Topic 22: Multi-Processor Parallelism

COS / ELE 375

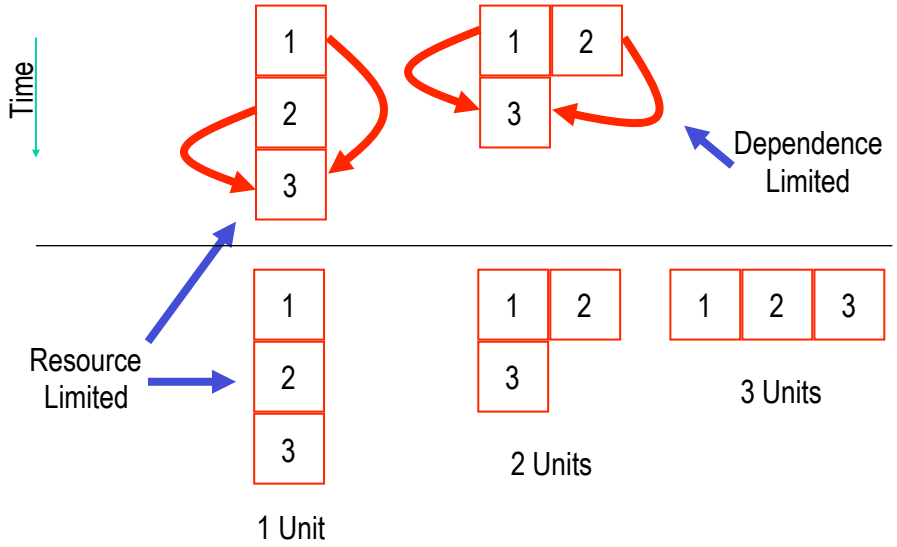
Computer Architecture and Organization

Princeton University
Fall 2015

Prof. David August

Review: Parallelism

Independent units of work can execute concurrently if sufficient resources exist



Review: Where to Find Parallelism?

Parallelism can be found/exists at different granularities

- Instruction Level
  - Ex: add instruction executes with multiply instruction
  - Compiler and hardware good at finding this
- Thread Level
  - Ex: screen redraw function executes with recalculate in spreadsheet
  - Programmers OK at finding this
- Process Level
  - Ex: Simulation job runs on same machines as spreadsheet
  - Users good at creating this

Thread Level Parallelism

Programmer generally makes TLP explicit
Compilers can extract threads in regular programs

```
for (i = 0; i < 200; i++)
  for(j = 1; j < 20000; j++)
    val[i,j] = val[i,j-1] + 1;
```

```
forall(i = 0; i < 200; i++)
  for(j = 1; j < 20000; j++)
    val[i,j] = val[i,j-1] + 1;
```

Thread Level Parallelism
Synchronization

- Unlike in ILP, flow of data/dependences must be explicit

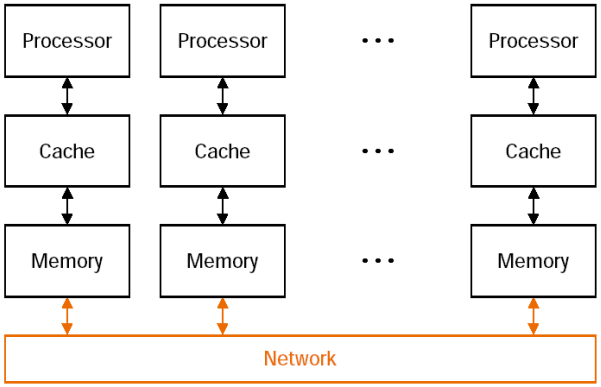
```
while(ptr = ptr->next)
  sum += ptr->val;
```

```
while(ptr = ptr->next)
  produce(ptr);
produce(NULL);
```

```
while(ptr = consume(ptr))
  sum += ptr->val;
```

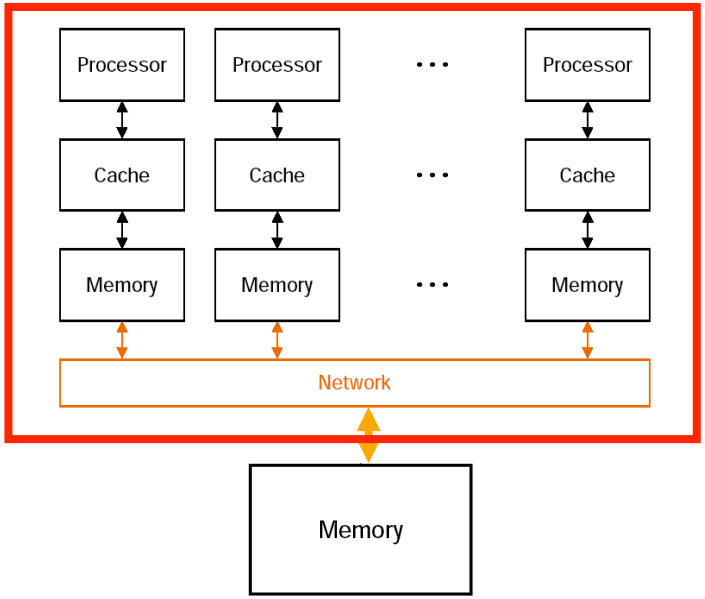
Communication and Synchronization… (order and flow)

Multiple Processor Organization
Message Passing/Private Memory



- Threads communicate directly (send, receive)
- Scales relatively well
- No memory coherence problem (for the hardware at least)

Multiple Processor Organization
May exist on single chip



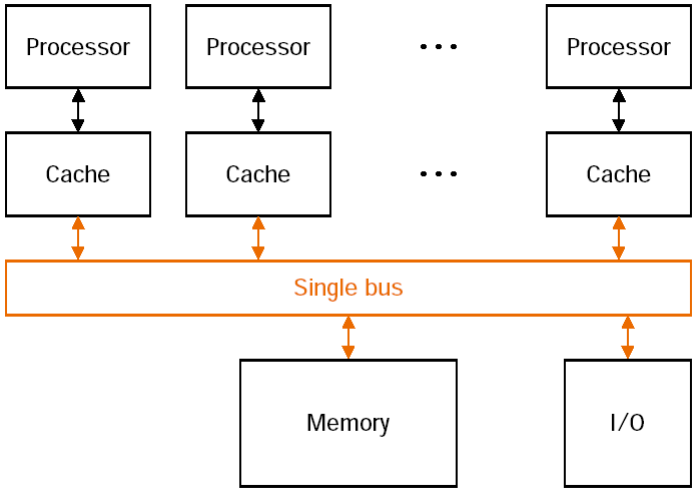
Thread Level Parallelism

Programmer generally makes TLP explicit
Compilers can extract threads in regular programs

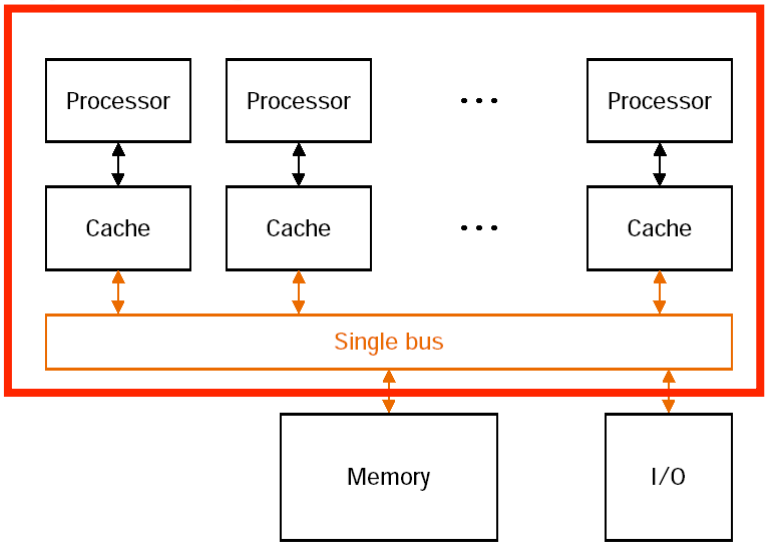
```
for (i = 0; i < 200; i++)
  for(j = 1; j < 20000; j++)
    val[i,j] = val[i,j-1] + 1;
```

```
forall(i = 0; i < 200; i++)
  for(j = 1; j < 20000; j++)
    val[i,j] = val[i,j-1] + 1;
```

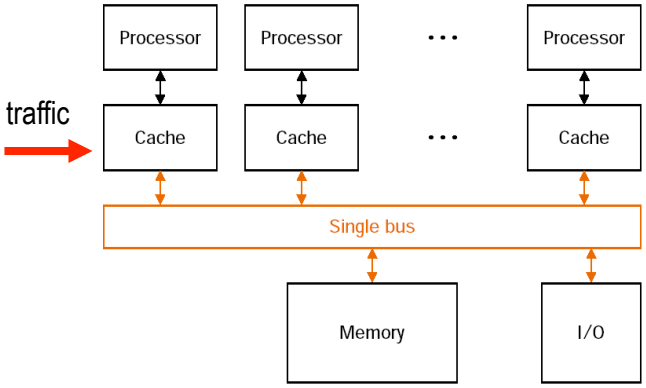
Multiple Processor Organizations
Shared Memory/Shared Bus



Multiple Processor Organizations
May be on single chip!



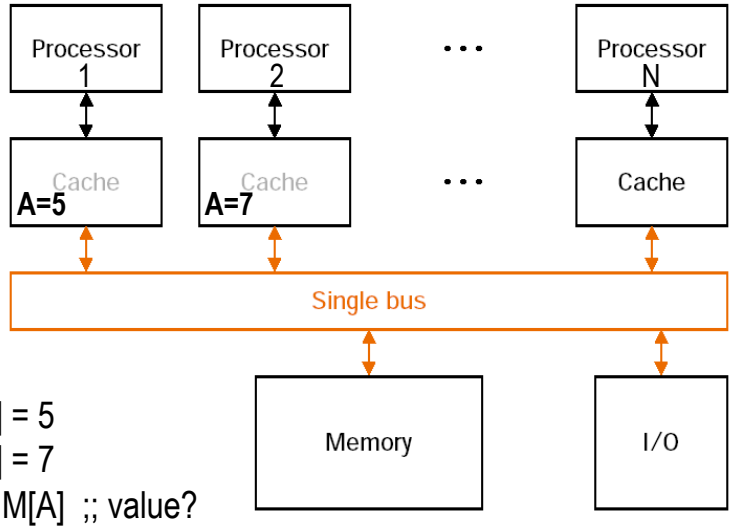
Multiple Processor Organizations
Shared Bus



Synchronization and Communication through memory

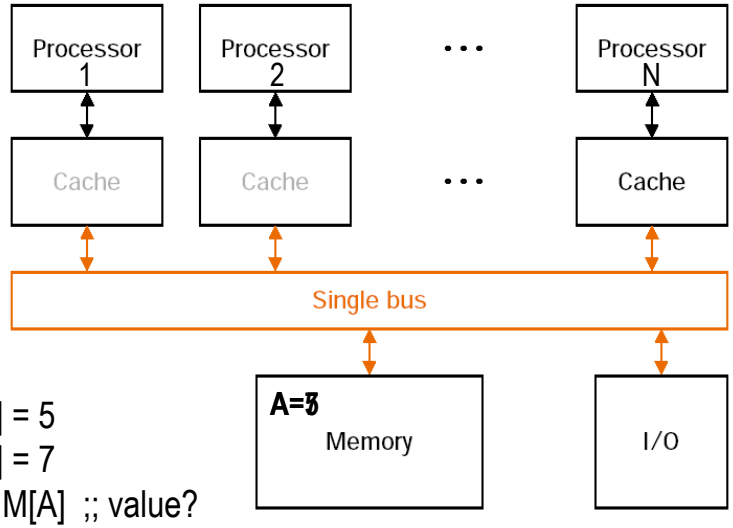
The cache coherency problem

Multiple Processor Organizations
Shared Bus



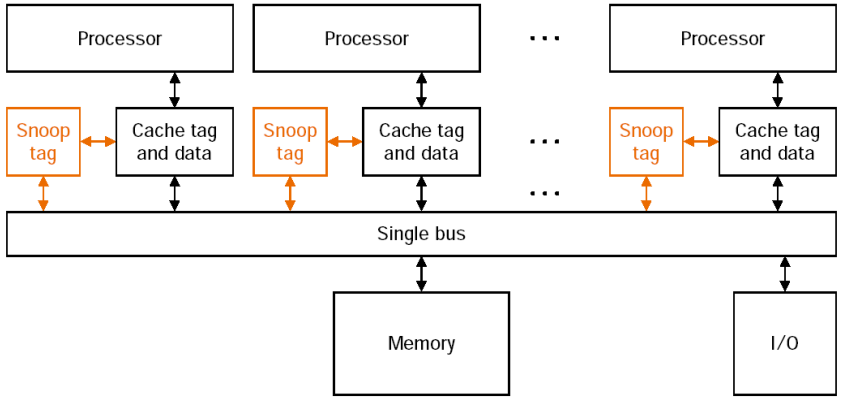
P1: M[A] = 5
P2: M[A] = 7
P1: r1 = M[A] ;; value?
PN: r1 = M[A] ;; value?

Cache Coherency "Solution"



P1: M[A] = 5
P2: M[A] = 7
P1: r1 = M[A] ;; value?
PN: r1 = M[A] ;; value?

Cache Coherency Solution: Snoopy Bus



Snooping Protocols

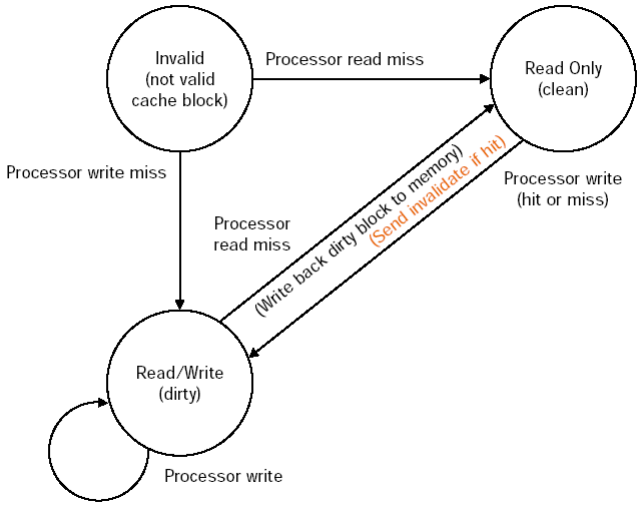
- Variety of protocols minimize traffic for different situations
- Generally many states including:
  invalid, dirty read/write, clean/read-only

Reads: just work

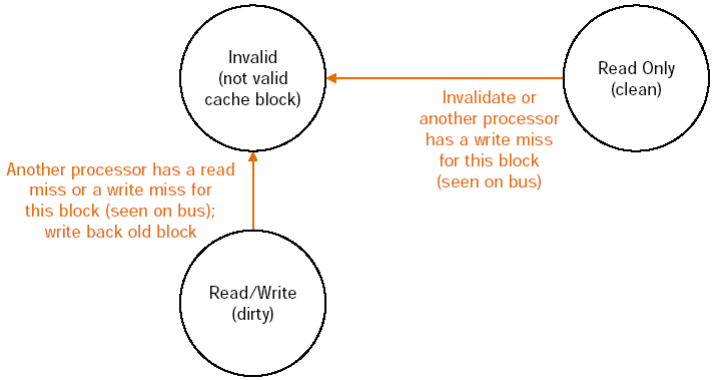
Writes:

- Write-Invalidate - other caches with address invalidate line (block) - only first write generates traffic
- Write-Update - other caches with address update the values in the line (block) - like write through

Sample Protocol Signals From Processor

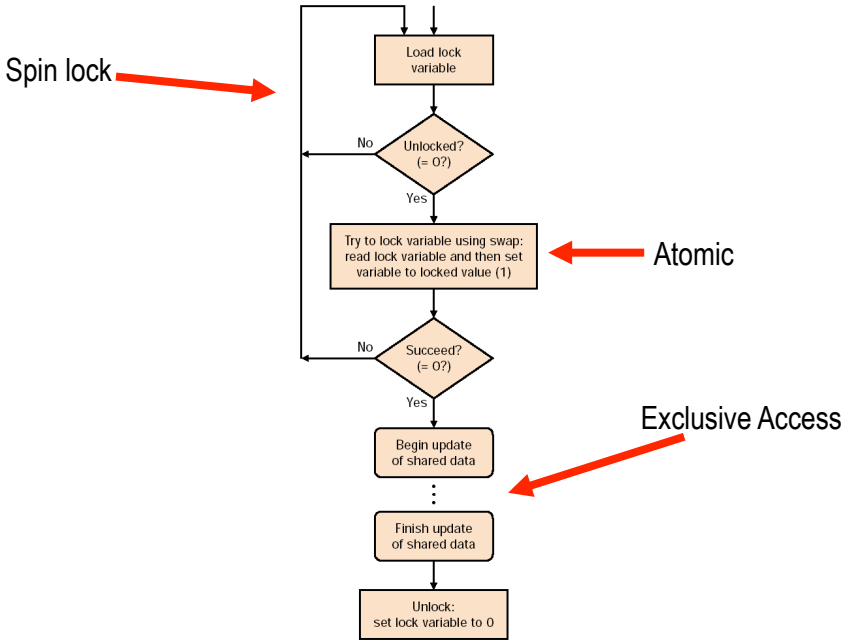


Sample Protocol
Signals From Bus



Other protocols are MESIer

Synchronization/Semaphores



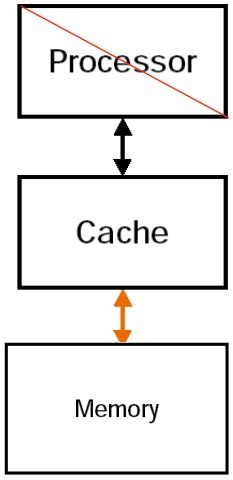
Multiple Processor Organizations
Simultaneous Multithreading ("Hyperthreading")

- Multiple threads in single core
- Helps when single thread ILP is low

Like ILP processor, but

- Multiple PCs, one per thread
- Instructions are tagged with thread ID
- Architectural register file per thread
- Threads share execution resources

- Cross thread synchronization and communication through memory/ cache



CFGs, PCs, and Cross-Iteration Deps

1. r1 = 10

1. r1 = r1 + 1
2. r2 = MEM[r1]
3. r2 = r2 + 1
4. MEM[r1] = r2
5. Branch r1 < 1000

No register live outs

Loop-Level Parallelization: DOALL

1. r1 = 10
1. r1 = r1 + 1 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r1 < 1000

1. r1 = 9
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r1 < 999

1. r1 = 10
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r1 < 1000

No register live outs

Another Example

1. r1 = 10
1. r1 = r1 + 1 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

1. r1 = 9
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

1. r1 = 10
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

No register live outs

Another Example

1. r1 = 10
1. r1 = r1 + 1 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

1. r1 = 9
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

1. r1 = 10
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

No register live outs

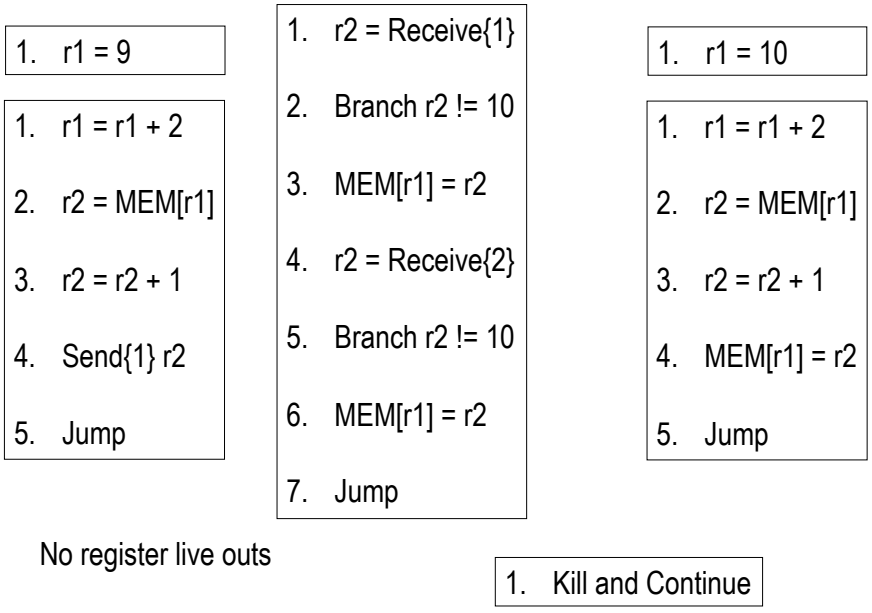
Speculation

1. r1 = 9
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

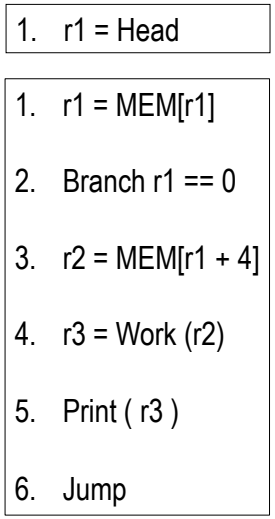
1. r1 = 10
1. r1 = r1 + 2 2. r2 = MEM[r1] 3. r2 = r2 + 1 4. MEM[r1] = r2 5. Branch r2 == 10

No register live outs

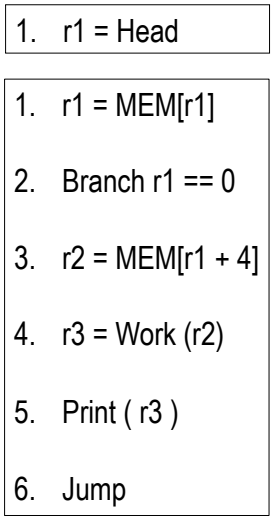
Speculation, Commit, and Recovery



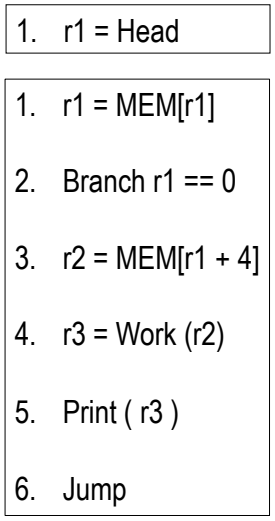
Difficult Dependences

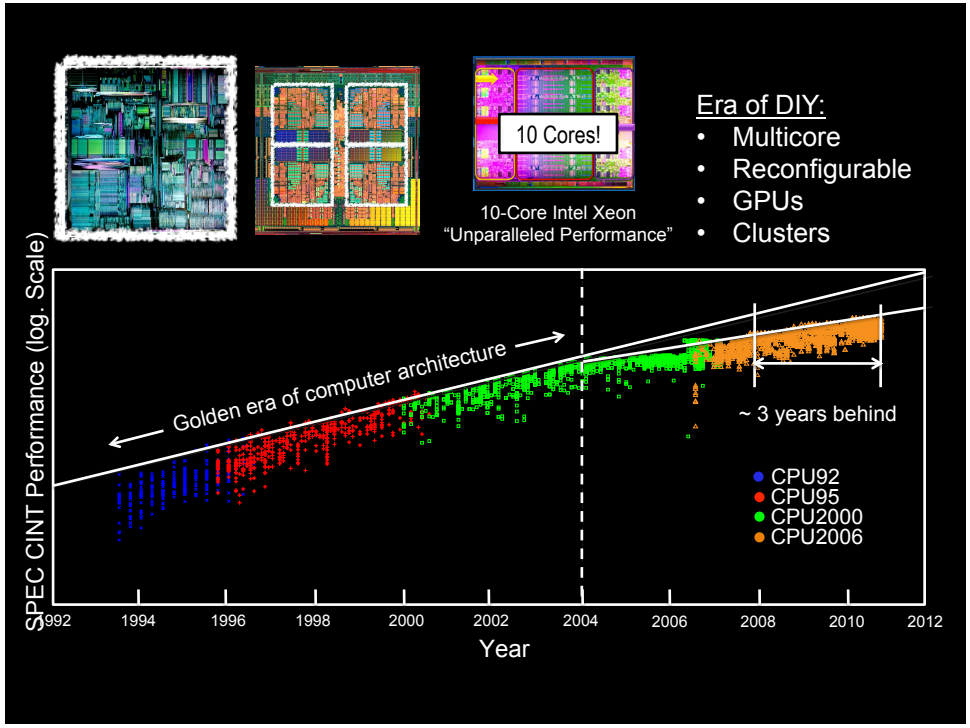


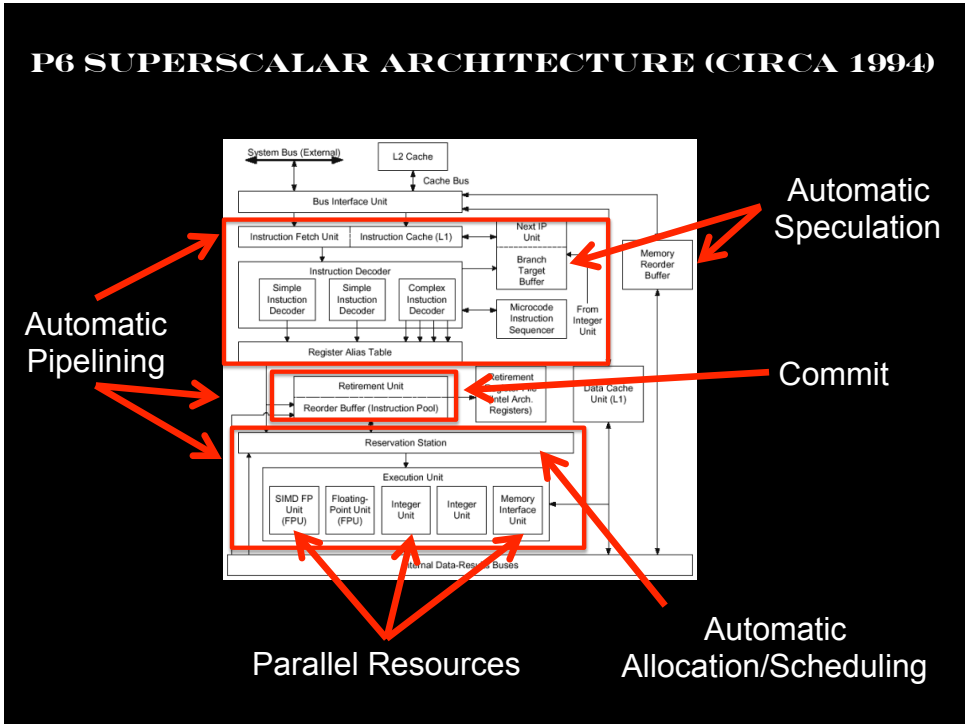
DOACROSS

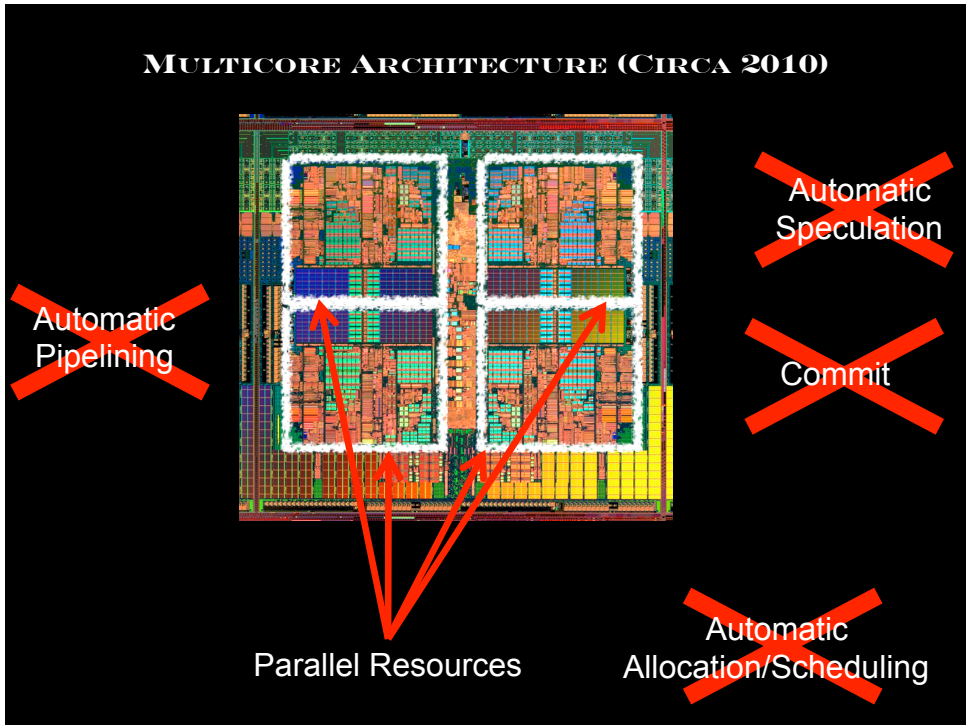


PS-DSWP



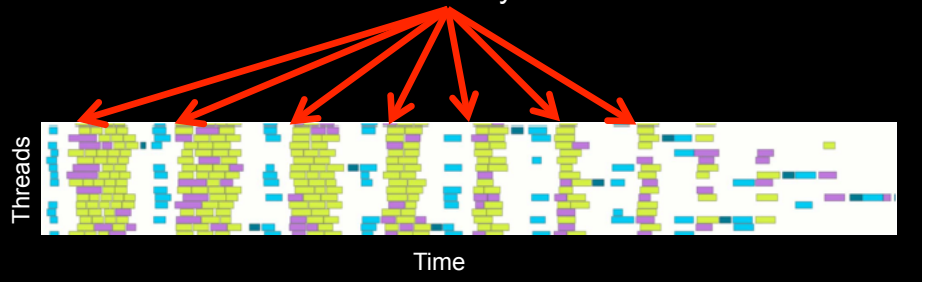




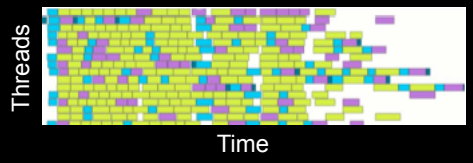


ABCPL
ACE
ACT++
Active messages
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*.
"C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL
CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
ECO
Eiffel
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda
GLU
GUARD
HAsL.
Haskell
HPC++
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold
Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
p4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm
Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
pC++
PCN
PCP:
PH
PEACE
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick Threads
Sage++
SCANDAL
SAM
pC++
SCHEDULE
SciTL
POET
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

Parallel Library Calls

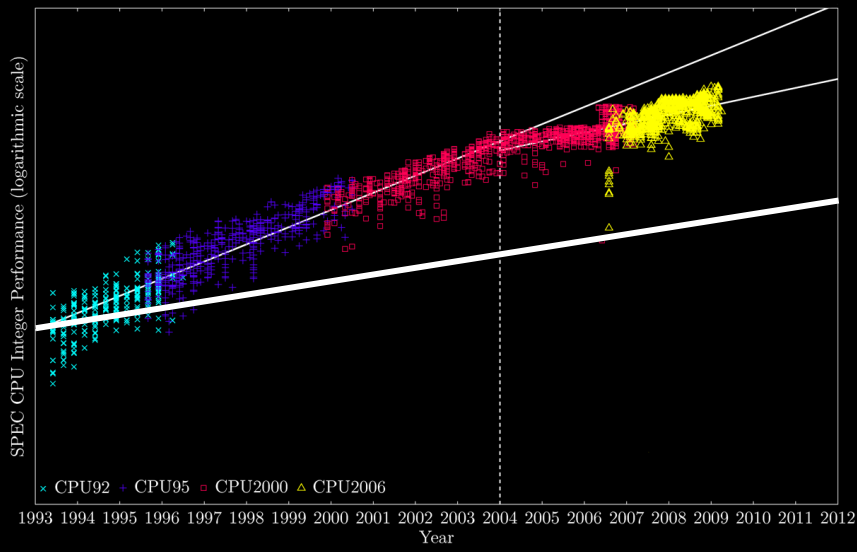


Realizable parallelism

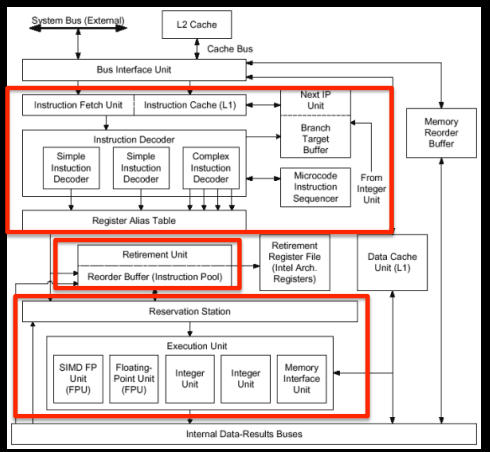


Credit: Jack Dongarra

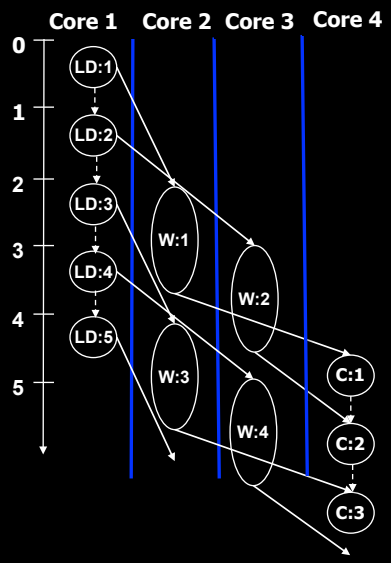
"Compiler Advances Double Computing Power Every 18 Years!"
– Proebsting's Law



P6 SUPERSCALAR ARCHITECTURE



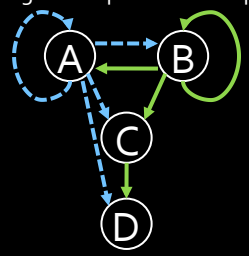
Spec-PS-DSWP



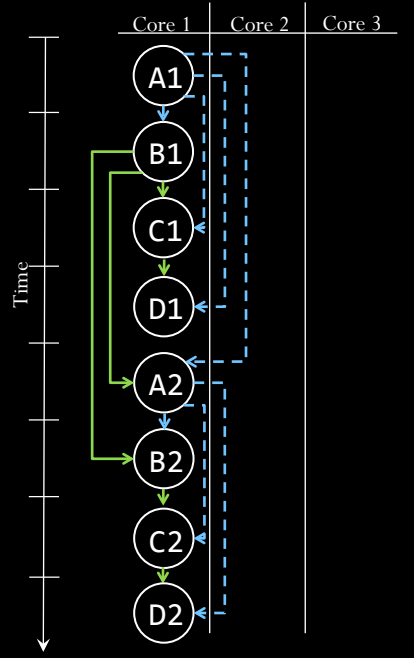
Example

```
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
   }
```

Program Dependence Graph



Control Dependence
Data Dependence

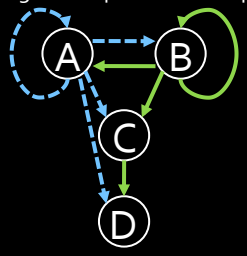


Spec-DOALL

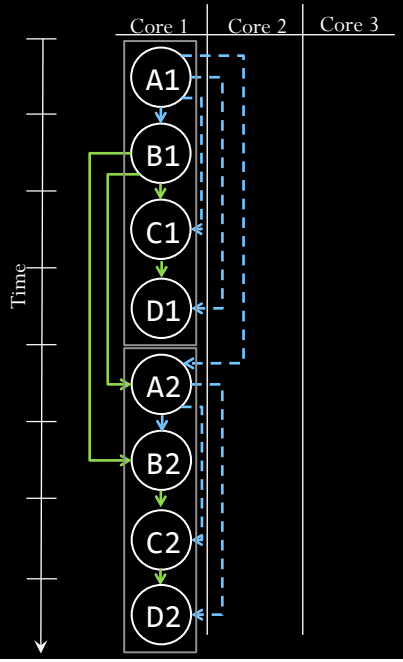
Example

```
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
   }
```

Program Dependence Graph



Control Dependence
Data Dependence

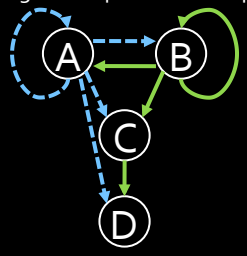


Spec-DOALL

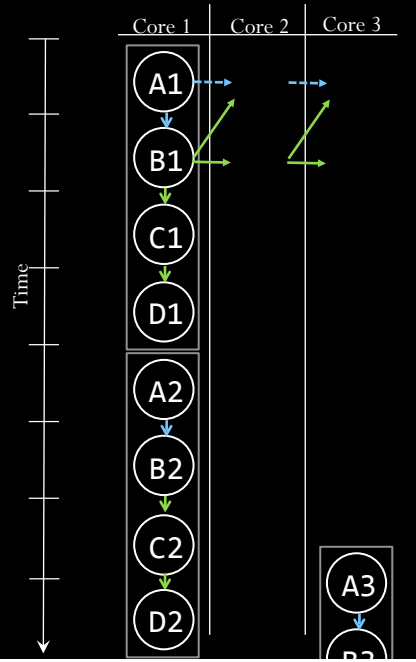
Example

```
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
   }
```

Program Dependence Graph



Control Dependence
Data Dependence



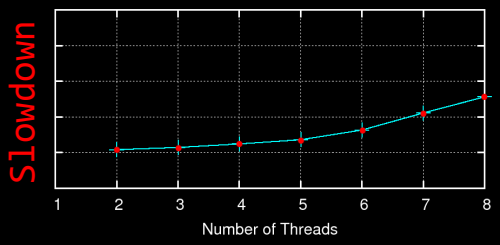
Spec-DOALL

Example

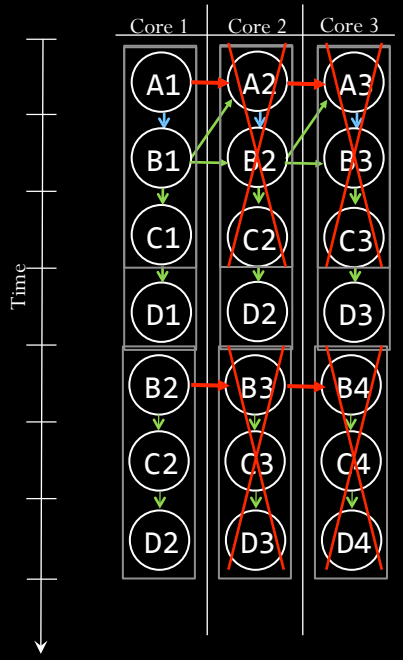
```
A: while (node) {
B:   node = node->next;
C:   res = work(node);
D:   write(res);
   }
```

Program Dependence Graph

197.parser

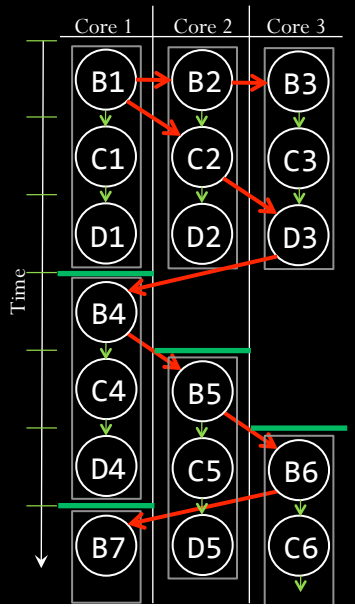


Control Dependence
Data Dependence



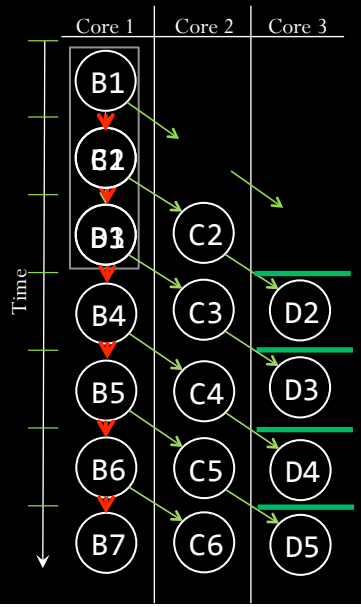
Spec-DOACROSS

Throughput: 1 iter/cycle

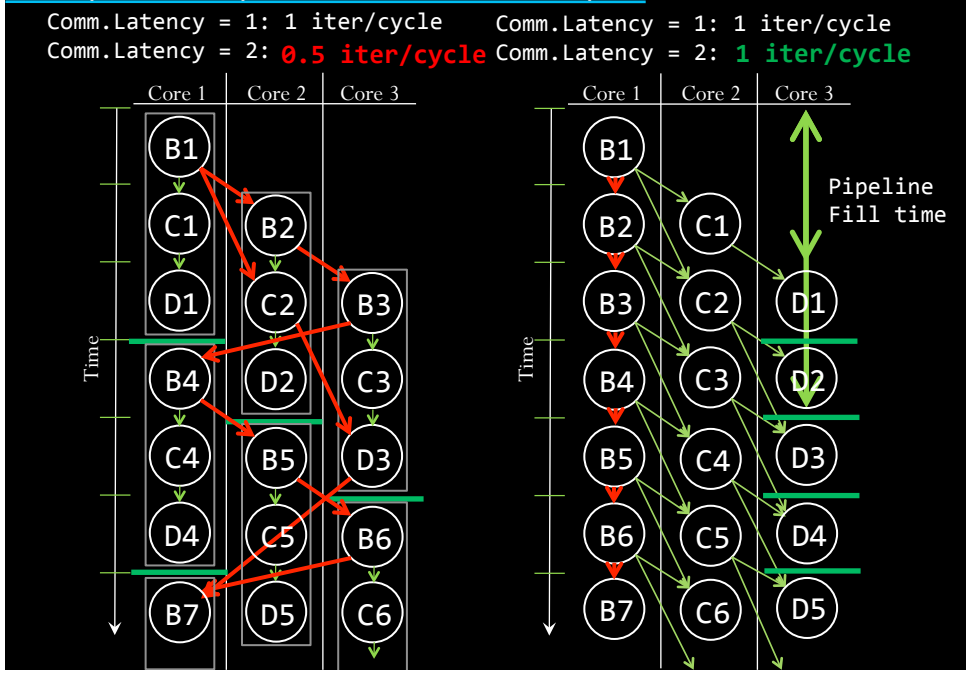


Spec-DSWP

Throughput: 1 iter/cycle



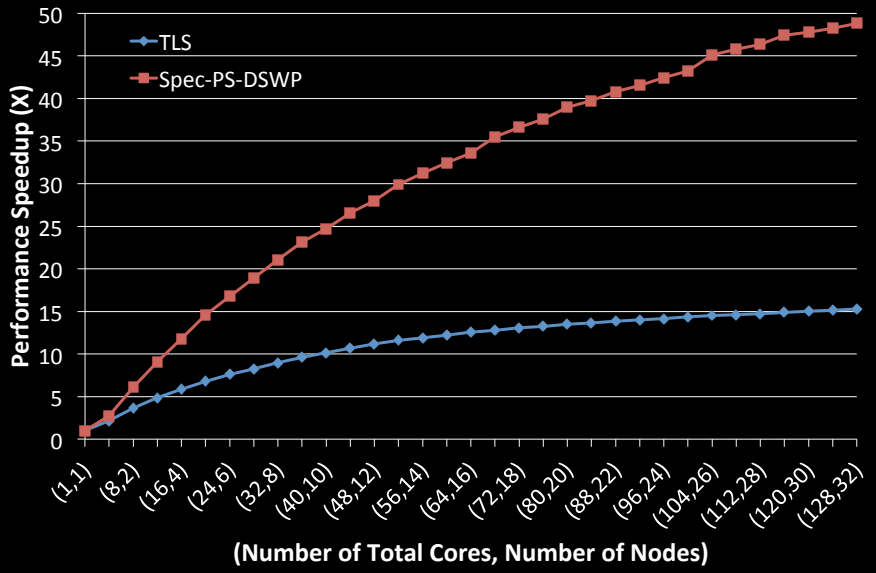
Comparison: Spec-DOACROSS and Spec-DSWP



Spec-DOACROSS vs. Spec-DSWP

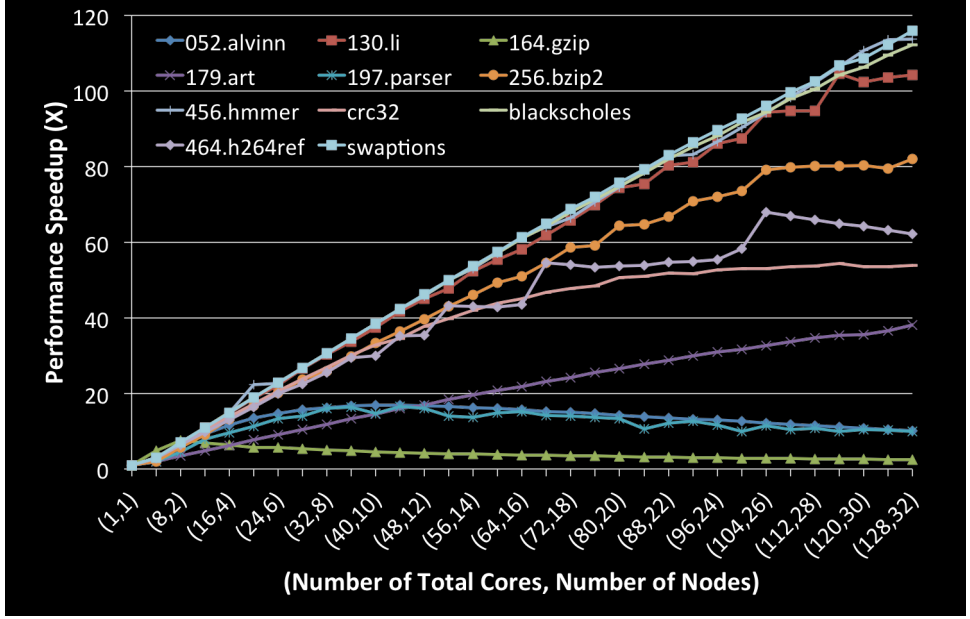
[MICRO 2010]

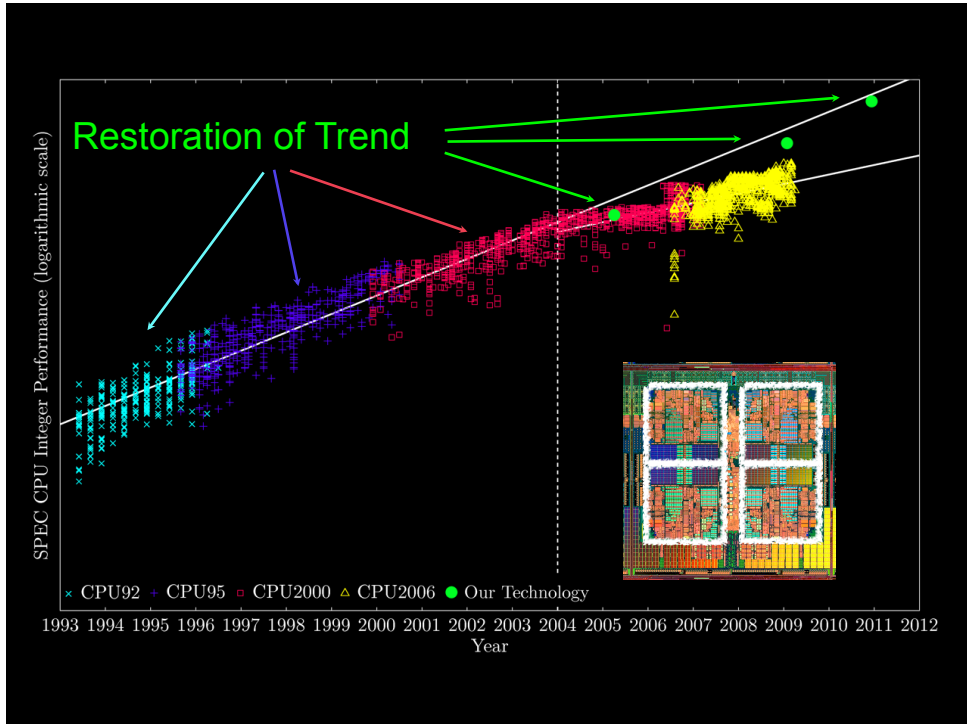
Geomean of 11 benchmarks on the same cluster



Performance relative to Best Sequential

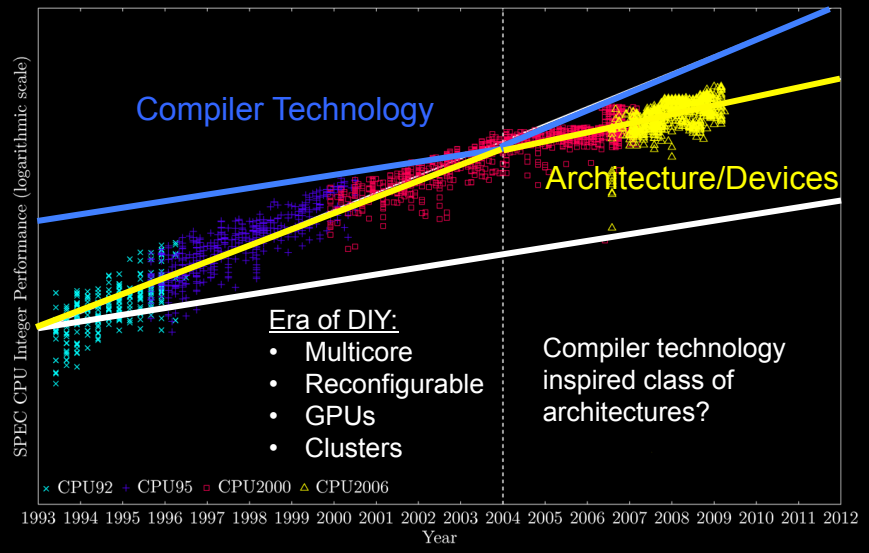
128 Cores in 32 Nodes with Intel Xeon Processors [MICRO 2010]





"Compiler Advances Double Computing Power Every 18 Years!"

– Proebsting's Law



Era of DIY:

- Multicore
- Reconfigurable
- GPUs
- Clusters

Compiler technology inspired class of architectures?