buffer (48 entries)

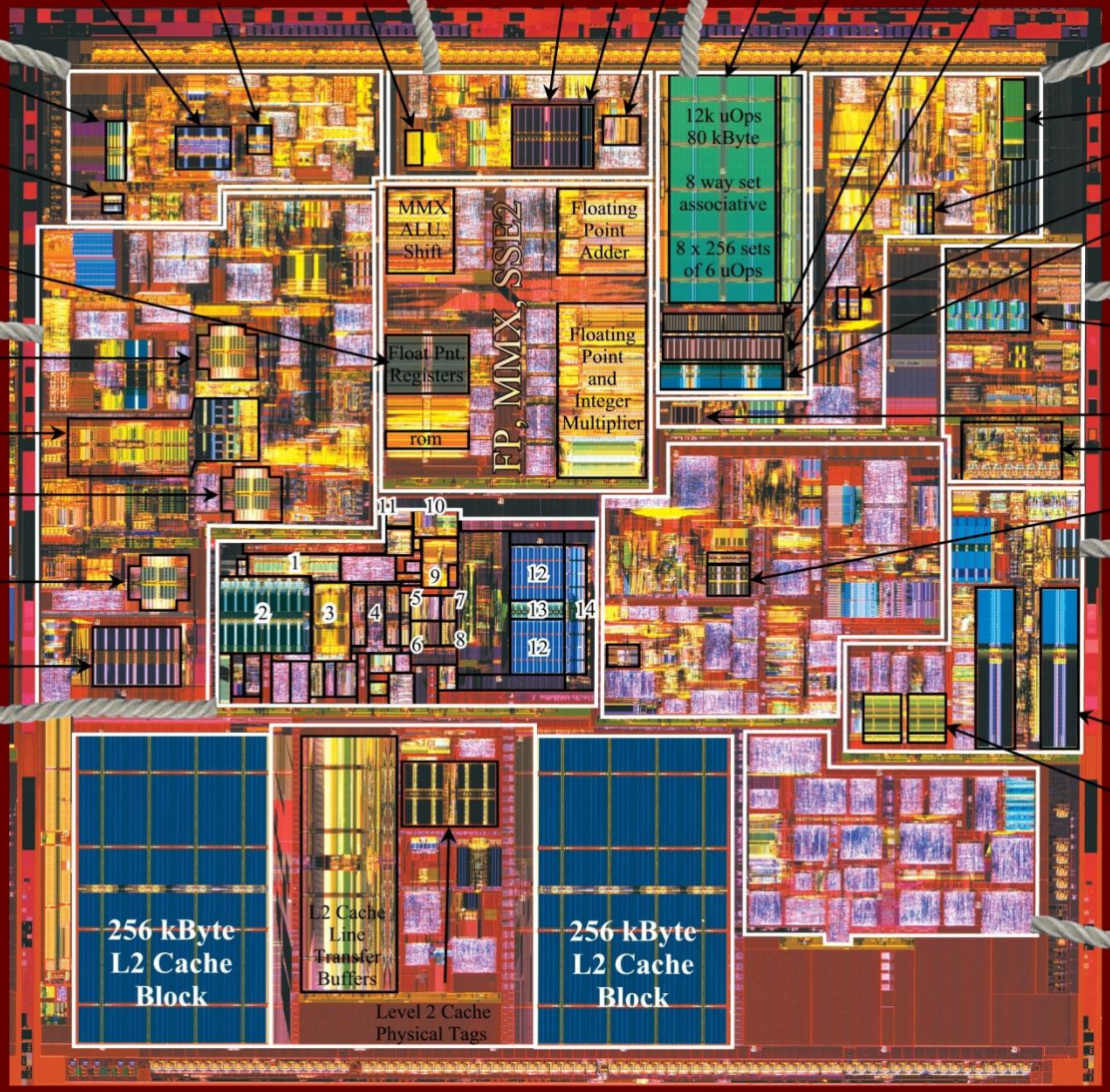
(10) Store Buffer (24 entries)

Execution Pipeline Start

Register Alias History Tables (2x126)
Register Alias Tables
uOp Queue

Instruction Trace Cache

Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags



Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 512 entries.

Return Stacks (2x16 entries)

Trace Cache next IP's (2x)

Miscellaneous Tag Data

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Trace Cache LRU bits

Raw Instruction Bytes in

Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

(11) ROB Reorder Buffer 3x42 entries

(12) 8 kByte Level 1 Data cache four way set associative. 1R/1W

(13) Summed Address Index decode and Way Predict

(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

April 19, 2003 www.chip-architect.com

Intel Pentium 5 Prescott

Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 1024 entries.

Return Stacks (4 x16 entries)

Trace Cache next IP's (4x)

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Raw Instruction Bytes in

Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 128 entry, fully associative for 4k and 4M pages. In: Virtual address [47:12] Out: Physical address [39:12] + 2 page level bits

Instruction Fetch from L2 cache and Branch Prediction

Front Side Bus Interface, 533..800 MHz

Instruction Trace Cache

Execution Pipeline Start

Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

uOp Schedulers

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

FP Move Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Load / Store uOp Scheduler: (8x8 dependency matrix)

FP, MMX, SSE1..3

Floating Point, MMX, SSE1..3 Renamed Register File 256 entries of 128 bit.

Integer Execution Core

(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle

(2) Integer Renamed Register File 256 entries of 32 bit (+ 6 status flags) 12 read ports and six write ports

(3) Databus switch & Bypasses to and from the Integer Register File.

(4) Flags, Write Back

(5) Double Pumped ALU 0

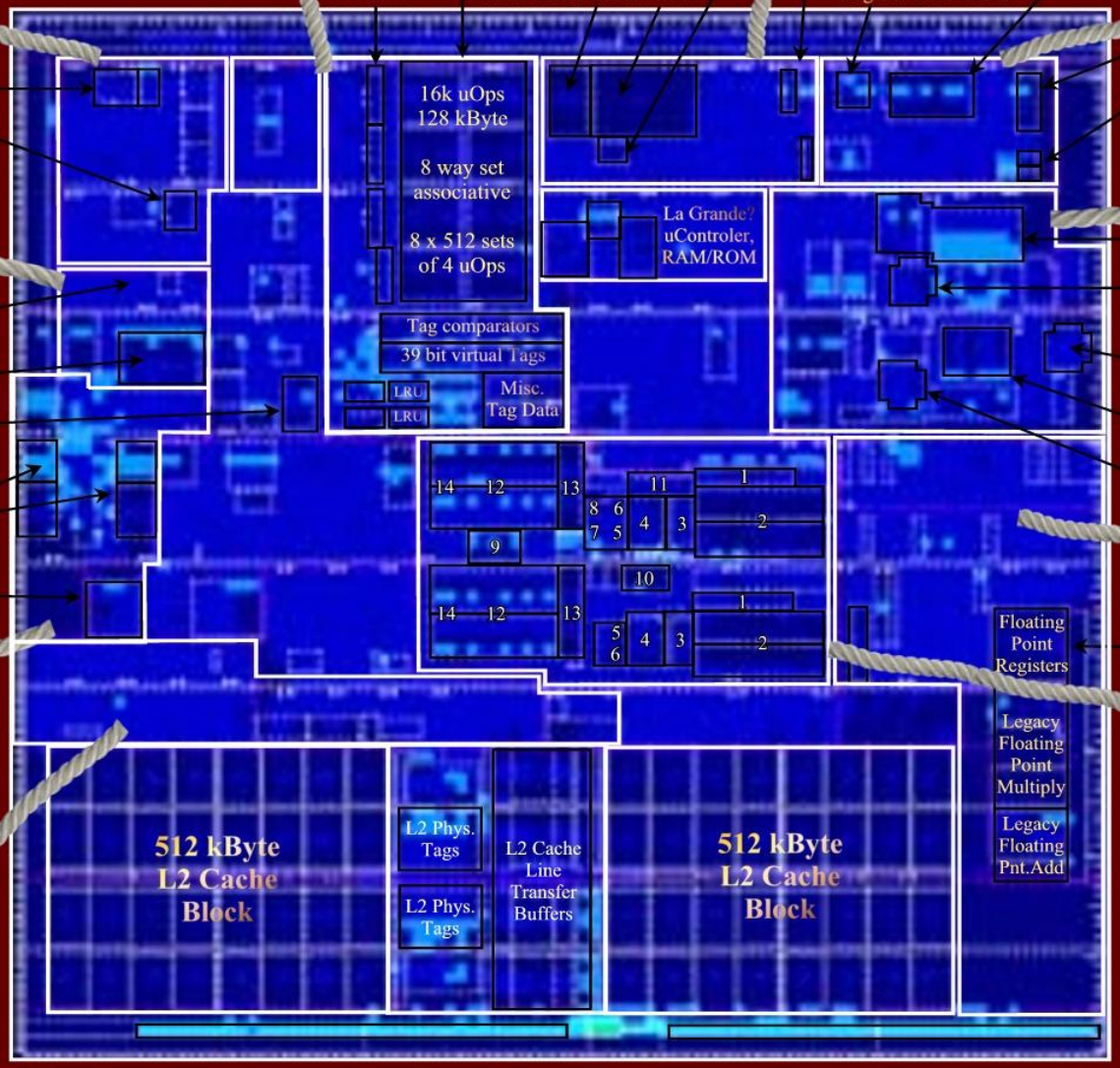
(6) Double Pumped ALU 1

(7) Load Address Generator Unit

(8) Store Address Generator Unit

(9) Load Buffer (96 entries)

(10) Store Buffer (48 entries)



(13) Databus multiplexing

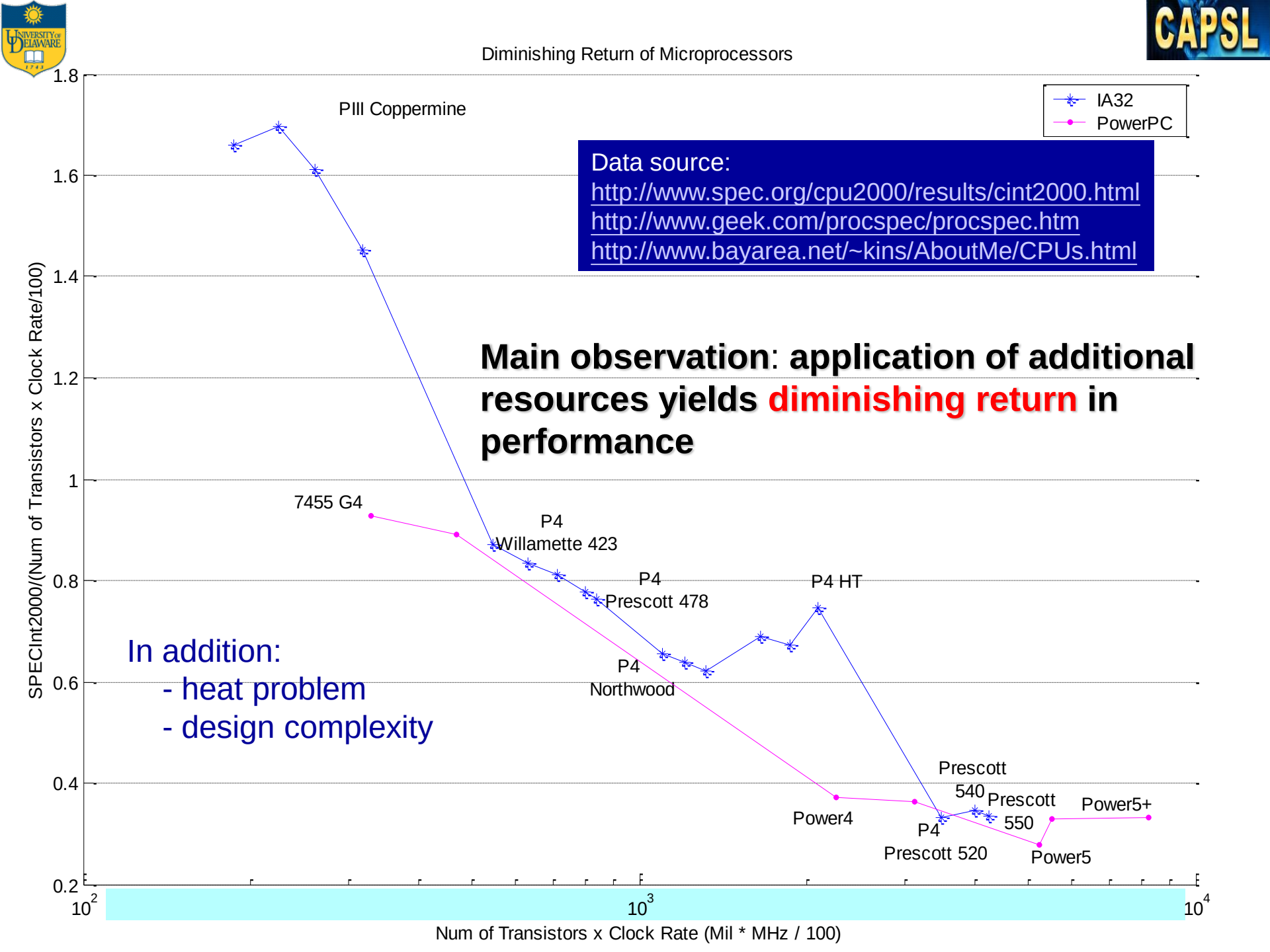
(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

(11) ROB Reorder Buffer 4x64 entries

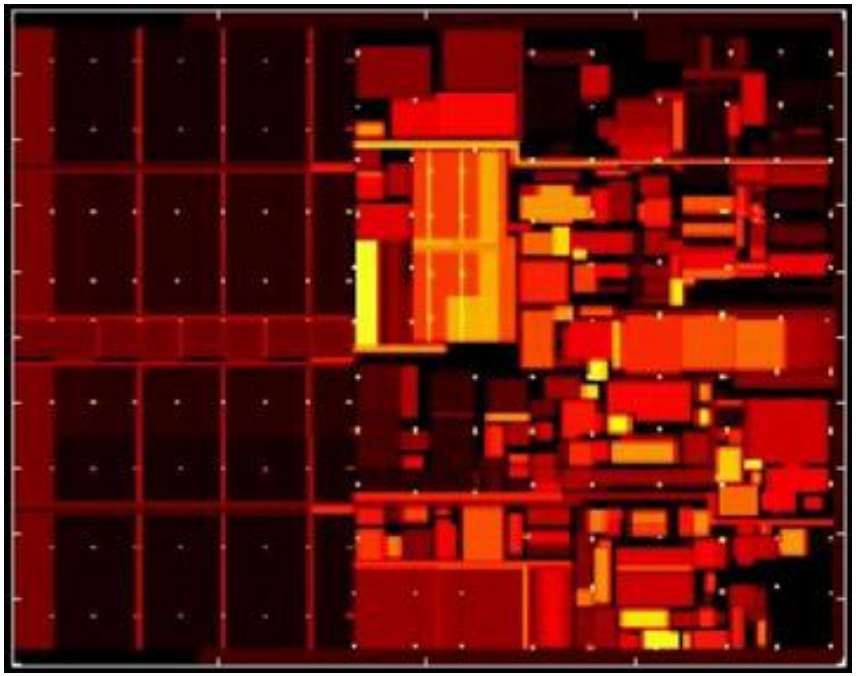
(12) 16 kByte Level 1 Data cache four way set associative. 1R/1W

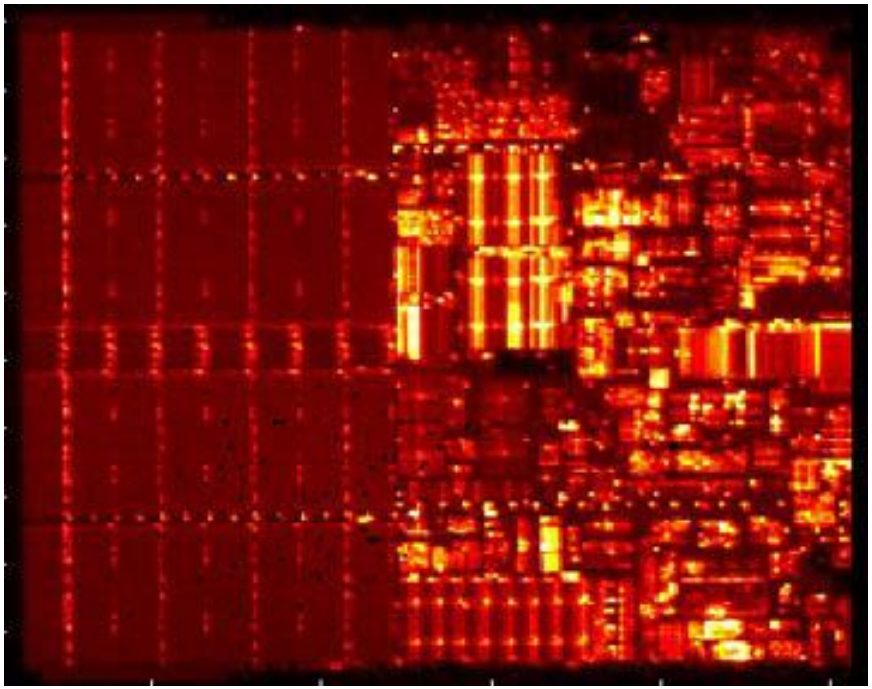
Technology Progress Overview

- Processor speed improvement: 2x per year (since 85). 100x in last decade.
- DRAM Memory Capacity: 2x in 2 years (since 96). 64x in last decade.
- DISK capacity: 2x per year (since 97). 250x in last decade.


Diminishing Return of Microprocessors
IA32
PowerPC
PIII Coppermine
Data source:
http://www.spec.org/cpu2000/results/cint2000.html
http://www.geek.com/procspec/procspec.htm
http://www.bayarea.net/~kins/AboutMe/CPUs.html
Main observation: application of additional resources yields diminishing return in performance
7455 G4
P4
Willamette 423
P4
Prescott 478
P4 HT
P4
Northwood
In addition:
- heat problem
- design complexity
Prescott
540
Prescott
550
Power5+
Power4
P4
Prescott 520
Power5
SPECInt2000/(Num of Transistors x Clock Rate/100)
Num of Transistors x Clock Rate (Mil * MHz / 100)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
10^2
10^3
10^4

Pentium M





Thermal Maps from the Pentium M obtained from simulated power density (left) and IREM measurement (right). Heat levels goes from black (lowest), red, orange, yellow and white (highest)

Figures courtesy of Dani Genossar and Nachum Shamir in their paper ***Intel ® Pentium ® M Processor Power Estimation, Bugdeting, Optimization and Validation*** *published in the Intel Technical Journal, May 21, 2003*

What Is Next ?

- Move to “multiprocessor on a chip” ?
  - cooler
  - simpler
  - cheaper
  - …

Architecture Features and Trends

(Revisitd)

- core arch- ***simpler and simpler : RISC Core***
- # of cores - ***larger and larger : 160 cores***
- on-chip memory per core - ***smaller and smaller :***

  < 32 KB/core
- On-chip bandwidth is becoming larger and larger :

  > 0.3 TB/sec
- Energy efficiency support - more and more :

  500 MHz

Outline

- A short history of parallel computing systems
- Program execution models (PXM)
- Examples of program execution models
  - Sequential execution model
  - Parallel execution model
    - SIMD and Vector processing model
    - MIMD model
    - Message-passing (e.g. MPI-style) execution model
    - Shared memory (e.g. OpenMP) execution model
    - ILP (instruction level parallelism) model
      - Superscalar model
      - VLIW model
- Evolution of (dynamic) fine-grain multithreaded program execution models.
- Summary

What is a Program Execution Model?

User Code

- Application Code
- Software Packages
- Program Libraries
- Compilers
- Utility Applications

PXM (API)

System

- Hardware
- Runtime Code
- Operating System

Features a User Program Depends On

Features expressed within a Programming language

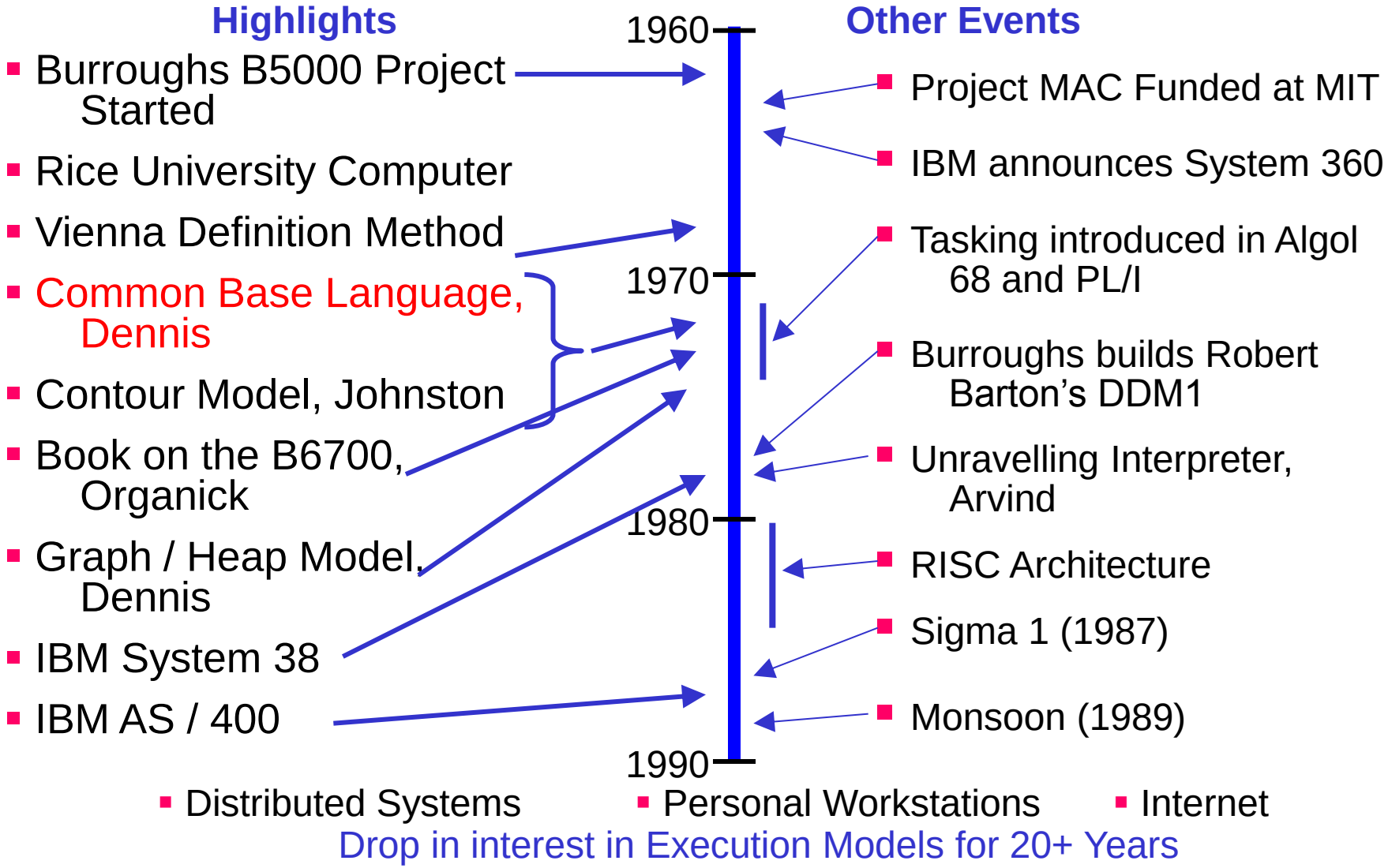
- Procedures; call/return
- Access to parameters and variables
- Use of data structures (static and dynamic)

But that's not all !!

Features expressed Outside a (typical) programming language

- File creation, naming and access
- Object directories
- Communication: networks and peripherals
- Concurrency: coordination; scheduling

Developments in the 1960s, 1970s



Contour Model: Algorithm; Nested Blocks and Contours

- Johnston, 1971

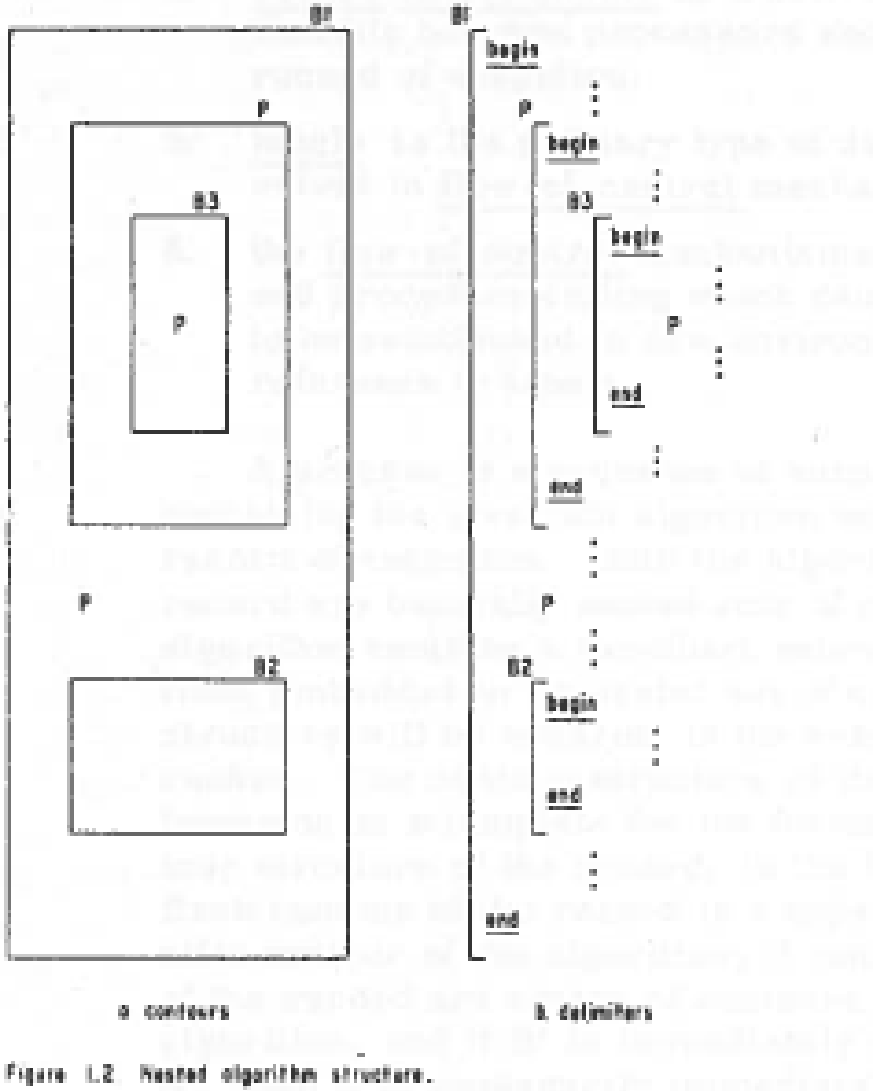


Figure 1.2 Nested algorithm structure.

Idea: A Common Base Language

This is a report on the work of the Computation Structures Group of Project MAC toward the design of a common base language for programs and information structures. We envision that the meanings of programs expressed in practical source languages will be defined by rules of translation into the base language.

The meanings of programs in the base language is fixed by rules of interpretation which constitute a transition system called the interpreter for the base language.

We view the base language as the functional specification of a computer system in which emphasis is placed on programming generality -- the ability of users to build complex programs by combining independently written program modules.

- Dennis, 1972

Terminology Clarification

- Parallel Model of Computation
  - Parallel Models for Algorithm Designers
  - Parallel Models for System Designers
    - Parallel Programming Models
    - Parallel Execution Models
    - Parallel Architecture Models

What Does Program Execution Model (PXM) Mean ?

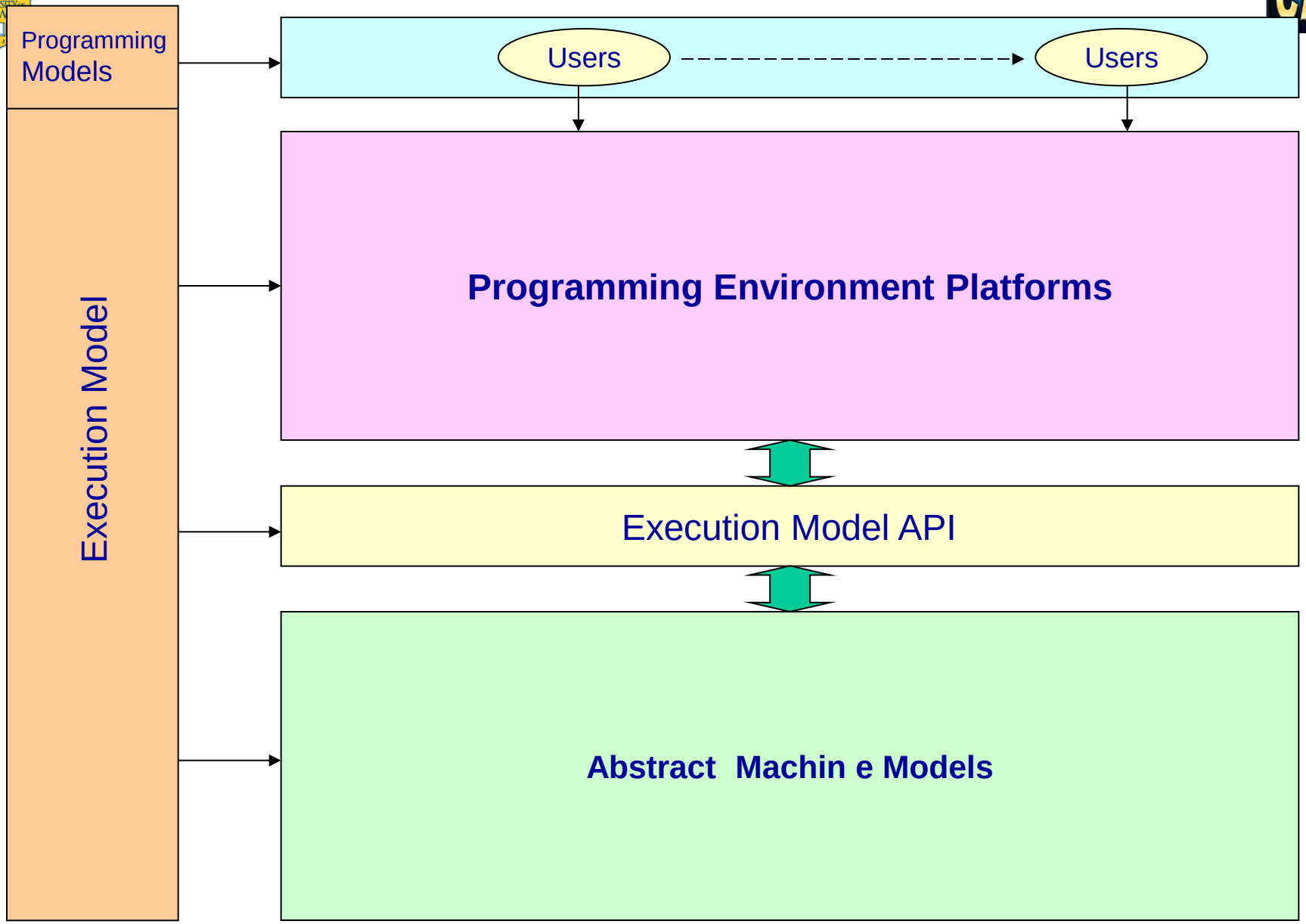
- The notion of PXM

  The program execution model (PXM) is the basic low-level abstraction of the underlying system architecture upon which our programming model, compilation strategy, runtime system, and other software components are developed.
- The PXM (and its API) serves as *an interface between the architecture and the software.*

Program Execution Model (PXM) – Cont'd

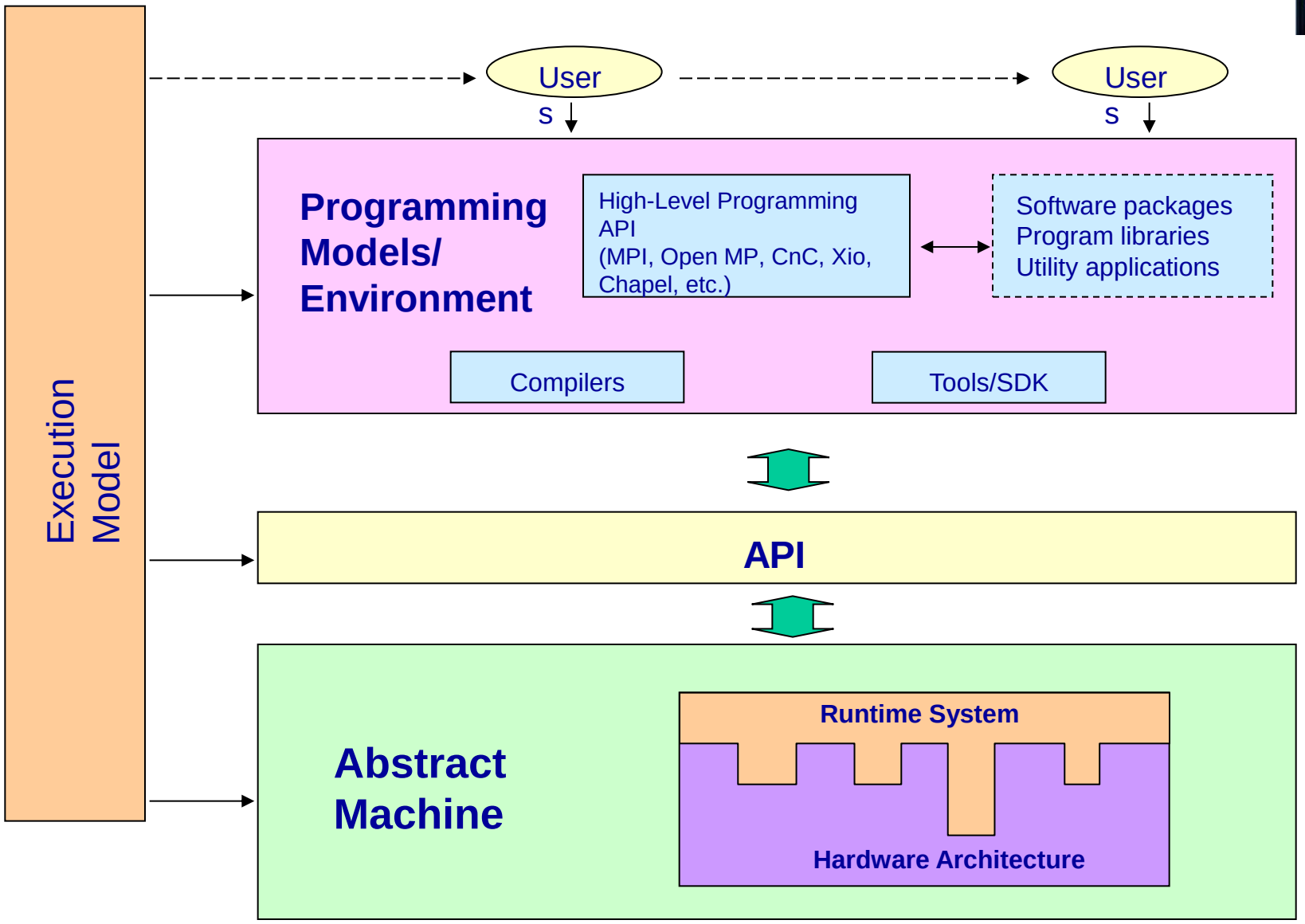
Unlike an instruction set architecture (ISA) specification, which usually focuses on lower level details (such as instruction encoding and organization of registers for a specific processor), the PXM refers to machine organization at a higher level for a whole class of high-end machines as view by the users

Gao, et. al., 2000



Execution Model and Abstract Machines

Abstract Machine Models May Be Heterogeneous!



Execution Model and Abstract Machines

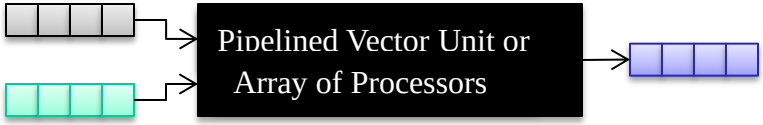
Outline

- A short history of parallel computing systems
- Program execution models (PXM)
- Examples of program execution models
  - Sequential execution model
  - Parallel execution model
    - SIMD and Vector processing model
    - MIMD model
    - Message-passing (e.g. MPI-style) execution model
    - Shared memory (e.g. OpenMP) execution model
    - ILP (instruction level parallelism) model
      - Superscalar model
      - VLIW model
- Evolution of (dynamic) fine-grain multithreaded program execution models.
- Summary

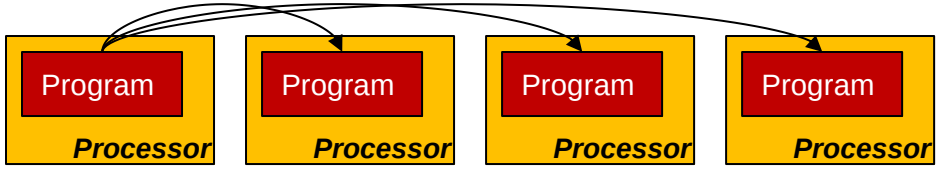
What is your "Favorite" Program Execution Model?

Course Grain Execution Models

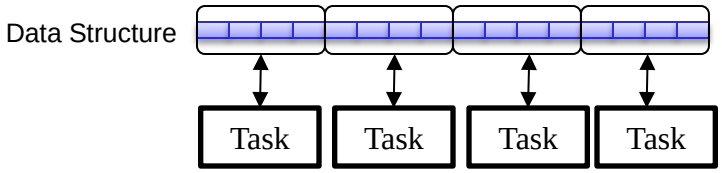
The Single Instruction Multiple Data (SIMD) Model



The Single Program Multiple Data (SPMD) Model



The Data Parallel Model



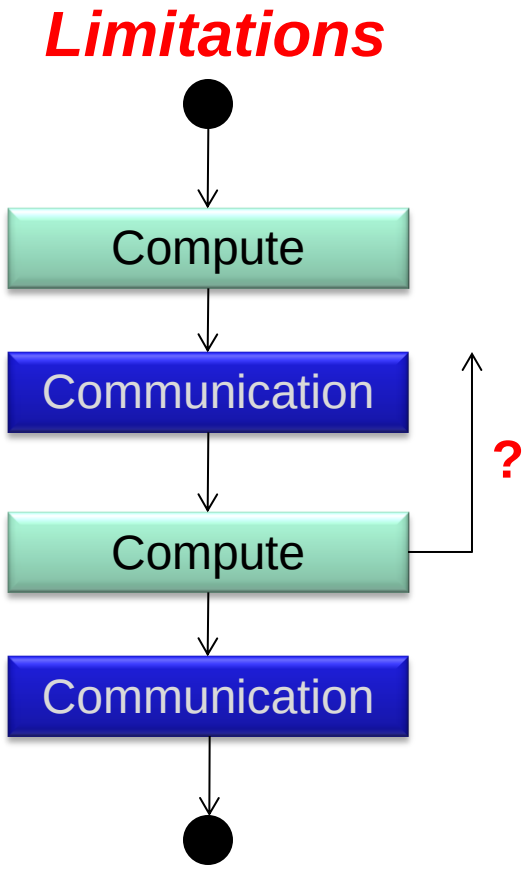
Data Parallel Model

Difficult to write unstructured programs

Convenient only for problems with regular structured parallelism.

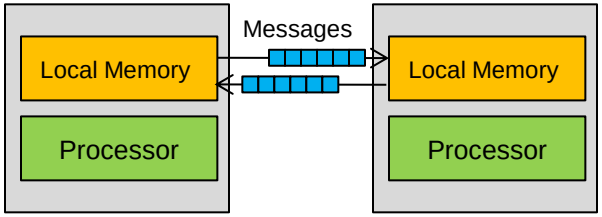
Limited composability!

Inherent limitation of coarse-grain multi-threading

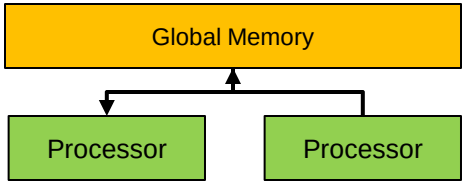


Programming Models for Multi-Processor Systems

- Message Passing Model
  - Multiple address spaces
  - Communication can only be achieved through ***"messages"***



- Shared Memory Model
  - Memory address space is accessible to all
  - Communication is achieved through memory



Comparison

Message Passing

+ Less Contention
+ Highly Scalable
+ Simplified Synch
  - Message Passing ➔ Sync + Comm.
  - But does not mean highly programmable
- Load Balancing
- Deadlock prone
- Overhead of small messages

Shared Memory

+ global shared address space
+ Easy to program (?)
+ No (explicit) message passing (e.g. communication through memory put/get operations)
- Synchronization (memory consistency models, cache models)
- Scalability

Comment on OS impact?

- Should compiler be OS-Aware too ? If so, how ?
- Or other alternatives ? Compiler-controlled runtime, of compiler-aware kernels, etc.
- Example: software pipelining …

Gao, ECCD Workshop, Washington D.C., Nov. 2007

Outline

- A short history of parallel computing systems
- Examples of program execution models
  - Sequential execution model
  - Parallel execution model
    - SIMD and Vector processing model
    - MIMD model
    - Message-passing (e.g. MPI-style) execution model
    - Shared memory (e.g. OpenMP) execution model
    - ILP (instruction level parallelism) model
      - Superscalar model
      - VLIW model
- Evolution of (dynamic) fine-grain multithreaded program execution models.
- What is program execution model anyway ?
- Summary

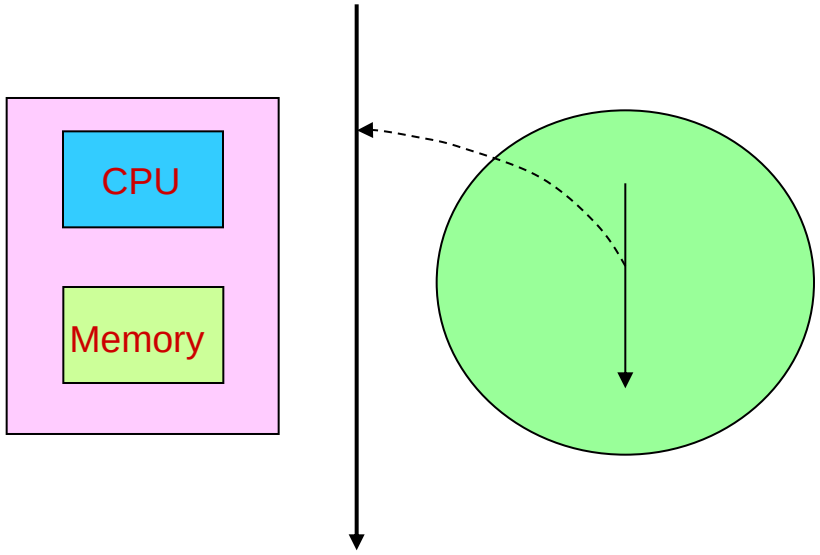
A Quiz: Have you heard the following terms ?

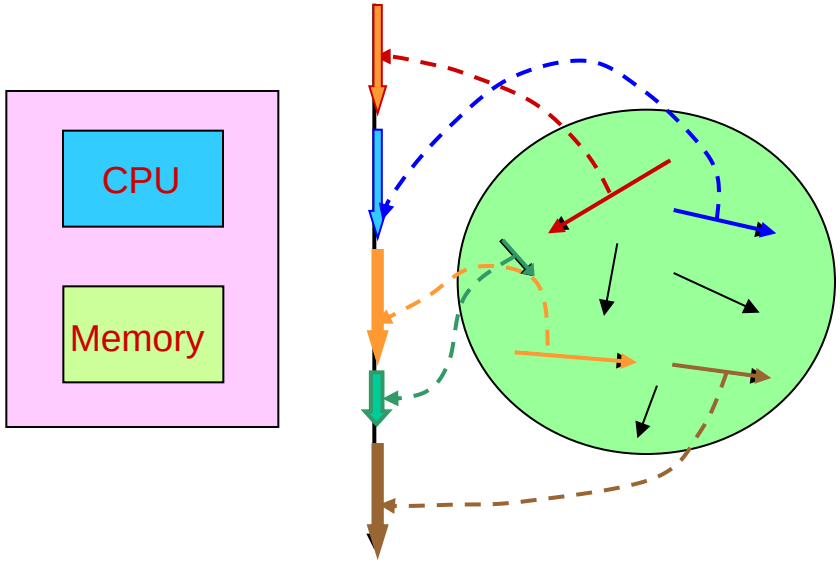
Actors (dataflow) ?

strand ?

fiber ?

codelet ?



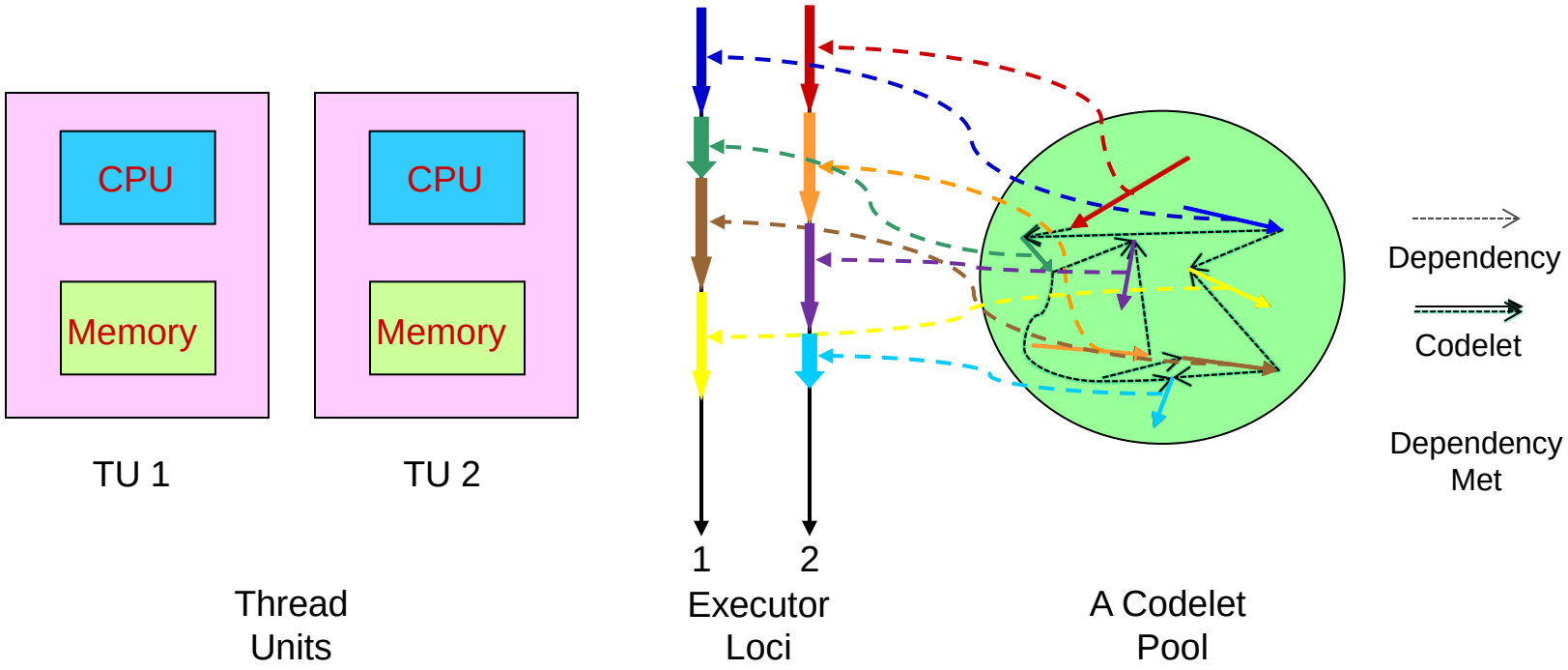


Coarse-Grain thread-
The family home model

Fine-Grain *non-preemptive* thread-
The "hotel" model

Coarse-Grain vs. Fine-Grain Multithreading

[Gao: invited talk at Fran Allen's Retirement Workshop, 07/2002]

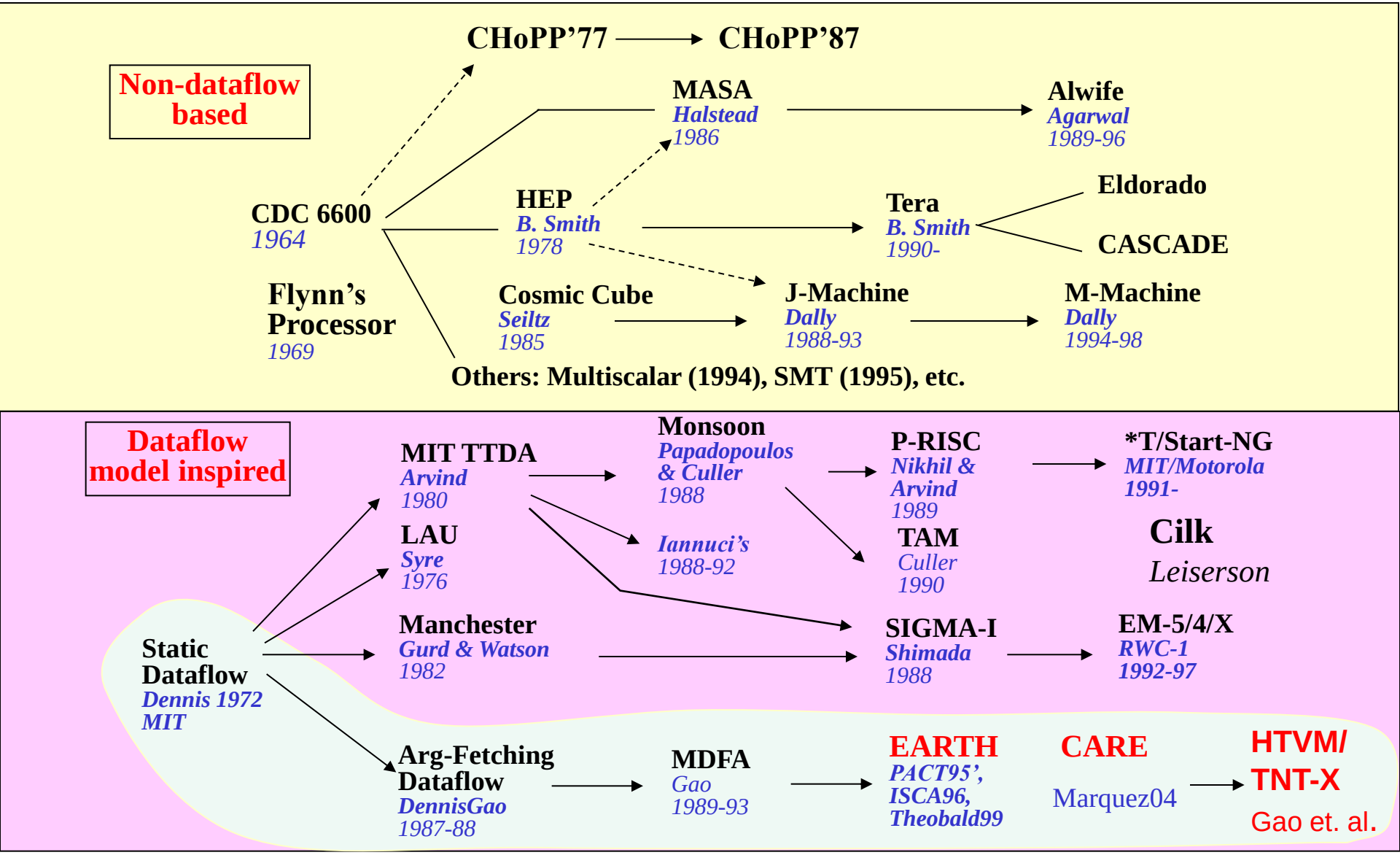


Fine-Grain *non-preemptive* thread-
The "hotel" model

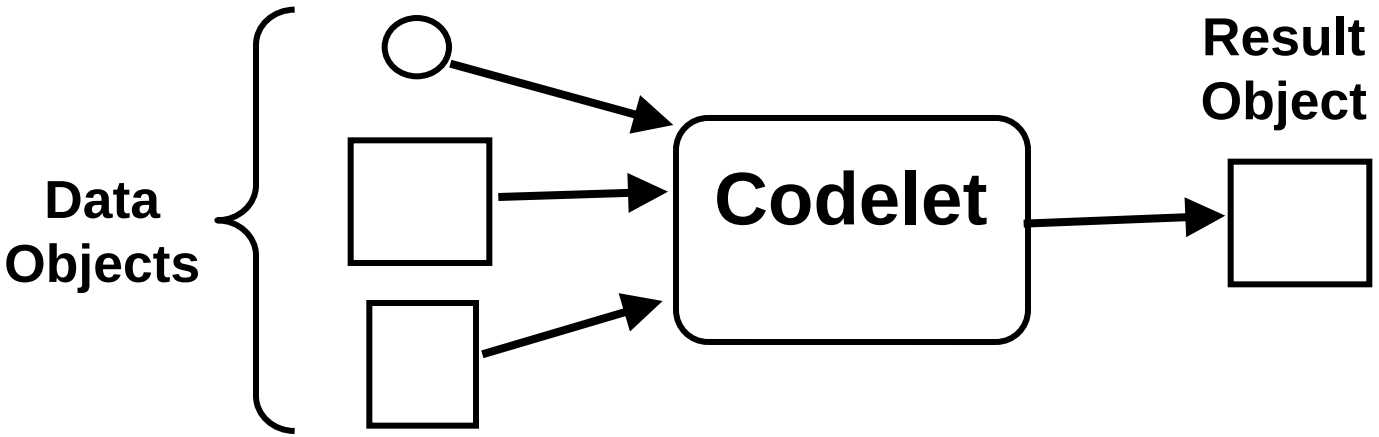
Coarse-Grain vs. Fine-Grain Multithreading

[Gao: invited talk at Fran Allen's Retirement Workshop, 07/2002]

Evolution of Multithreaded Execution and Architecture Models



The Codelet: A Fine-Grain Piece of Computing



Supports Massively Parallel Computation!

Outline

- A short history of parallel computing systems
- Examples of program execution models
  - Sequential execution model
  - Parallel execution model
    - SIMD and Vector processing model
    - MIMD model
    - Message-passing (e.g. MPI-style) execution model
    - Shared memory (e.g. OpenMP) execution model
    - ILP (instruction level parallelism) model
      - Superscalar model
      - VLIW model
- Evolution of (dynamic) fine-grain multithreaded program execution models.
- What is program execution model anyway ?
- Summary