

Parallel Programming

Parallelism

Jan Schönherr

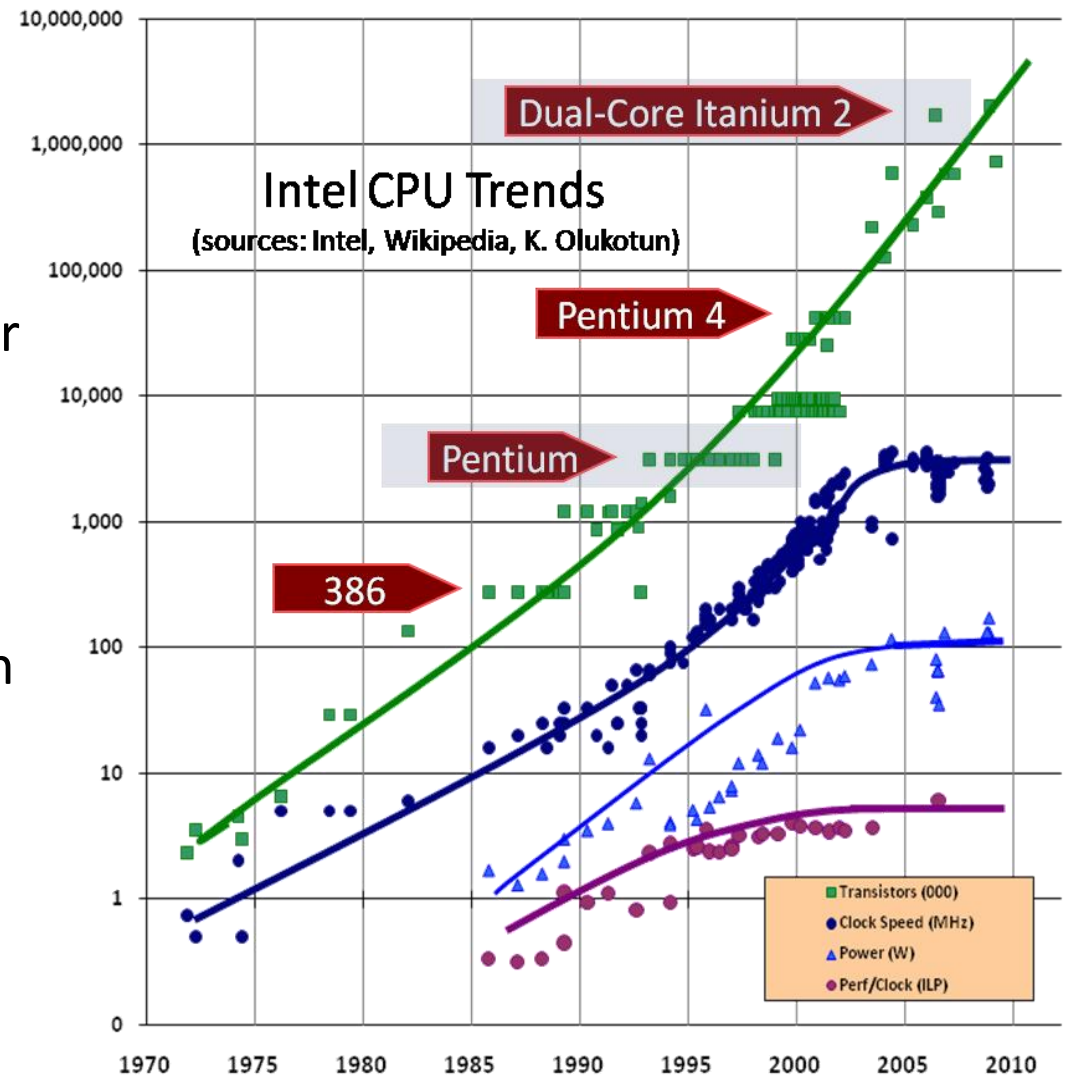
Technische Universität Berlin
School IV – Electrical Engineering and Computer Sciences
Communication and Operating Systems (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Why Parallelism?

- > High performance computing
 - > Solve large problems faster
(or be able to solve them in time at all)
 - > Solve larger problems in the same time
- > Enterprise computing
 - > Increase throughput
- > Computing in general
 - > Satisfy an increasing performance demand while avoiding physical limitations (frequency, power, ...)
 - > Save money (lots of small chips are cheaper than a complex one)
 - > Save energy (small or specialized chips are more energy efficient for certain tasks)

Limits in Processors

- > Power Consumption
 - > Limited to about 150W for air cooling
 - > Higher frequency needs higher core voltages -> more power
- > Memory
 - > Upon a cache miss, execution might stall (despite Instruction Level Parallelism)
- > Instruction Level Parallelism
 - > It is hard to extract more ILP out of (general) programs



Source: The Free Lunch Is Over, H. Sutter

Power Wall

- > Power consumption of a processor

$$P_{dynamic} \sim \alpha C \cdot f \cdot V^2$$

$$P_{static} \sim V \cdot I_{leak}$$

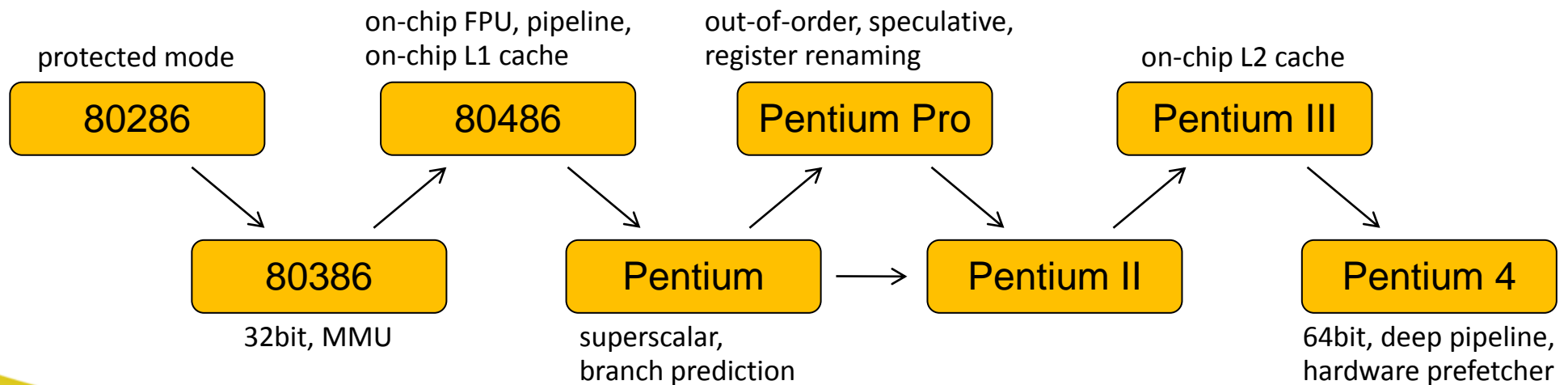
- > Frequency and voltage are not independent
 - > Transistor switching speed depends on voltage
 - > Critical path must settle during a clock cycle
- > Lower frequencies are more energy efficient

Memory Wall

- > Memory is slow
 - > E. g. a read latency of up to 190 processor cycles on an Intel Xeon X5570
- > Various efforts to hide that
 - > Multi-level cache hierarchies
 - > Lower latencies (e. g. 4 cycles for L1, 10 for L2, 38 for L3 cache)
 - > Memory prefetching
 - > Out-of-order execution
 - > Execute something else while waiting
- > Still, the performance of many tasks does not scale linearly with the processor frequency

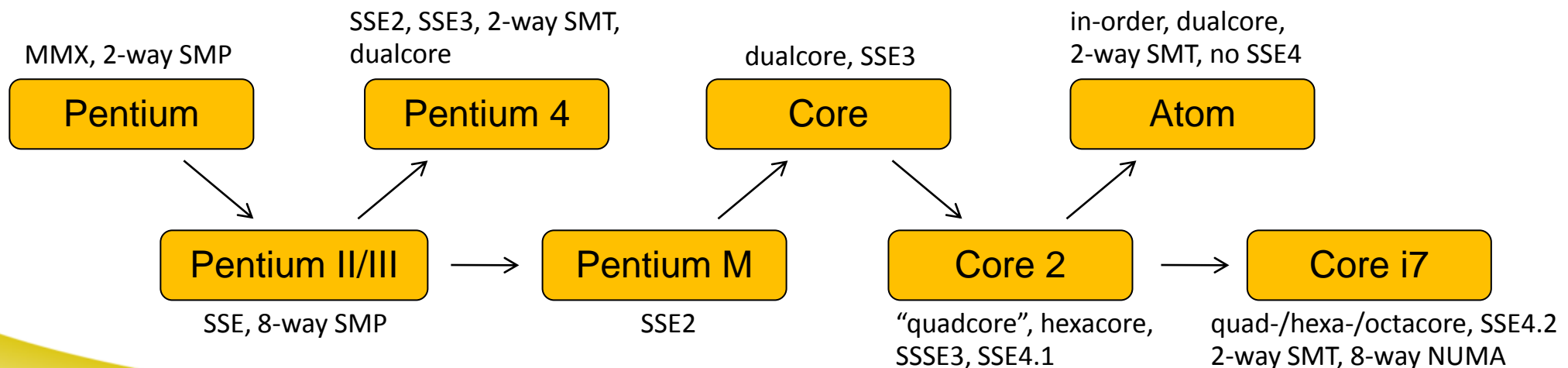
Historical Processor Development (i)

- > Starting with (relatively) simple processors
- > Increase clock frequency
- > Increase IPC (Instructions Per Cycle)
 - > By adding on-chip memory
 - > By increasing ILP (Instruction Level Parallelism)
- > Increase word size (Bit Level Parallelism)



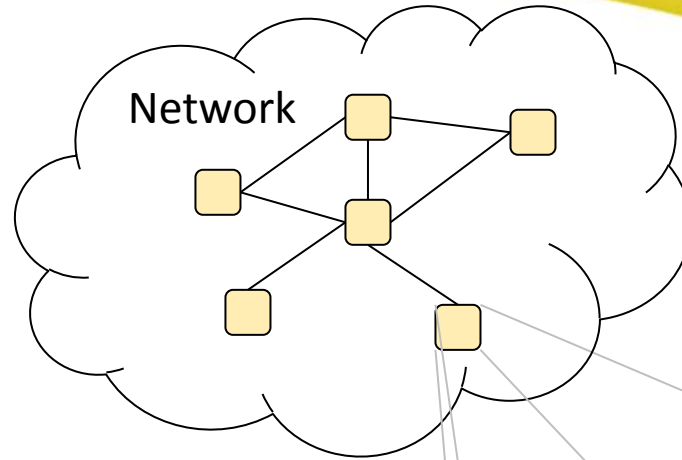
Historical Processor Development (ii)

- > Parallelism was introduced at various locations in various combinations
 - > Multisocket systems (first SMP, later NUMA architectures)
 - > Multicore processors
 - > Simultaneous Multithreading (SMT)
 - > SIMD units

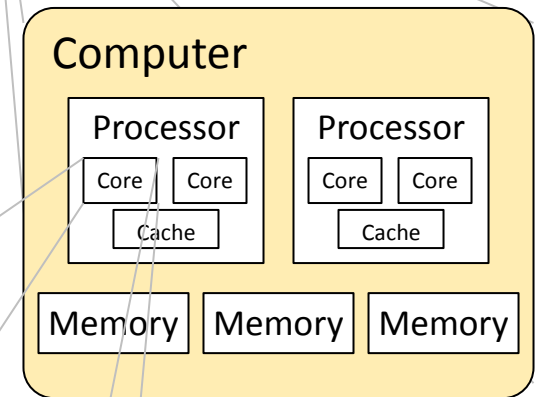


Typical Hardware

Network of Computers
(Cluster, Grid, Internet, ...)

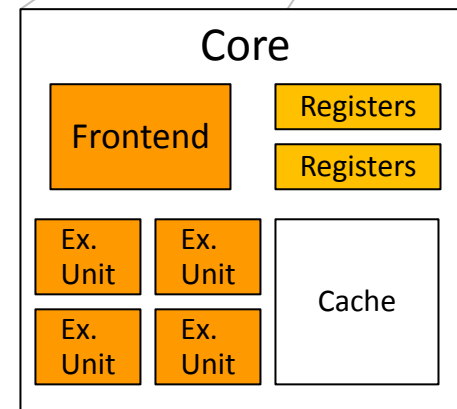


Computer
(Parallel, Server, Desktop, Mobile, Embedded, ...)



Processor
(CPU, GPU, ...)

Core
(single threaded/SMT, in-order/out-of-order, ...)



Execution Unit
(Integer, Floating Point, SIMD, ...)

Bits

Levels of Parallelism

- > Bit level parallelism
 - > Parallel processing of bits of a word
- > Instruction level parallelism
 - > Parallel processing of instructions of the same task
- > Data level parallelism
 - > Identical operations on sets of data
- > Task level parallelism
 - > Independent blocks of operations

Typical “Desktop” Software

- > Mostly single threaded software
 - > Threads only to provide responsiveness
- > Seldom optimized for target architecture
- > Only special purpose software is aware of the available parallelism, e. g.
 - > Multimedia applications
 - > Number crunching applications

Parallelism in Sequential Software

- > Sequential software is already (a little bit) parallel
 - > Realized by compiler and/or hardware
- > Bit Level Parallelism
- > Instruction Level Parallelism (ILP)
 - > To increase instruction throughput
 - > Many different techniques
 - > Pipelining, superscalar processors, out-of-order execution, speculative execution, ...
- > Data Level Parallelism
 - > SIMD instructions to speed up some applications

Parallelism in Parallel Programs

- > Parallel programs exploit more parallelism than sequential programs!
- > Data Level Parallelism
 - > Identical/similar operations on data structures are captured on a larger scale and carried out in parallel.
- > Task Level Parallelism
 - > Independent blocks of operations are carried out in parallel.

New Problems with Parallelism

- > You have to care about
 - > Problem decomposition
 - > Coordination of interacting processes/threads
 - > Correctness, deadlocks, race conditions
 - > Via shared memory or message passing
 - > Scalability
 - > Environment
 - > Interconnection network (latency, bandwidth, topology)
 - > Heterogeneity
 - > Exclusiveness

Means to create Parallel Programs

- > Parallelizing compilers
 - > Pro: compile sequential code without changes into a parallel program
 - > Con: very hard to do efficiently, limited applicability
- > Parallel programming languages
 - > Pro: easy to exploit parallelism
 - > Con: low acceptance, compiler support needed
- > Augmented sequential programming languages
 - > Pro: higher acceptance
 - > Con: also needs compiler support
- > Libraries enabling parallelism
 - > Pro: high acceptance, no compiler support needed
 - > Con: tends to be a little complex

Parallel Programming Environments from the 1990s



"C* in C	COOL	GAMMA	Mirage	ParLib++	SHMEM
ABCPL	CORRELATE	Glenda	Modula-2*	ParLin	SIMPLE
ACE	CparPar	GLU	Modula-P	Parlog	Sina
ACT++	CPS	GUARD	MOSIX	Parmacs	SISAL
ADDAP	CRL	HASL	MpC	Parti	SMI
Adl	CSP	HORUS	MPC++	pC	SONiC
Adsmith	Cthreads	HPC	MPI	pC++	Split-C
AFAPI	CUMULVS	HPC++	Multipol	PCN	SR
ALWAN	DAGGER	HPF	Munin	PCP:	Sthreads
AM	DAPPLE	IMPACT	Nano-Threads	PCU	Strand
AMDC	Data Parallel C	ISETL-Linda	NESL	PEACE	SUIF
Amoeba	DC++	ISIS	NetClasses++	PENNY	SuperPascal
AppLeS	DCE++	JADA	Nexus	PET	Synergy
ARTS	DDD	JADE	Nimrod	PETSc	TCGMSG
Athapscan-Ob	DICE	Java RMI	NOW	PH	Telegraphos
Aurora	DIPC	javaPG	Objective Linda	Phosphorus	The FORCE
Automap	Distributed Smalltalk	JAVAR	Occam	POET	Threads.h++
bb_threads	DOLIB	JavaSpaces	Omega	Polaris	TRAPPER
Blaze	DOME	JIDL	OOF90	POOL-T	TreadMarks
BlockComm	DOSMOS	Joyce	Orca	POOMA	UC
BSP	DRL	Karma	P++	POSYBL	uC++
C*	DSM-Threads	Khoros	P-RIO	PRESTO	UNITY
C**	Ease	KOAN/Fortran-S	P3L	Prospero	V
C4	ECO	LAM	P4-Linda	Proteus	ViC*
CarLOS	Eilean	Legion	Pablo	PSDM	Visifold V-NUS
Cashmere	Emerald	Lilac	PADE	PSI	VPE
CC++	EPL	Linda	PADRE	PVM	Win32 threads
Charlotte	Excalibur	LiPS	Panda	QPC++	WinPar
Charm	Express	Locust	Papers	Quake	WWWinda
Charm++	Falcon	Lparx	Para++	Quark	XENOOPS
Chu	Filaments	Lucid	Paradigm	Quick Threads	XPC
Cid	FLASH	Maisie	Parafrase2	Sage++	Zounds
Cilk	FM	Manifold	Paralation	SAM	ZPL
CM-Fortran	Fork	Mentat	Parallaxis	SCANDAL	
Code	Fortran-M	Meta Chaos	Parallel Haskell	SCHEDULE	
Concurrent ML	FX	Midway	Parallel-C++	SciTL	
Converse	GA	Millipede	ParC	SDDA	

From *Patterns for Parallel Programming*, page 14

Today's Parallel Programming Environments – a subjective list

- > Clusters
 - > Established: MPI

- > Multi-core systems
 - > Established: explicit threading
 - > Java, C#, pthreads, Win32 threads, ...
 - > Upcoming: OpenMP

- > GPUs
 - > Established: CUDA
 - > Upcoming: OpenCL