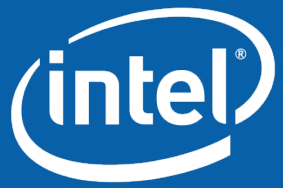


Bias Scheduling in Heterogeneous Multicore Architectures

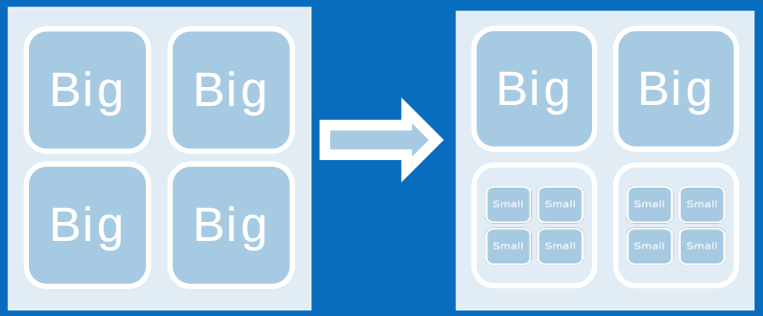
David Koufaty

Dheeraj Reddy

Scott Hahn

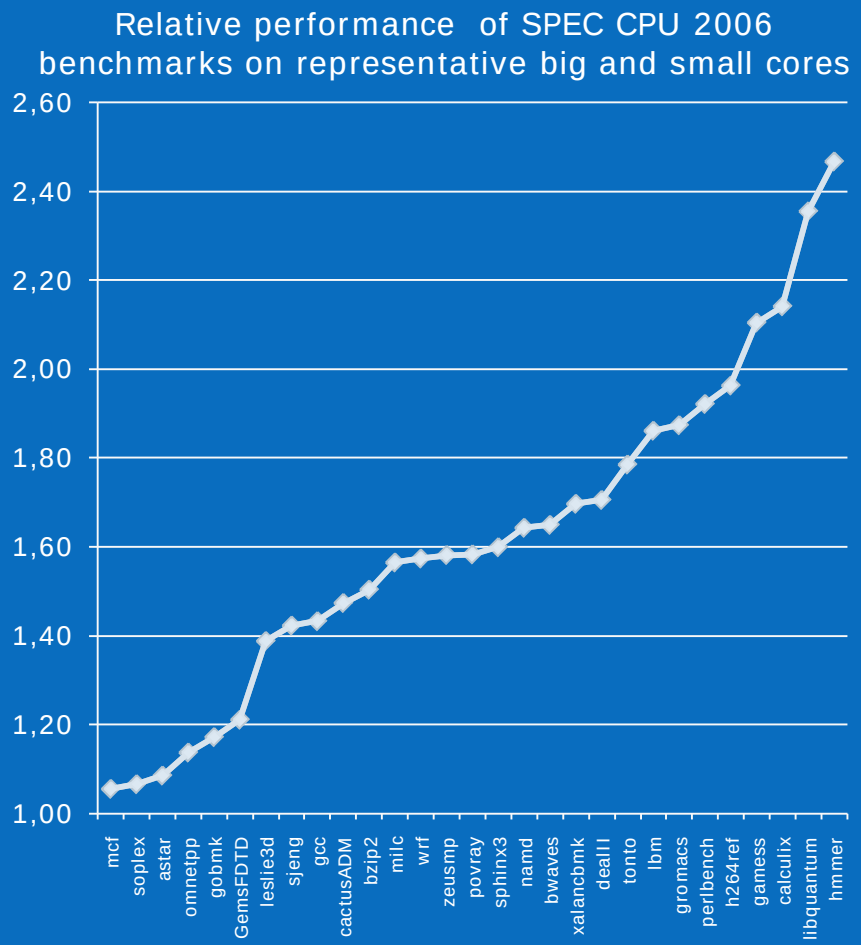
Motivation

- Mainstream multicore processors consist of identical cores
- Complexity dictated by product goals, generally fall into two camps
  - Complex microarchitectures: large area, power inefficient, great single thread performance
  - Simple microarchitectures: small area, power efficient, poor single thread performance
- Heterogeneous systems achieve the benefits of both but pose significant new system challenges



Why asymmetry matters – a challenge

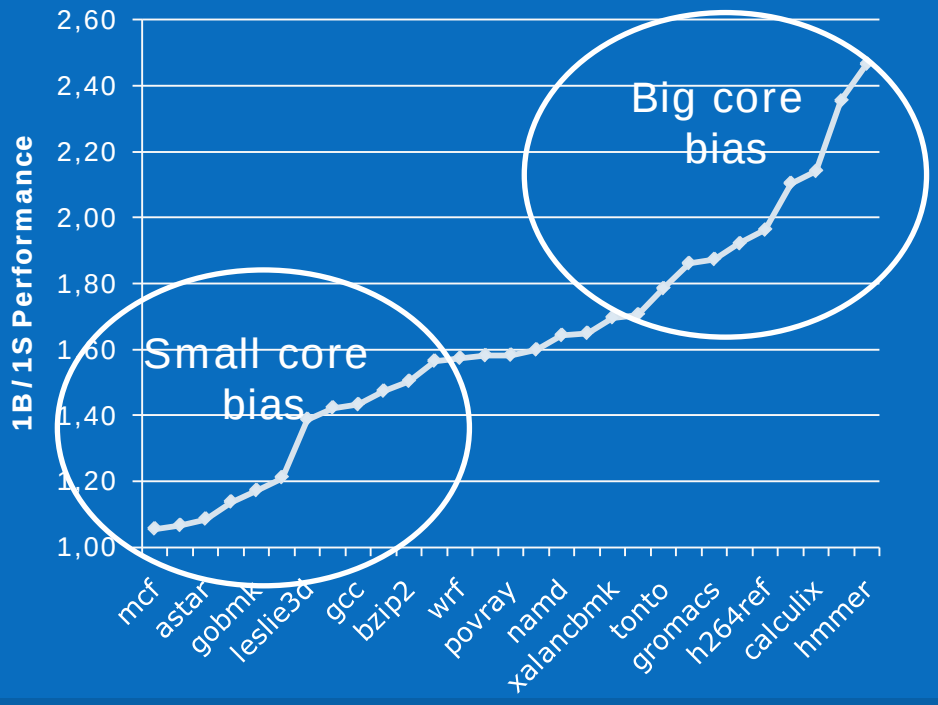
- OS has traditionally assumed homogeneous hardware
- Asymmetric systems do not provide a uniform computing capability
  - Performance of a process varies dramatically
  - Some care little about core type, while others show a 2x slowdown
- We focus on a performance asymmetric systems with two core types



Bias

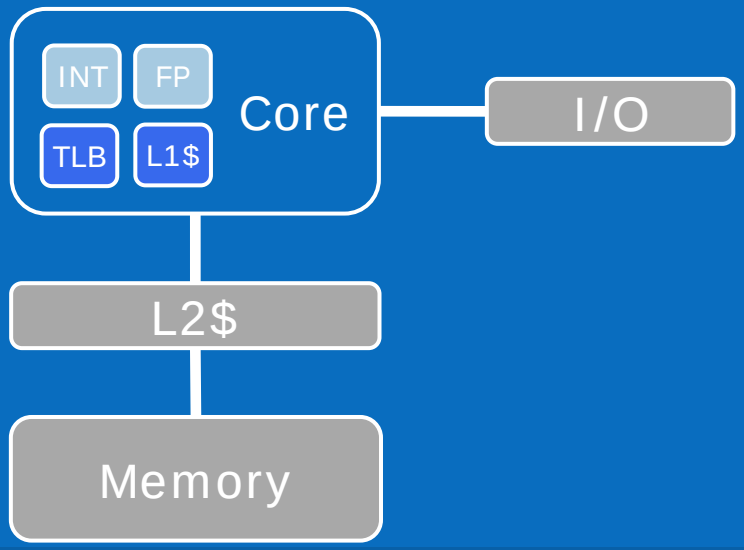
- Affinity of a **thread** to a core type
  - A thread has big core bias if its big/small core speedup is ***large***
  - A thread has small core bias if its big/small core speedup is ***small***
- Bias is dynamic, changes over time and with thread input

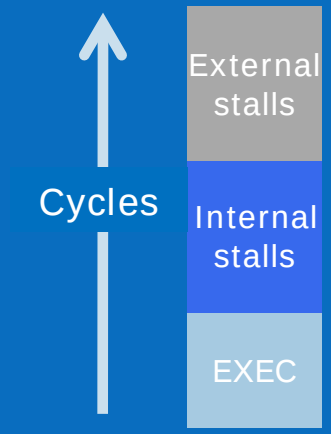
- Are there indicators of bias?
  - Without offline profiling
  - Without sampling on both cores
  - Adaptive and dynamic



CPI breakdown

- CPI summarizes thread performance
- Stalls can be broken down broadly into two types
  - Stalls related to on internal core resources (TLB, private caches, branch mispredicts, other resources)
  - Stalls related to external resources (shared caches, memory, I/O)



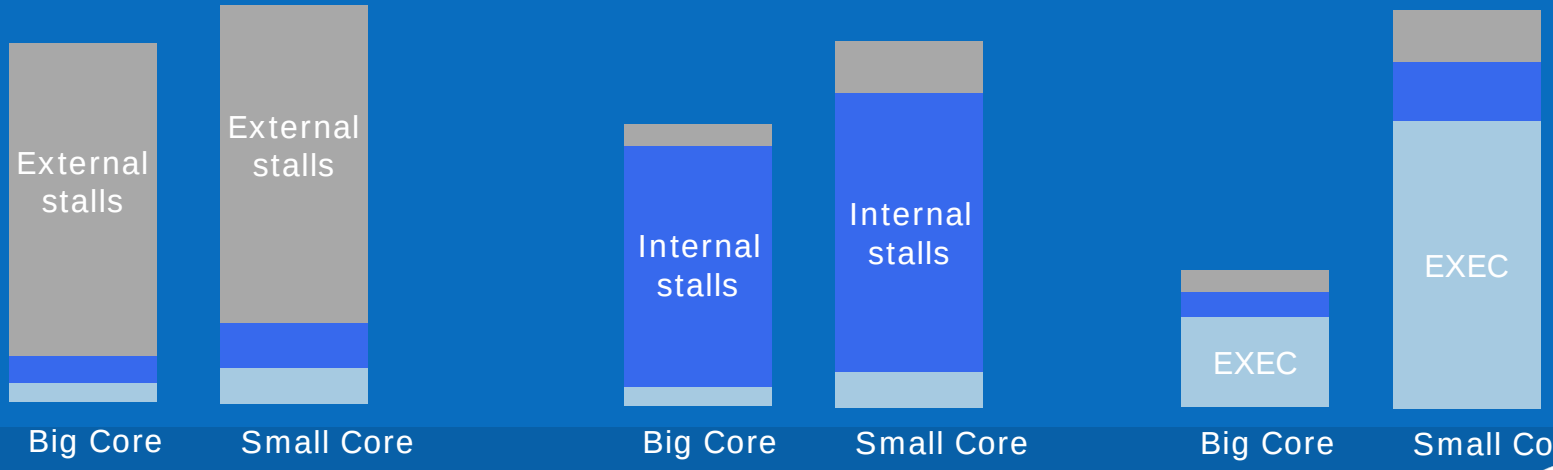


Typical CPI stack, showing were clocks are being spent on average for each instruction

Bias and stalls correlation

Three common cases:

- CPI is dominated by external stalls
  - Growing core/memory gap, latency is hidden only slightly better in big core
  - Thread will exhibit small gains on a big core
- CPI is dominated by internal stalls
  - Large number of branch mispredicts, TLB misses, dependencies, etc.
  - Big core resources underutilized can lead to small gains on a big core
- CPI is dominated by execution cycles
  - Big core execution resources usage maximized
  - Thread will exhibit large gains on a big core

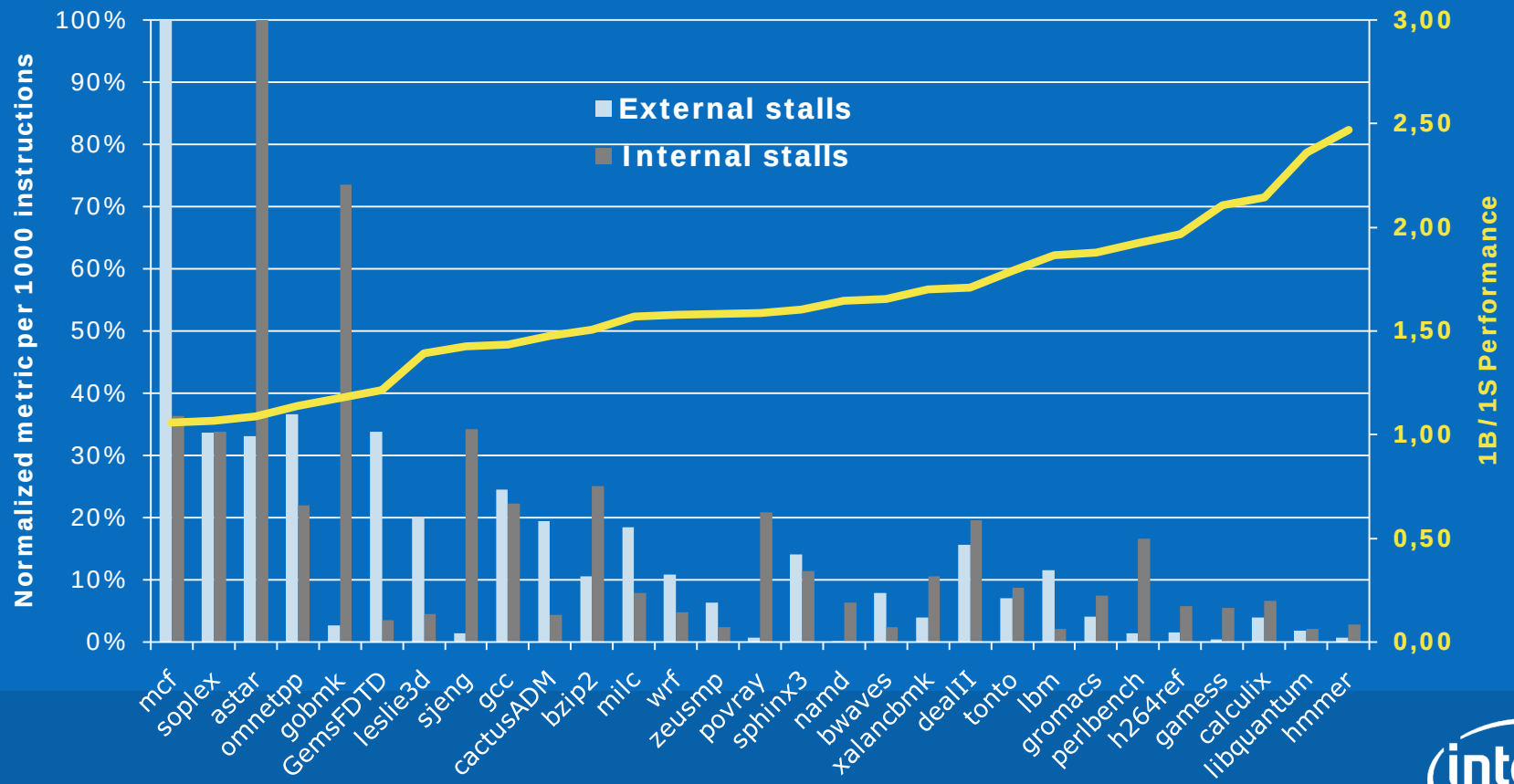


Computing bias

- Use performance monitoring hardware to get online measurements of internal and external stalls
- Accurate counts are hard to come by due to speculation and overlapping events
- Stall metrics require knowledge of the two core microarchitectures
  - Not all events results in stalls (e.g. prefetch)
  - Not all stalls equally impact the two microarchitecture
  - Microarchitects should provide abstraction in future cores
- Our empirical approach uses specific existing counters correlated with the big/small core performance
    - Offcore requests (demand reads and RFOs)
    - Instruction starvation

Estimating stalls

- Combination of external and internal stalls provide a good online estimation of bias
- More work needed to increase accuracy on outliers

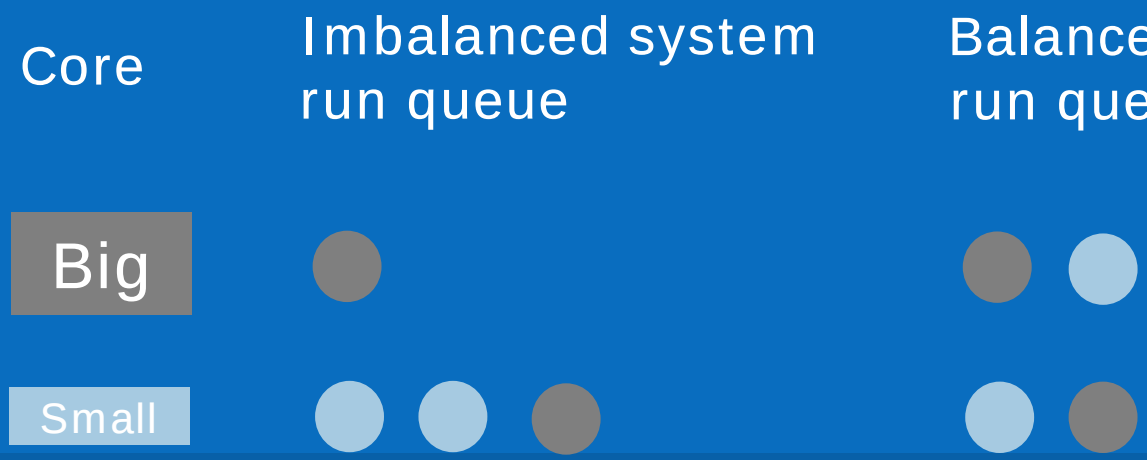


Bias scheduling

- Computing thread bias
  - Virtualize performance monitoring hardware to keep per thread counts
  - Compute metrics per 1K instructions
  - Running average of metric over a sliding instruction window
  - Bias determined by value of the metric falling below or above configurable thresholds
- Bias aware scheduling design
  - Bias scheduling does not change fundamental scheduling decisions
  - Most changes limited to the load balancing code
  - Other run queue optimizations to improve certain searches

Biased load balancing

- On imbalanced systems, migrate threads with misplaced bias
  - Threads are migrated from busiest to idlest core
  - When the core types are different, select a thread with the highest bias towards the target core type
  - If none exists, perform load balancing as usual
- On balanced systems, switch cores for threads with opposite bias
  - Periodically check the big cores for threads with small core bias
  - If some exists, check the small cores for threads with big core bias
  - Perform the switch

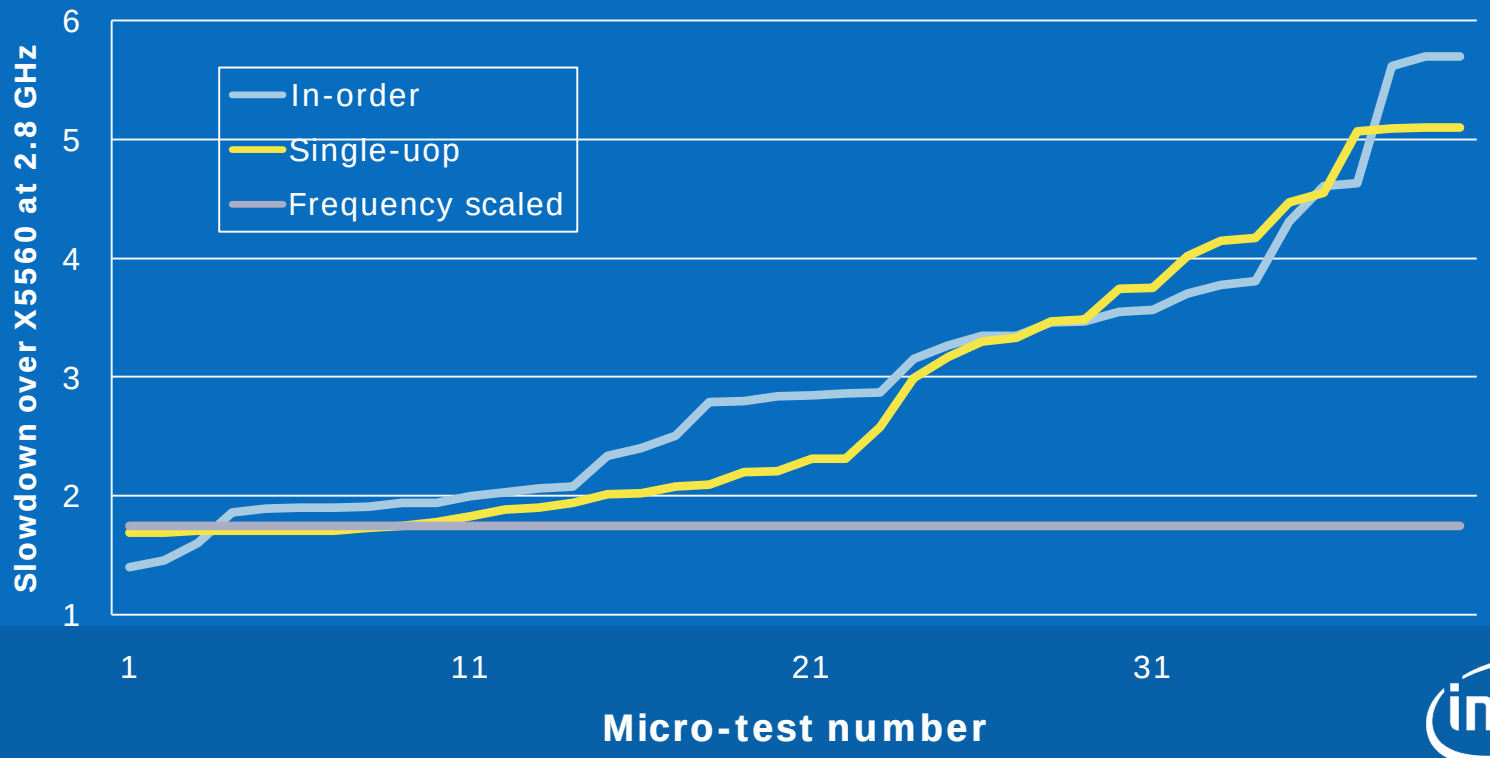


Asymmetric platform prototype

- State of the art research has been using DVFS to emulate heterogeneous platform
  - Cores at different frequencies
- Instead, our approach changes the internal operation of the core to mimic a different core type
  - Using proprietary tools, we restrict instruction retirement bandwidth to a single instruction
- Our experience with this system shows a significantly different behavior than DVFS

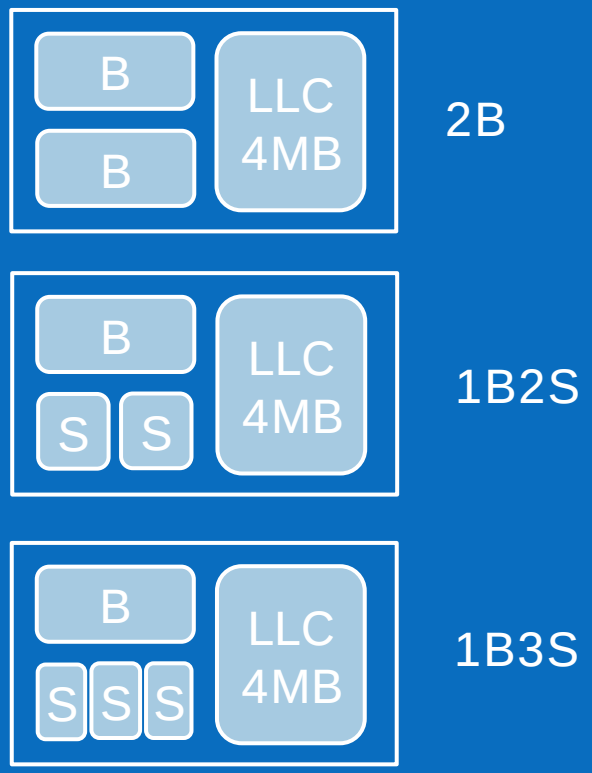
Asymmetric platform prototype

- We compare the **core** performance of 2.8 GHz X5560 against
  - A X5560 at 1.6 GHz (Frequency scaled)
  - A X5560 at 2.8 GHz in single uop retirement mode (Single-uop)
  - An in-order core at 2.8 GHz (In-order)
- Using 40 memory independent micro-tests



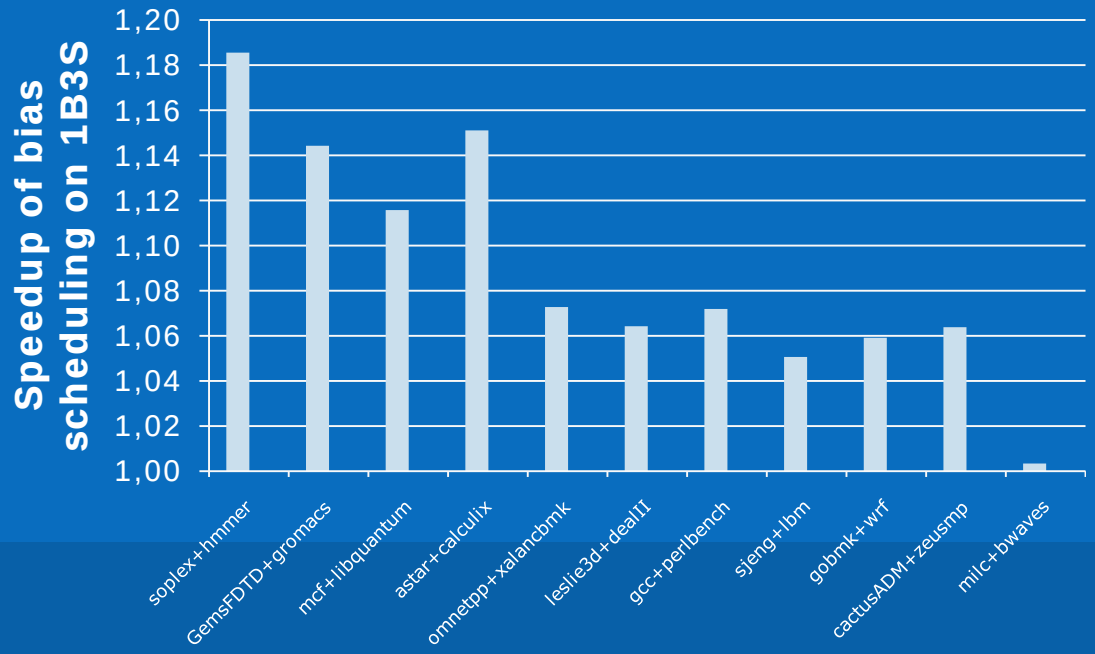
Evaluation

- Asymmetric system has 2:1 and 3:1 small/big ratios, based on projected area (1B2S, 1B3S)
- Same frequency for all cores
- Implementation on Linux 2.6.27
- Heterogeneous and homogeneous workloads composed of a mix of SPEC CPU 2006 components
  - 22 of 28 benchmarks represented, picked all over the spectrum



Heterogeneous workloads

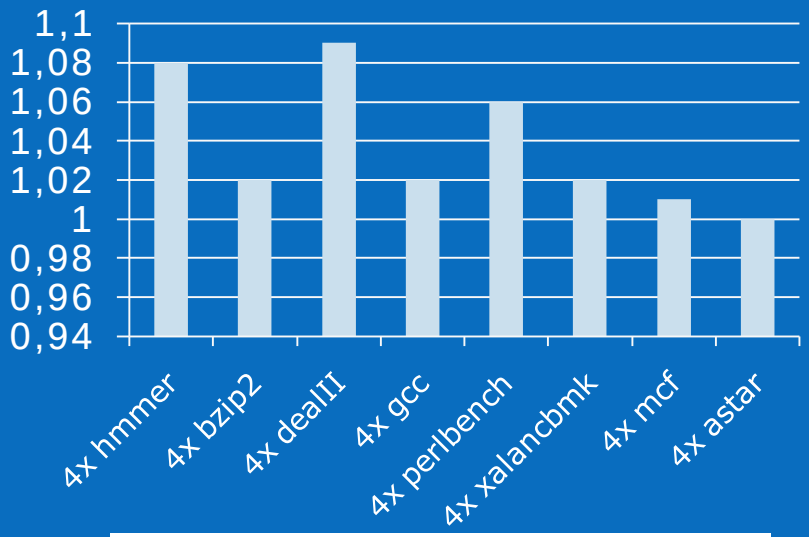
- Compare bias scheduling against stock kernel on 1B3S
- Gains are proportional to the amount of bias diversity
  - 11% gains when bias difference is strong, 4% when bias is similar
  - Trade-off gains on big core bias app (65%) for loss on small core bias app (-5%)
- Larger gains as the ratio of big/small approaches one (e.g. 1B2S)
  - Gains of big core are diluted with more small cores
  - Systems with few big cores do not gain from bias scheduling
- Outperforms the best static scheduling by up to 5%

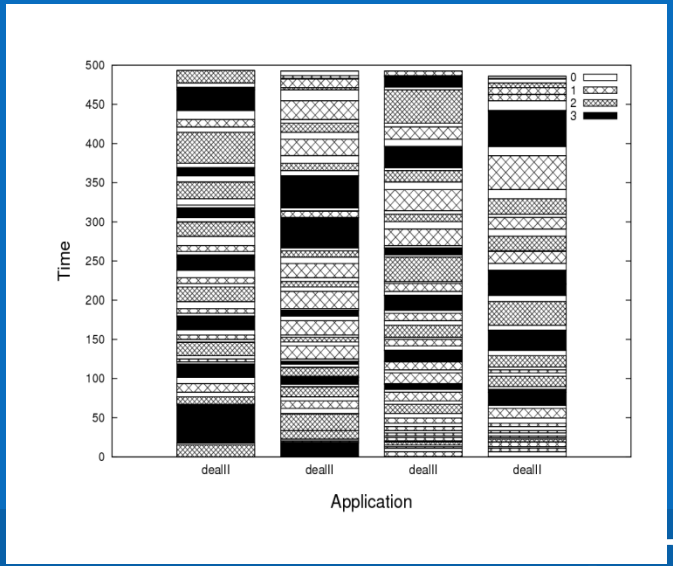


3-copies	1B/1S	1-copy	1B/1S
soplex	1.07	hmmer	2.47
GemsFDTD	1.21	gromacs	1.87
mcf	1.06	libquantum	2.36
astar	1.09	calculix	2.14
omnetpp	1.14	xalancbmk	1.70
leslie3d	1.39	dealII	1.71
gcc	1.43	perlbench	1.92
sjeng	1.42	lbm	1.86
gobmk	1.17	wrf	1.57
cactusADM	1.46	zeusmp	1.58
milc	1.57	bwaves	1.65

Homogeneous workloads

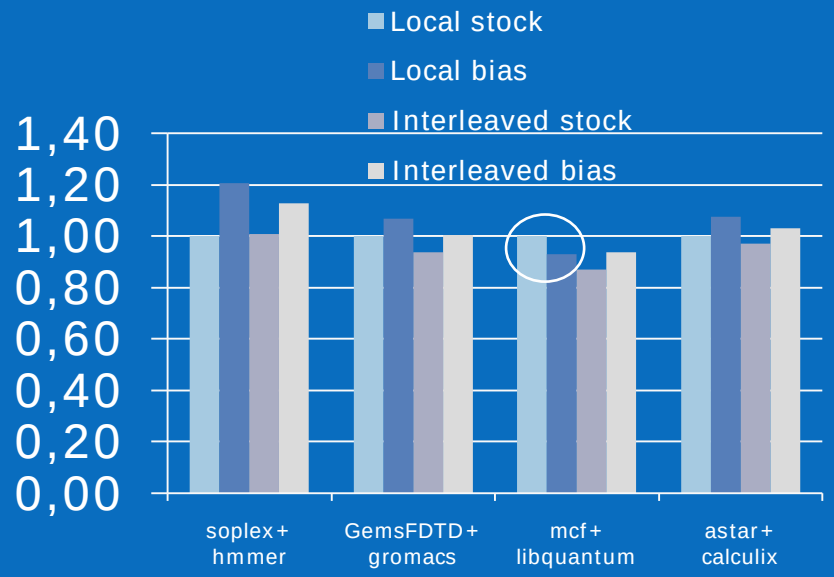
- Two types
  - Parallel applications
  - Multiple copies of a single application
- Smaller gains expected due to homogeneity of the test
  - Exploit temporal changes in bias during phase changes
  - Up to 9% gains, 3% on average
- Big core is distributed to the thread that most benefits from it at the time
  - Increases fairness when it matters
  - Even when threads run in lockstep (parallel workloads)

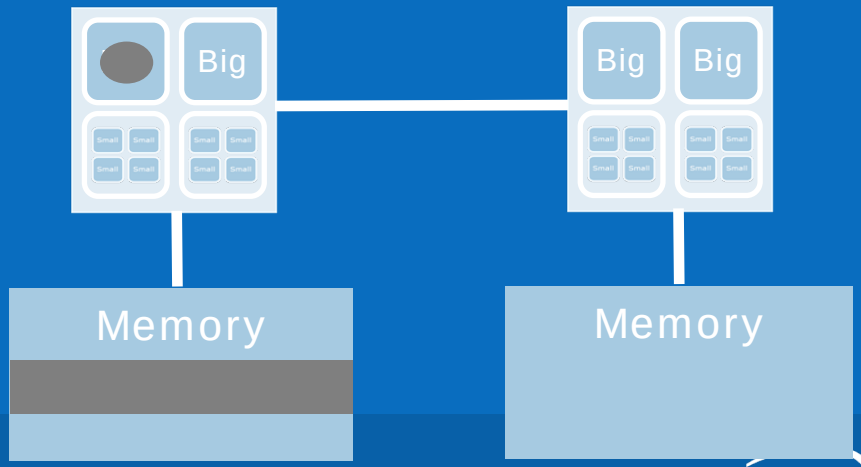




NUMA

- Previous data is on a single socket, memory is uniform
- Thread migrations not always beneficial in NUMA systems
- Depends on memory allocation policy
  - No policy is always best
- Interleaved memory is best to amortize penalty, maximize bandwidth
  - Bias is always good as well
- Local memory can minimize latency, maximize performance
  - Bias can migrate thread away from memory





Tuning the system

- Instruction window controls granularity of bias sample
  - A balance between fast detection of phase changes and migration cost
  - Large instructions windows (e.g. 4 billion) tend to outperform small ones (e.g. 256 million)
  - Sensitivity is not uniform across applications
- Stall thresholds
  - Chosen empirically based on application profiles
  - Can be automated by sampling application performance once
- Metrics
  - The biggest challenge to the practicality of any approach based on performance monitoring
  - Relevant events can change with microarchitecture
  - Desirable to design platform independent OS algorithms

Conclusions

- Two key proposals
  - Decomposing core stalls to predict the core type best suited for an application, its bias
  - An online, bias aware scheduling algorithm with minimal impact to existing schedulers
- Our results show that
  - Using DVFS to model asymmetry simplifies challenges
  - Bias scheduling delivers performance gains in proportion to the amount of bias diversity in the workload
  - Adjusting to dynamic application behavior delivers a performance benefit
- Future work
  - Build a real system to enable other studies like power optimizations, support for supervisor instruction asymmetry
  - We currently have a real heterogeneous system running Linux

```
[root@hetero ~]# grep "model name" /proc/cpuinfo
model name      : Pentium 75 - 200
model name      : Intel(R) Xeon(R) CPU          X5355  @ 2.66GHz
```