Parallel Programming

Parallelism

Jan Schönherr

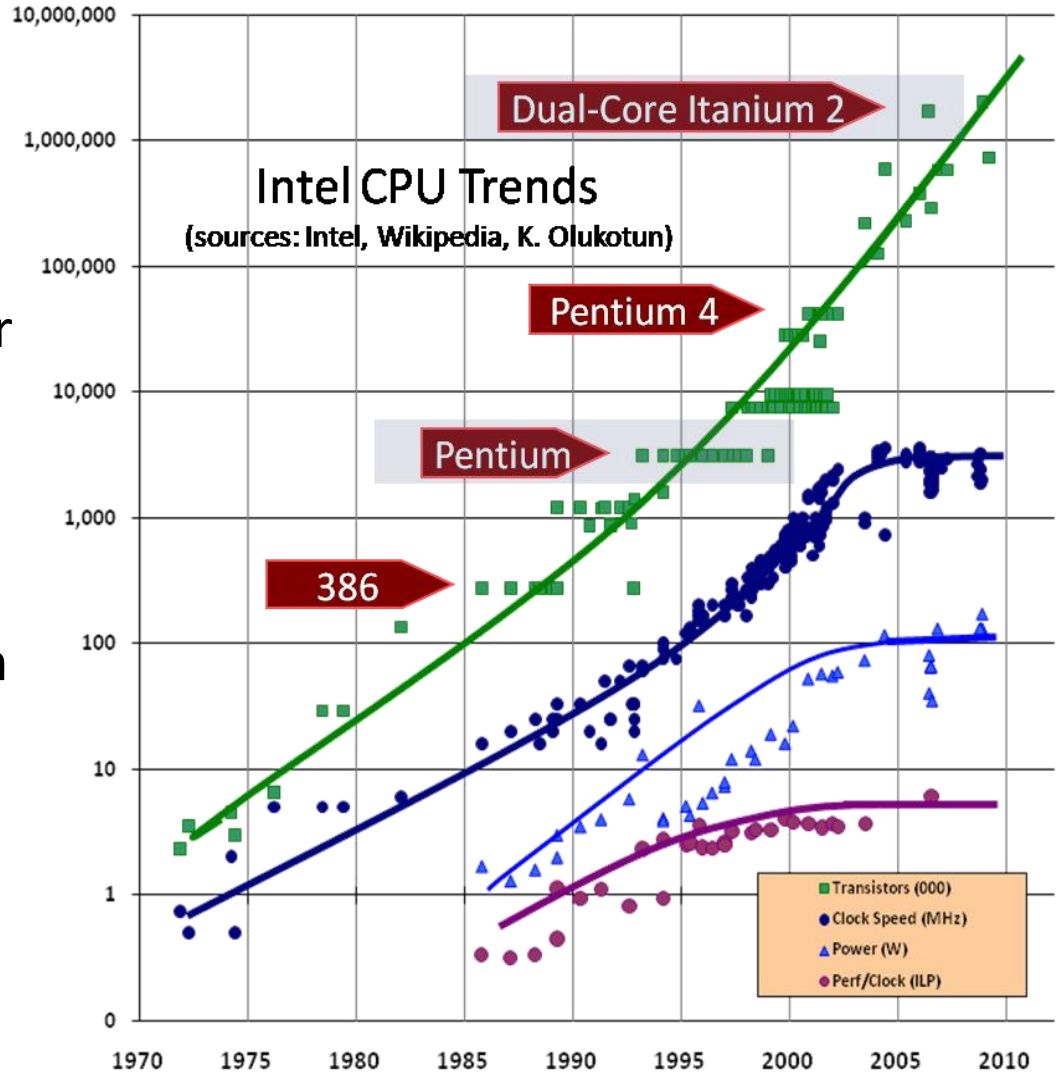
Technische Universität Berlin
School IV – Electrical Engineering and Computer Sciences
Communication and Operating Systems (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

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

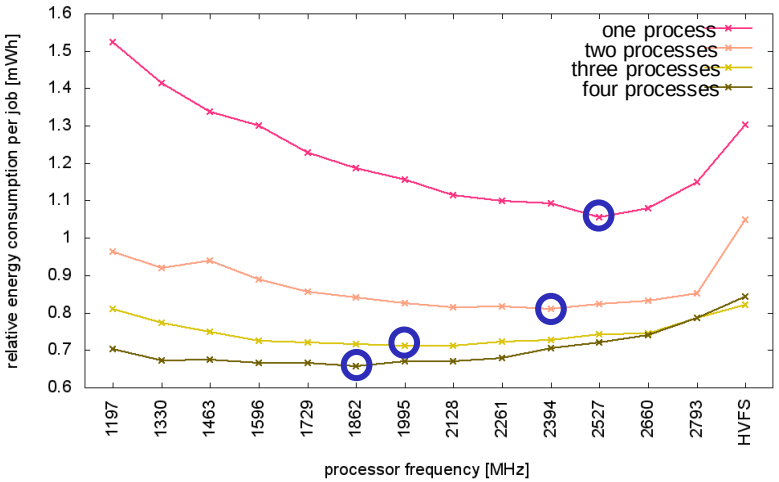
- Power consumption of a processor

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

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

- 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
  - (until influence of static power consumption becomes too large)



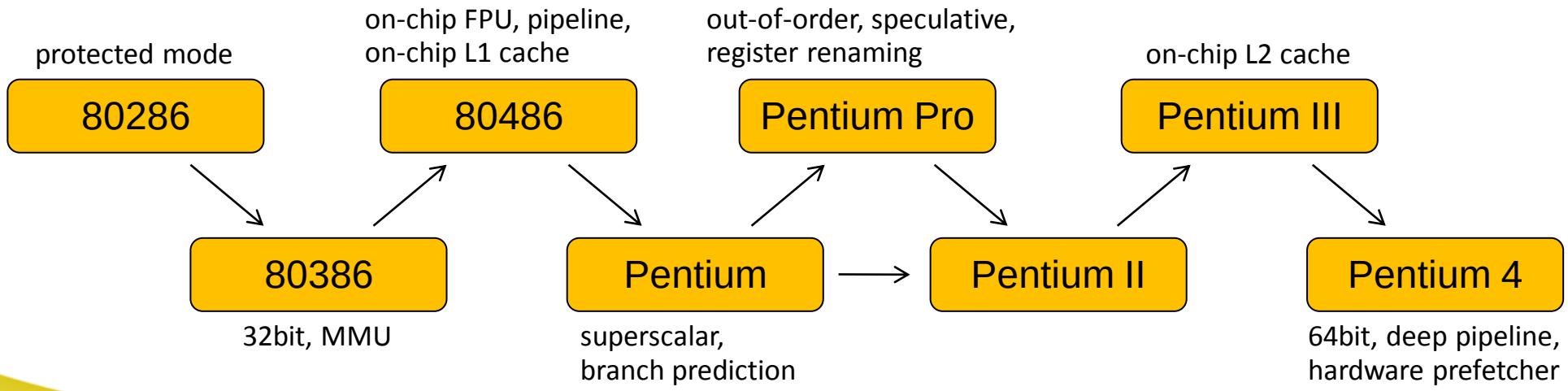
Memory Wall

- 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)

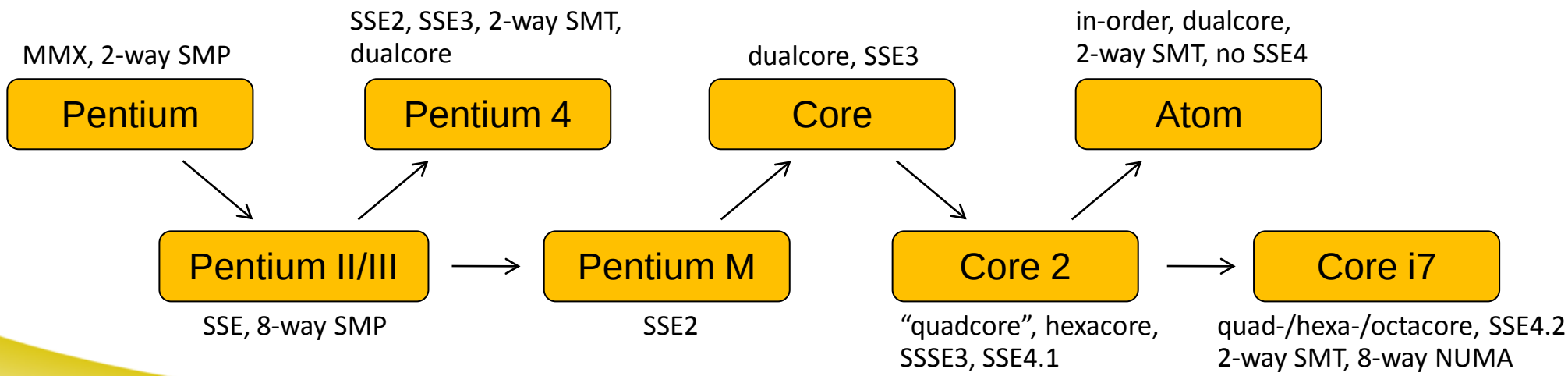
- Starting with (relatively) simple processors

- 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, …)

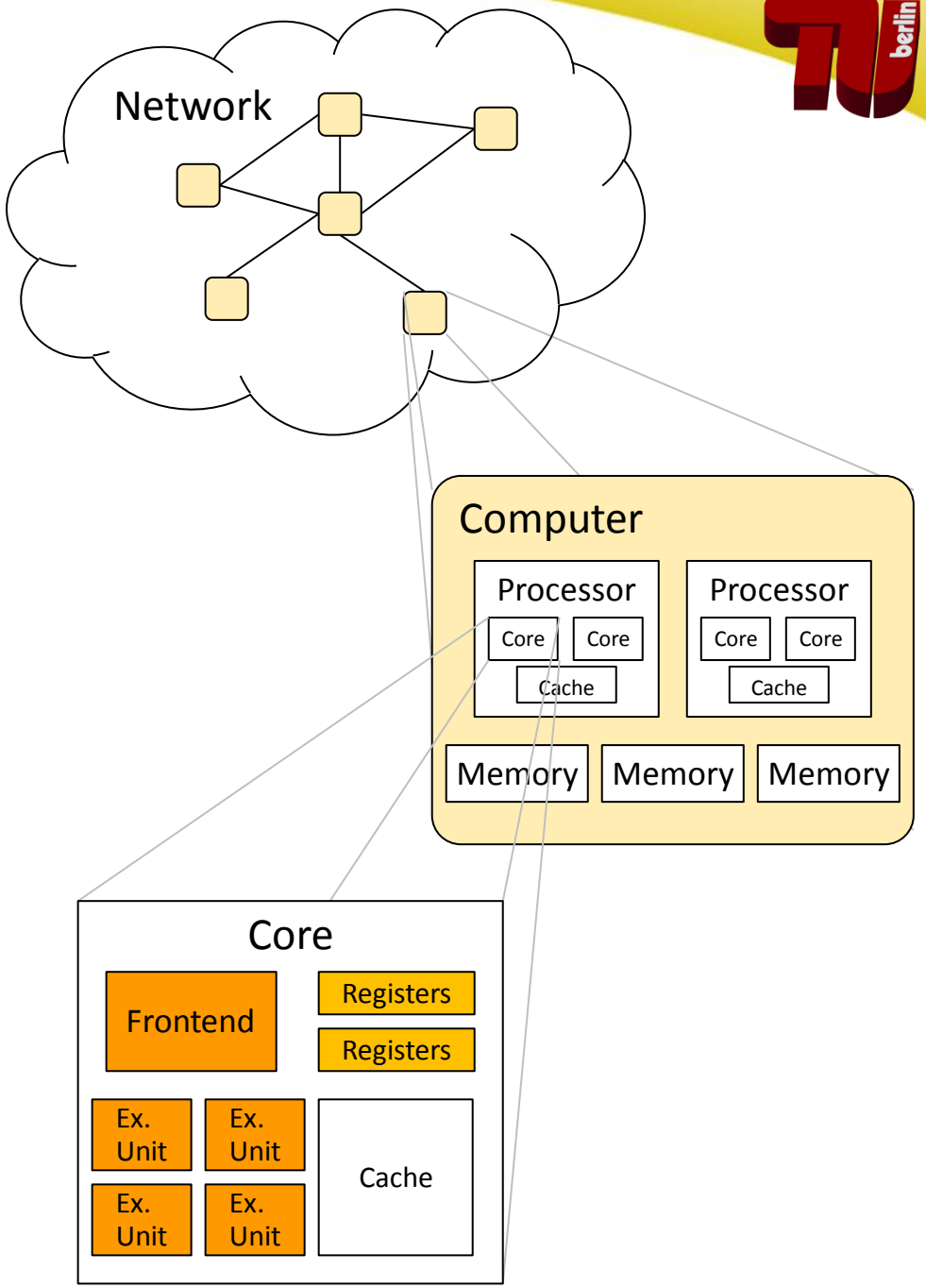
Computer
(Parallel, Server, Desktop, Mobile, Embedded, …)

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
ABCPL
ACE
ACT++
ADDAP
Adl
Adsmith
AFAPI
ALWAN
AM
AMDC
Amoeba
AppLeS
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BlockComm
BSP
C*
C**
C4
CarlOS
Cashmere
CC++
Charlotte
Charm
Charm++
Chu
Cid
Cilk
CM-Fortran
Code
Concurrent ML
Converse
COOL
CORRELATE
CparPar
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE
DIPC
Distributed Smalltalk
DOLIB
DOME
DOSMOS
DRL
DSM-Threads
Ease
ECO
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FLASH
FM
Fork
Fortran-M
FX
GA
GAMMA
Glenda
GLU
GUARD
HAsL
HORUS
HPC
HPC++
HPF
IMPACT
ISETL-Linda
ISIS
JADA
JADE
Java RMI
javaPG
JAVAR
JavaSpaces
JIDL
Joyce
Karma
Khoros
KOAN/Fortran-S
LAM
Legion
Lilac
Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold
Mentat
Meta Chaos
Midway
Millipede
Mirage
Modula-2*
Modula-P
MOSIX
MpC
MPC++
MPI
Multipol
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OOF90
Orca
P++
P-RIO
P3L
P4-Linda
Pablo
PADE
PADRE
Panda
Papers
Para++
Paradigm
Parafrase2
Paralation
Parallaxis
Parallel Haskell
Parallel-C++
ParC
ParLib++
ParLin
Parlog
Parmacs
Parti
pC
pC++
PCN
PCP:
PCU
PEACE
PENNY
PET
PETSc
PH
Phosphorus
POET
Polaris
POOL-T
POOMA
POSYBL
PRESTO
Prospero
Proteus
PSDM
PSI
PVM
QPC++
Quake
Quark
Quick Threads
Sage++
SAM
SCANDAL
SCHEDULE
SciTL
SDDA
SHMEM
SIMPLE
Sina
SISAL
SMI
SONiC
Split-C
SR
Sthreads
Strand
SUIF
SuperPascal
Synergy
TCGMSG
Telegraphos
The FORCE
Threads.h++
TRAPPER
TreadMarks
UC
uC++
UNITY
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

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