Hyper-Threading:

Simultaneous Multithreading on Pentium 4

Presented by:

Thomas Repantis

`trep@cs.ucr.edu`

Overview

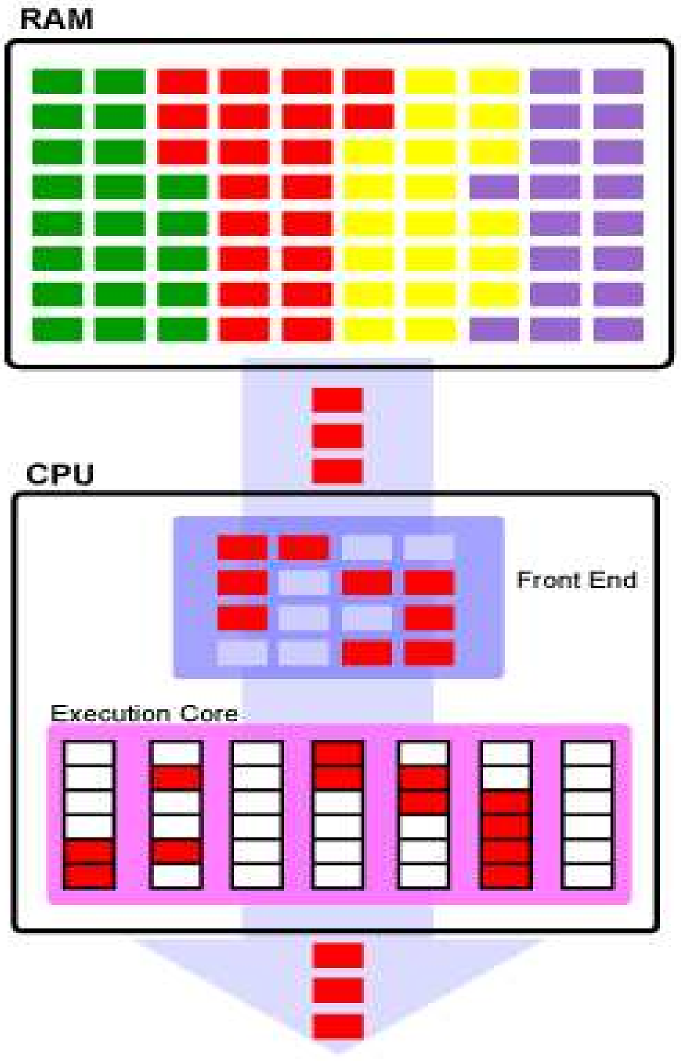
Multiple threads executing on a single processor without switching.

1. Threads
2. SMT
3. Hyper-Threading on P4
4. OS and Compiler Support
5. Performance for Different Applications

Threads

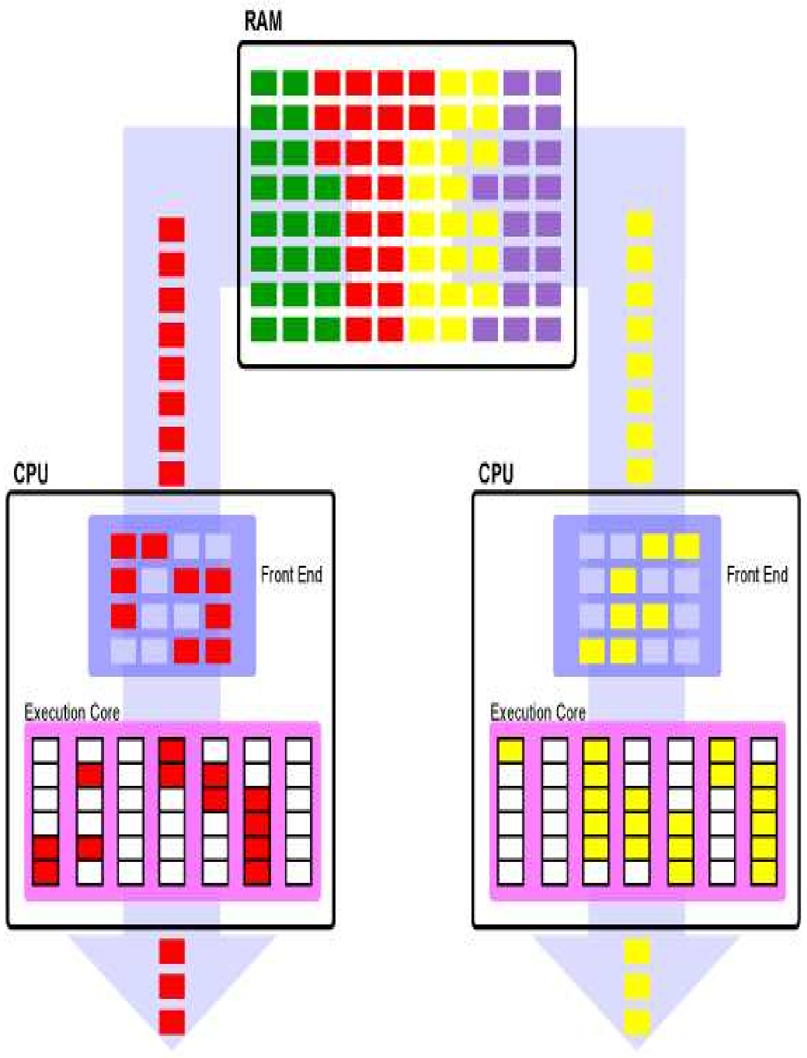
- **Process**: “A task being run by the computer.”
- **Context**: Describes a process’s current state of execution (registers, flags, PC...).
- **Thread**: A “light-weight” process (has its own PC and SP, but single address space and global variables).
- Each process consists of at least one thread.
- Threads allow faster context-switching and fine-grain multitasking.

Single-Threaded CPU



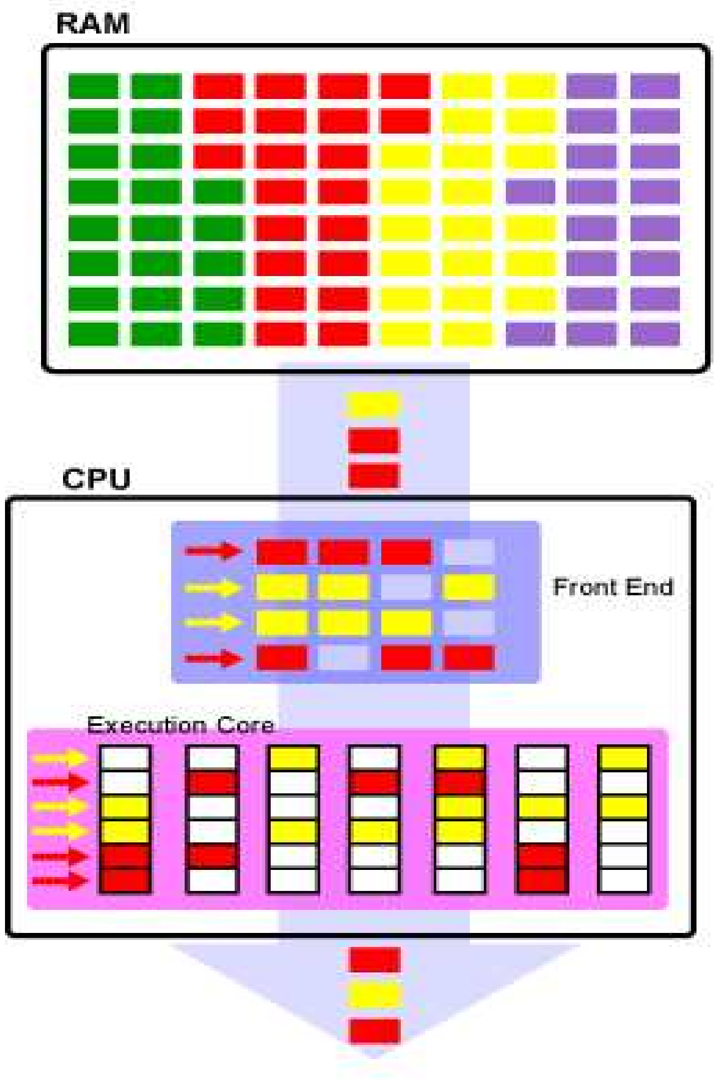
A lot of bubbles in the instruction issue and in the pipeline!

Single-Threaded SMP



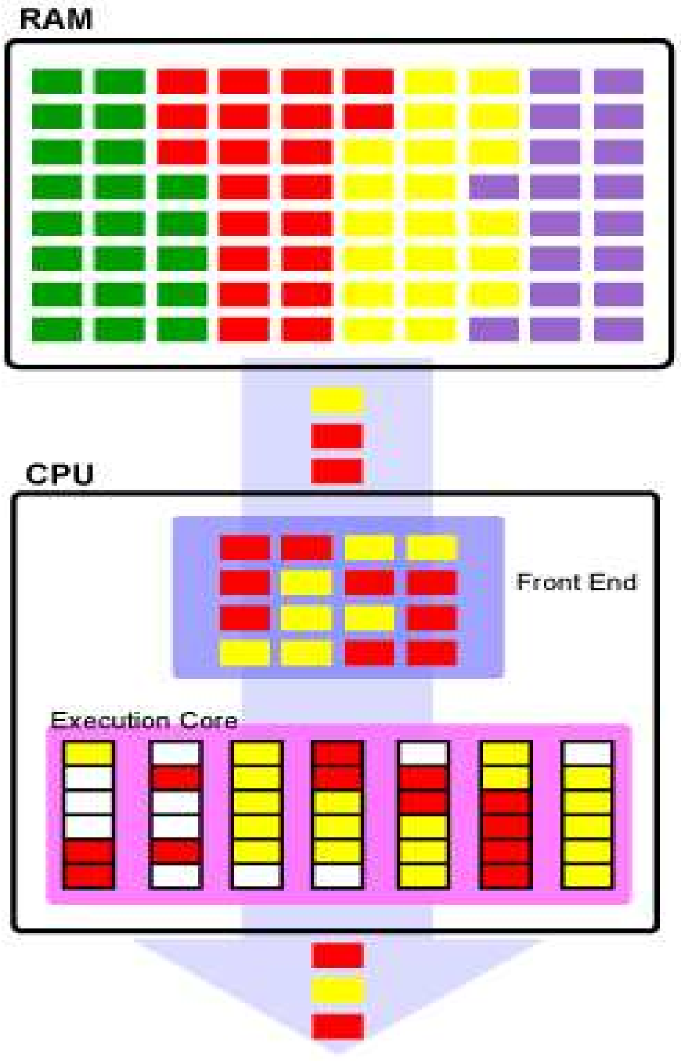
Executing processes are doubled, but bubbles are doubled as well!

Superthreaded CPU



Each issue and each pipeline stage can contain instructions of the same thread only.

Hyper-Threaded CPU (SMT)



Instructions of different threads can be scheduled on the same stage.

SMT vs TeraMTA

- Each processor of the TeraMTA has 128 streams, that include a PC and 32 registers.
- Each stream is assigned to a thread.
- Instructions from different streams can be pipelined on the same processor.
- However, in TeraMTA **only a single thread is active on any given cycle**.

SMT Benefits

SMT:

- Gives the OS the illusion of several (currently two) **logical processors**.
- Makes efficient use of resources.
- Overcomes the barrier of limited amount of ILP within just one thread.
- Is implemented by dividing processor resources to replicated, partitioned, and shared.

Replicated Resources

Each logical processor has independent:

- Instruction Pointer
- Register Renaming Logic
- Instruction TLB
- Return Stack Predictor
- Advanced Programmable Interrupt Controller
- Other architectural registers

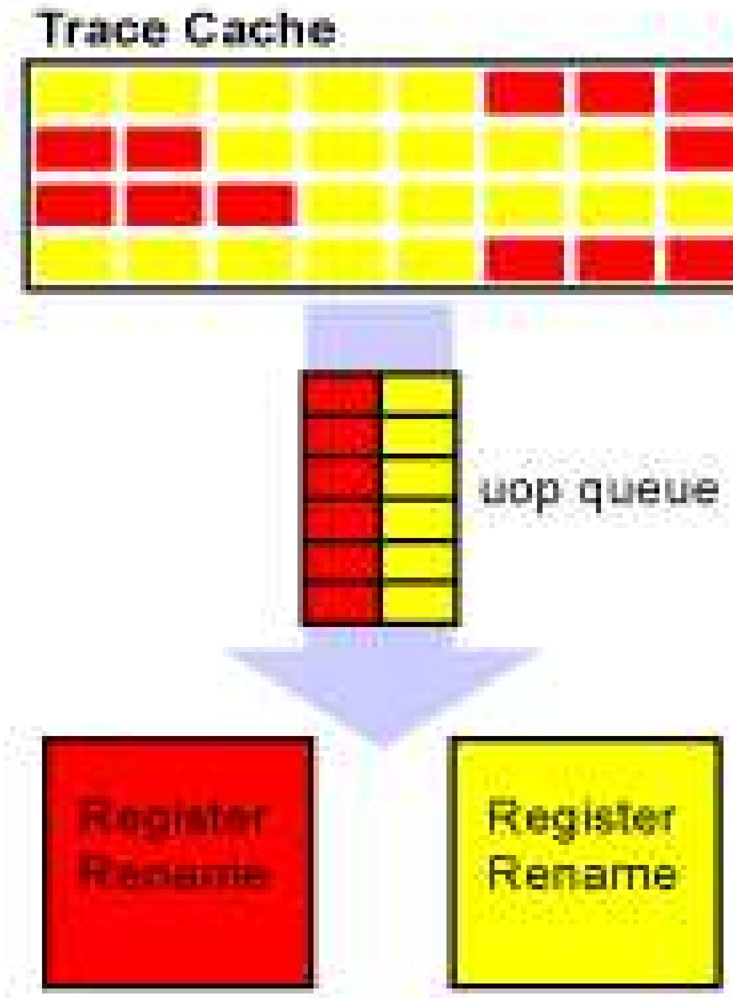
Partitioned Resources

Each logical processors gets exactly half of:

- Re-order buffers (ROBs)
- Load/Store buffers
- Several queues (e.g. scheduling, uop (micro-operations))

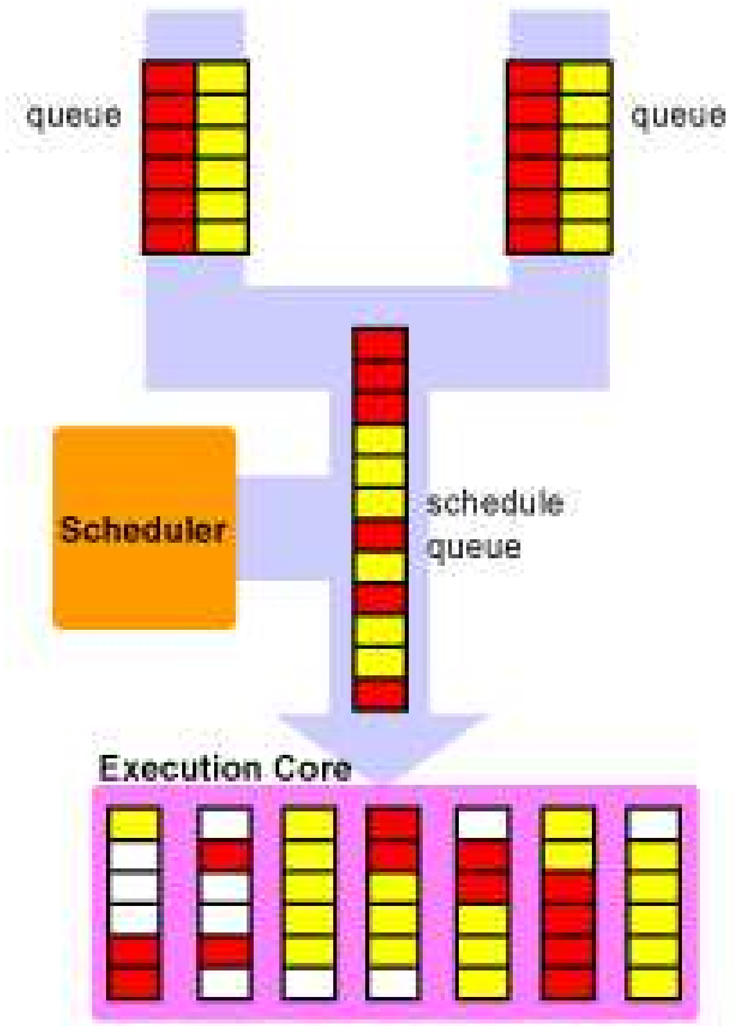
Partitioning prohibits a logical processor from monopolizing the resources.

Statically Partitioned Queue



Specific positions are assigned to each processor.

Dynamically Partitioned Queue



A limit is imposed to the positions each processor can use, but no specific positions are assigned.

Shared Resources

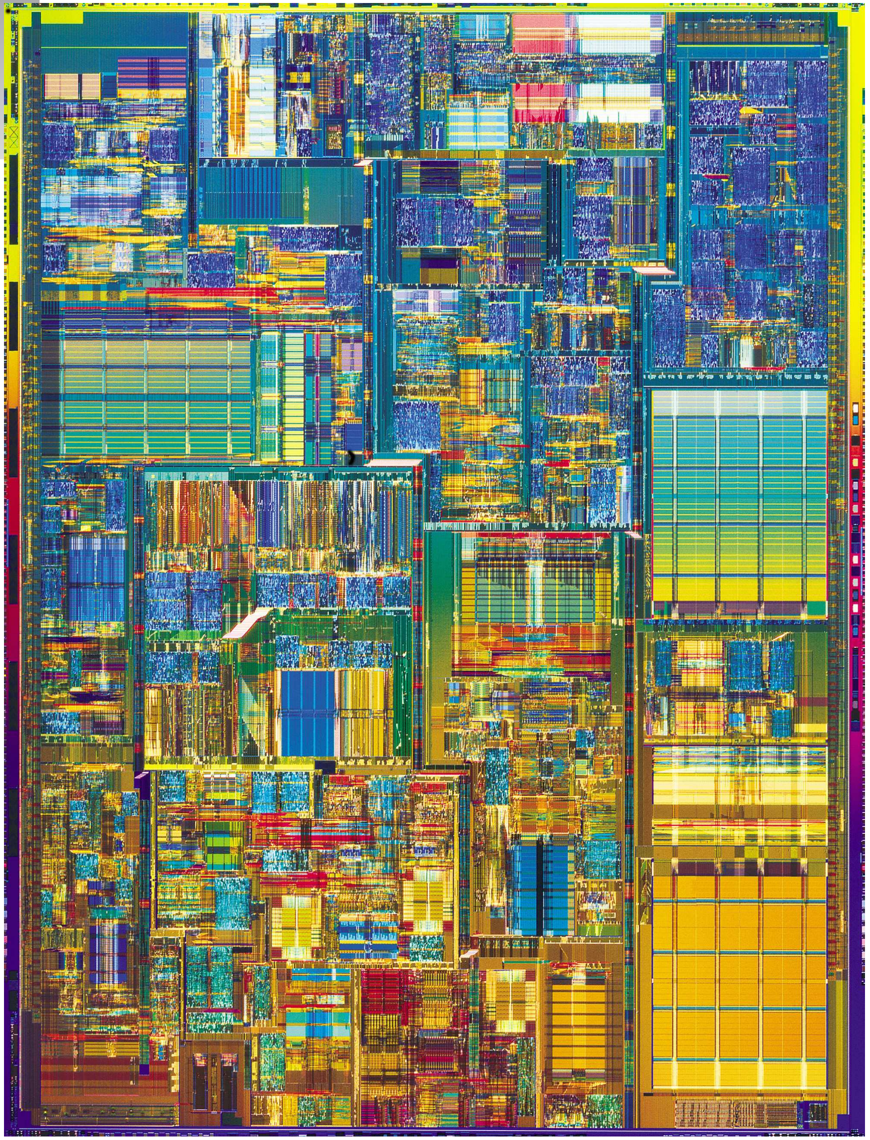
Each logical processor shares SMT-unaware resources:

- Execution Units
- Microarchitectural registers (GPRs, FPRs)
- Caches: trace cache, L1, L2, L3

Sharing:
+ Enables efficient use of resources, but...

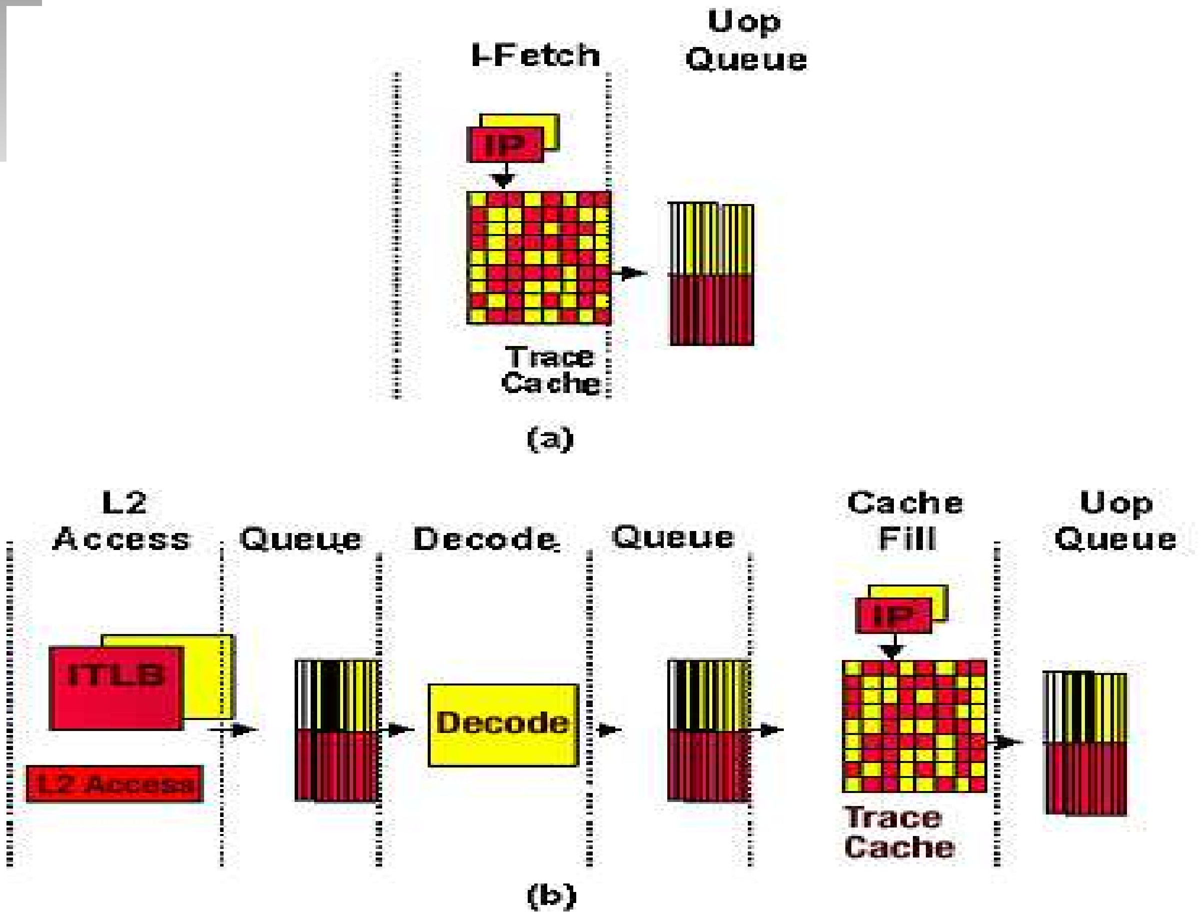
- Allows a thread to monopolize a resource (e.g. cache thrashing).

Pentium 4



- 32-bit
- 2.4 to 3.4 GHz clock frequency
- 800 MHz system bus
- 0.13-micron technology
- 8KB L1 data cache, 12KB L1 instruction cache, 256KB to 1MB L2 cache, 2MB L3 cache
- NetBurst microarchitecture (hyper-pipelined)
- Hyper-Threading technology

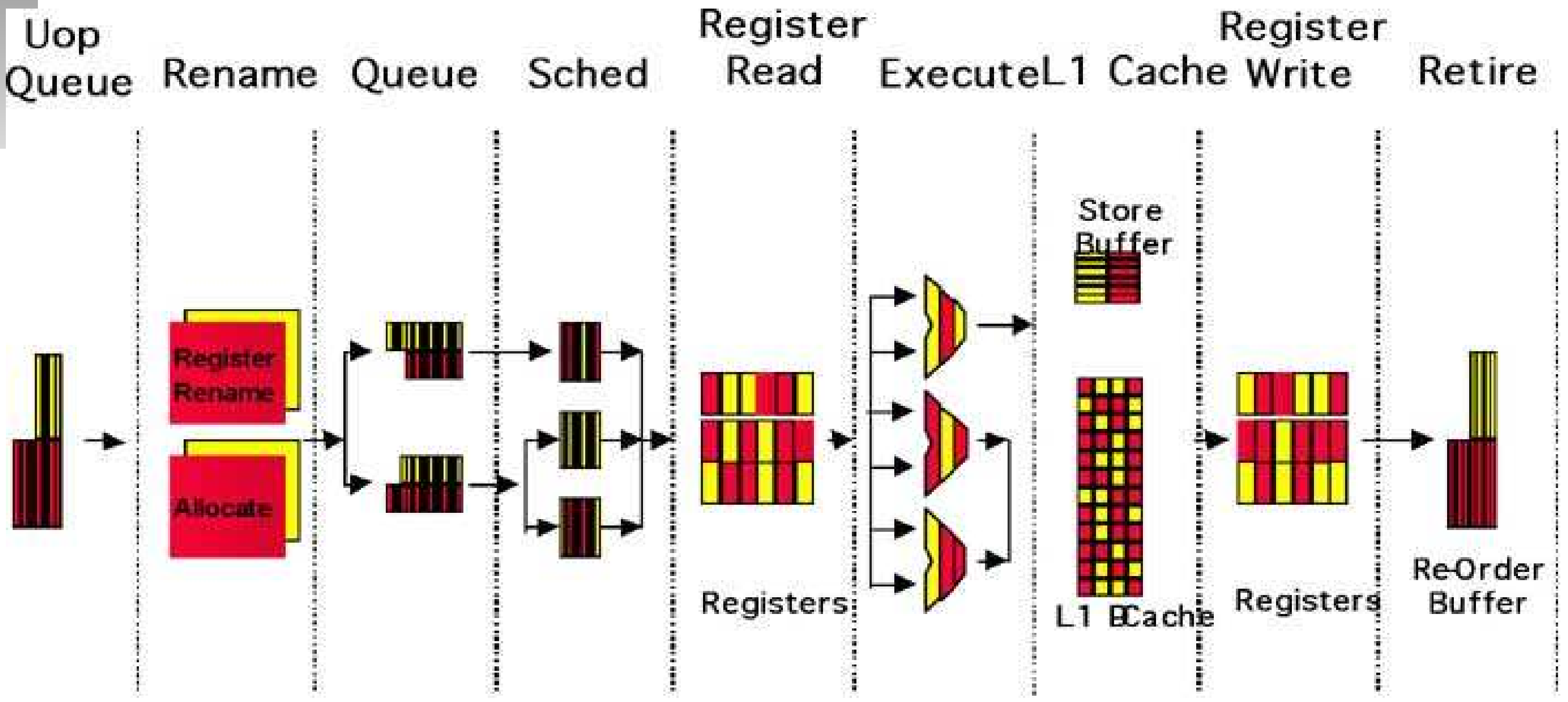
Front-End Pipeline



(a) Trace Cache Hit

(b) Trace Cache Miss

Out-Of-Order Execution Engine Pipeline

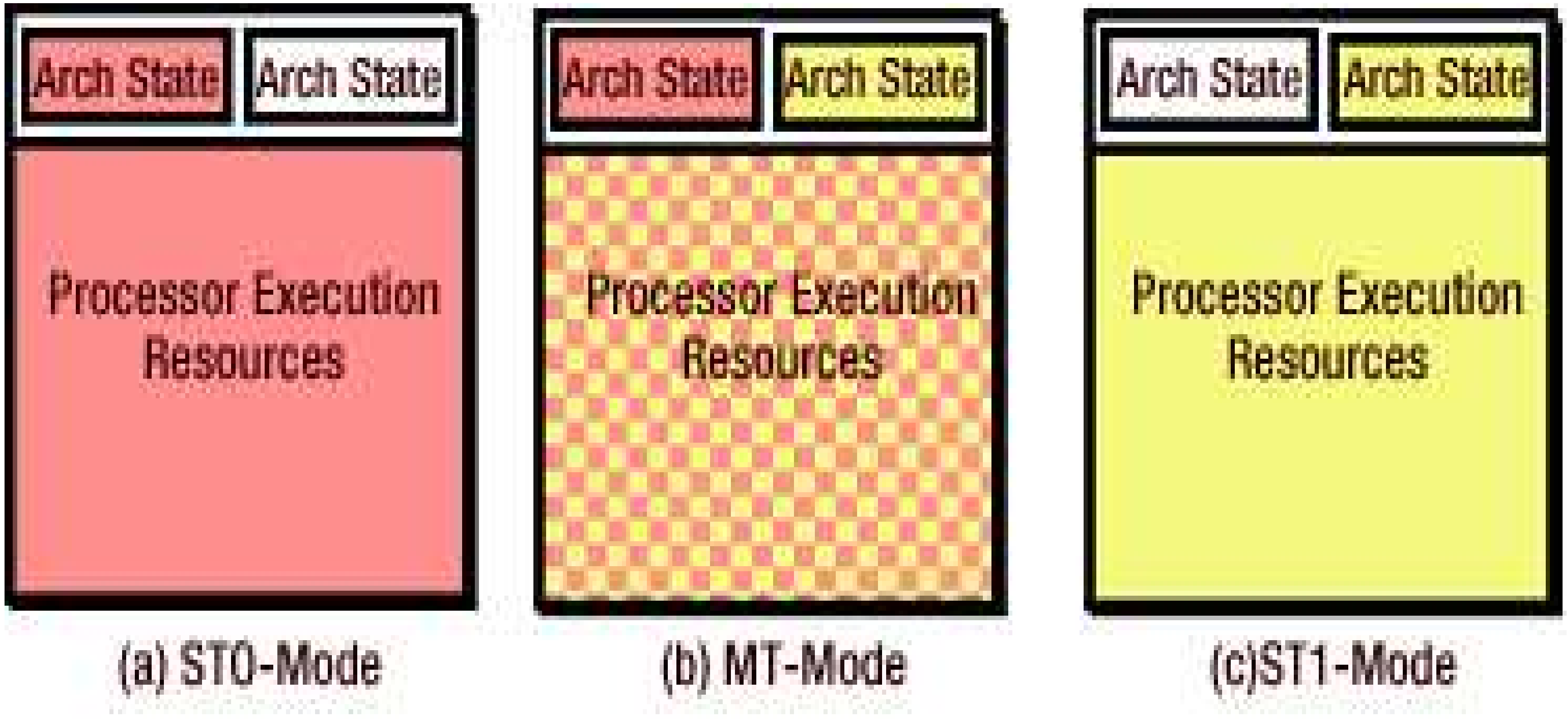


Implementation Goals Achieved

- Minimal die area cost (less than 5% more die area).
- Stall of one logical processor does not stall the other (buffering queues between pipeline logic blocks).
- When only one thread is running, speed should be the same as without H-T (partitioned resources are dedicated to it).

Single- and Multi-Task Modes

Partitioned resources are dedicated to one of the logical processors when the other is HALTed.



Operating System Optimizations

When the OS schedules threads to logical processors it should:

- HALT an inactive logical processor, to avoid wasting resources for idle loops (continuously checking for available work).
- Schedule threads to logical processors on different physical processors instead of the same (when possible), to avoid using the same physical execution resources.

OS Optimizations

The Linux kernel (2.6 series) distinguishes between logical and physical processors:

- H-T-aware passive and active load-balancing
- H-T-aware task pickup
- H-T-aware affinity
- H-T-aware wakeup

Compiler Optimizations

Intel 8.0 C++ and FORTRAN compilers:

Automatic optimizations:

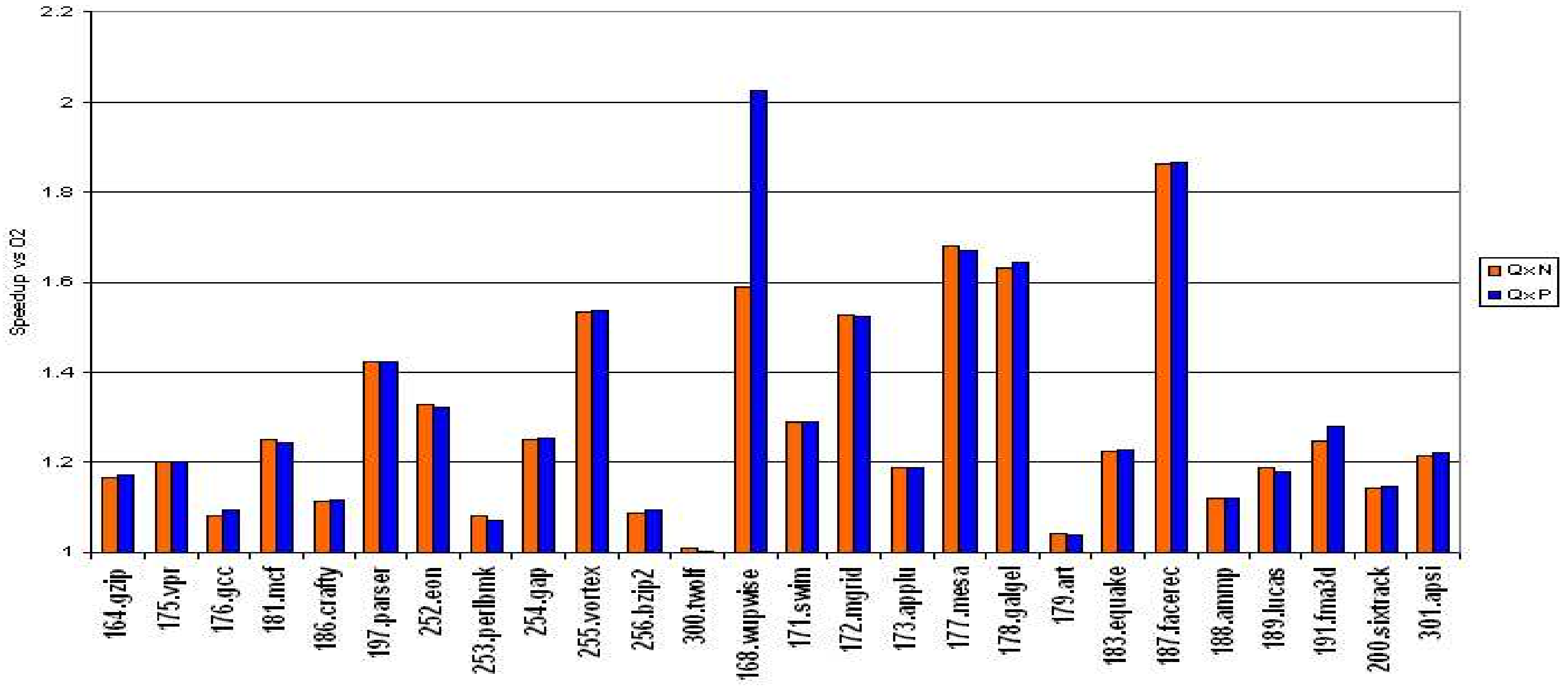
- Vectorization
- Advanced instruction selection

Programmer-controlled optimizations:

- Insertion of Streaming-SIMD-Extensions 3 (SSE3) instructions
- Insertion of OpenMP directives

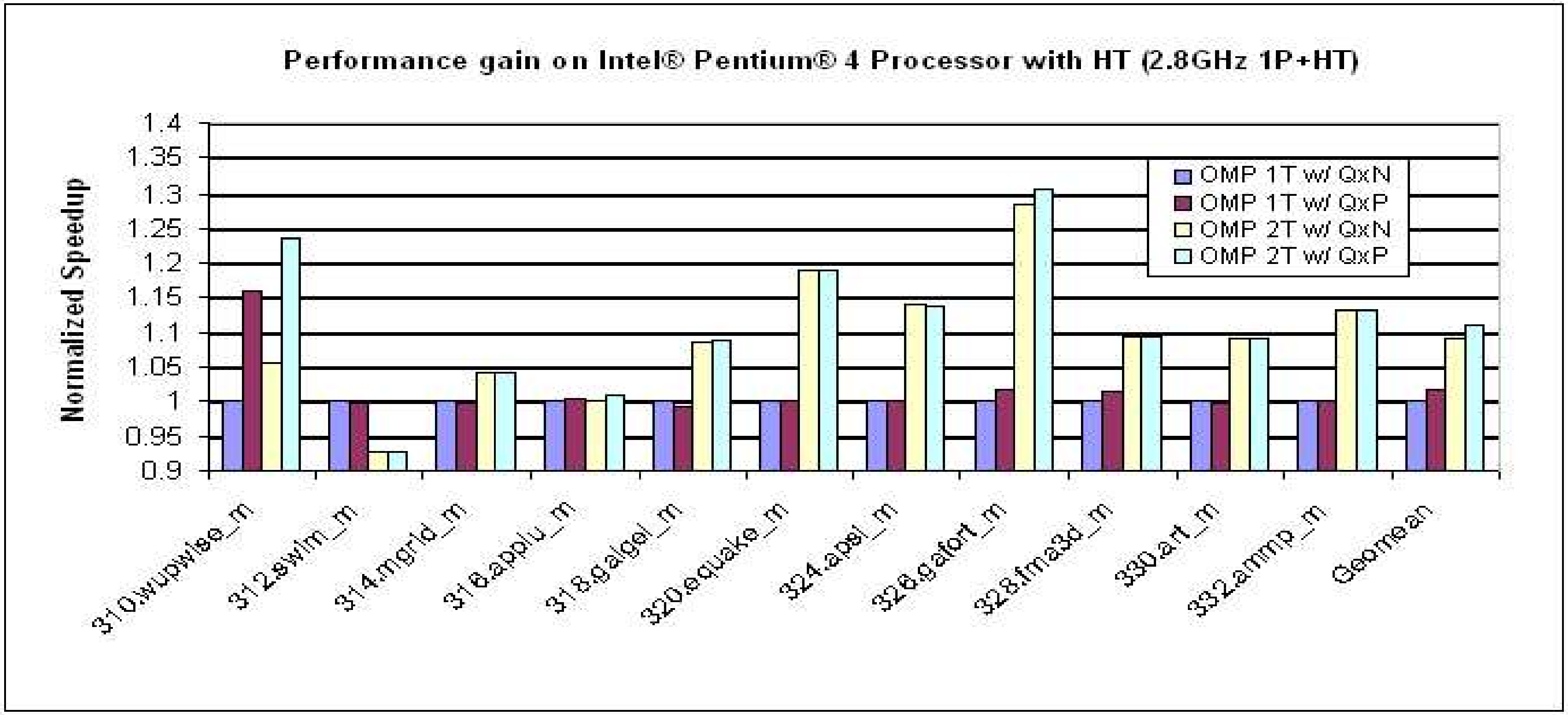
Performance gain from automatic optimizations

SPEC CPU 2000 shows significant speedup not only from H-T specific (QxP) but even for general P4 (QxN) optimizations.



Performance gain from manual optimizations

SPEC OMPM 2001 shows speedup achieved by automatic optimizations in combination with OpenMP directives.

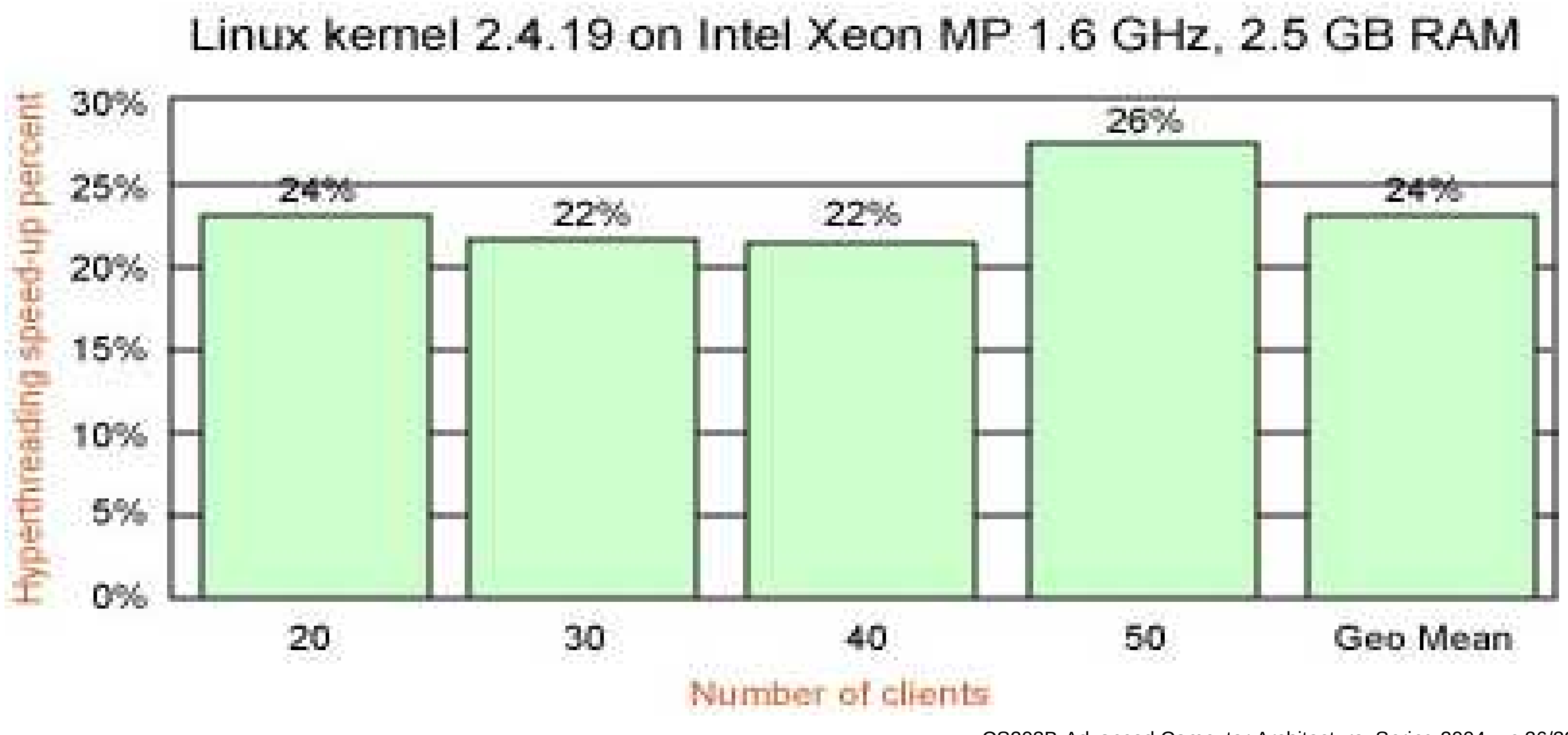


Thread-level Parallelism of Desktop Applications

- Unlike server workloads, interactive desktop applications focus on response time and not on end-to-end throughput.
- Average response time improvement on dual- vs uni-processor measured 22%.
- The application programmer has to exploit multi-threading.
- More than 2 processors yield no great improvements.

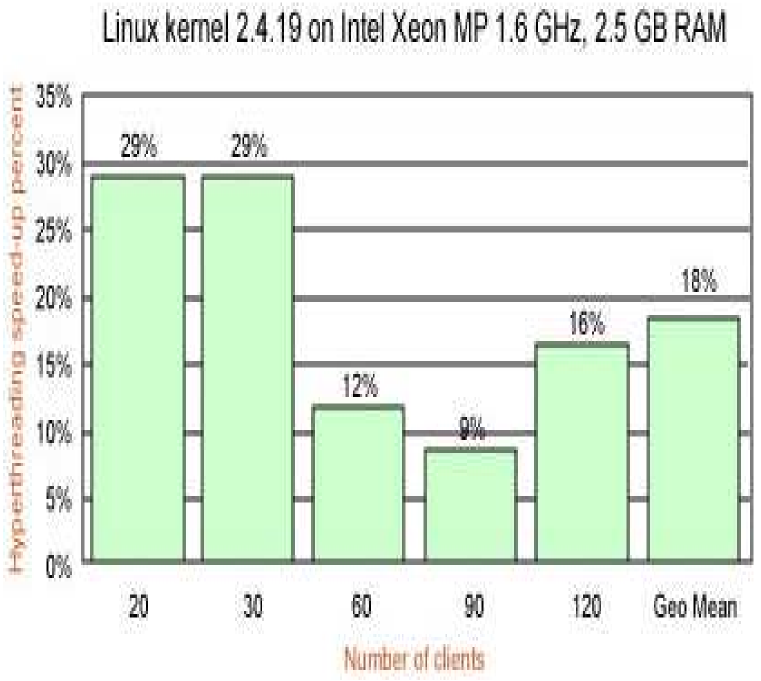
Performance in Client-Server Applications

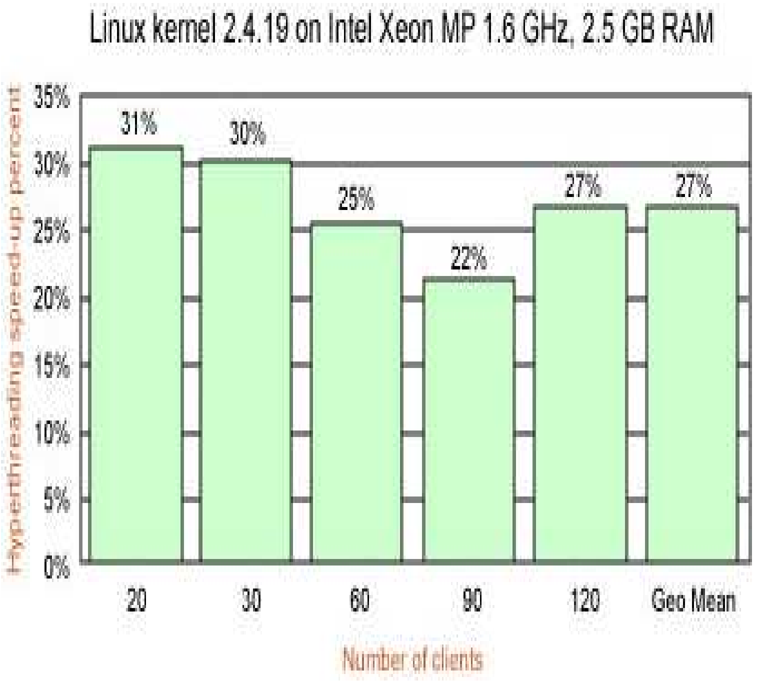
While H-T offers **no gain or degradation** in API calls and user application workloads, it achieves considerable speedups in multi-threaded workloads.



Performance in File Server Workloads

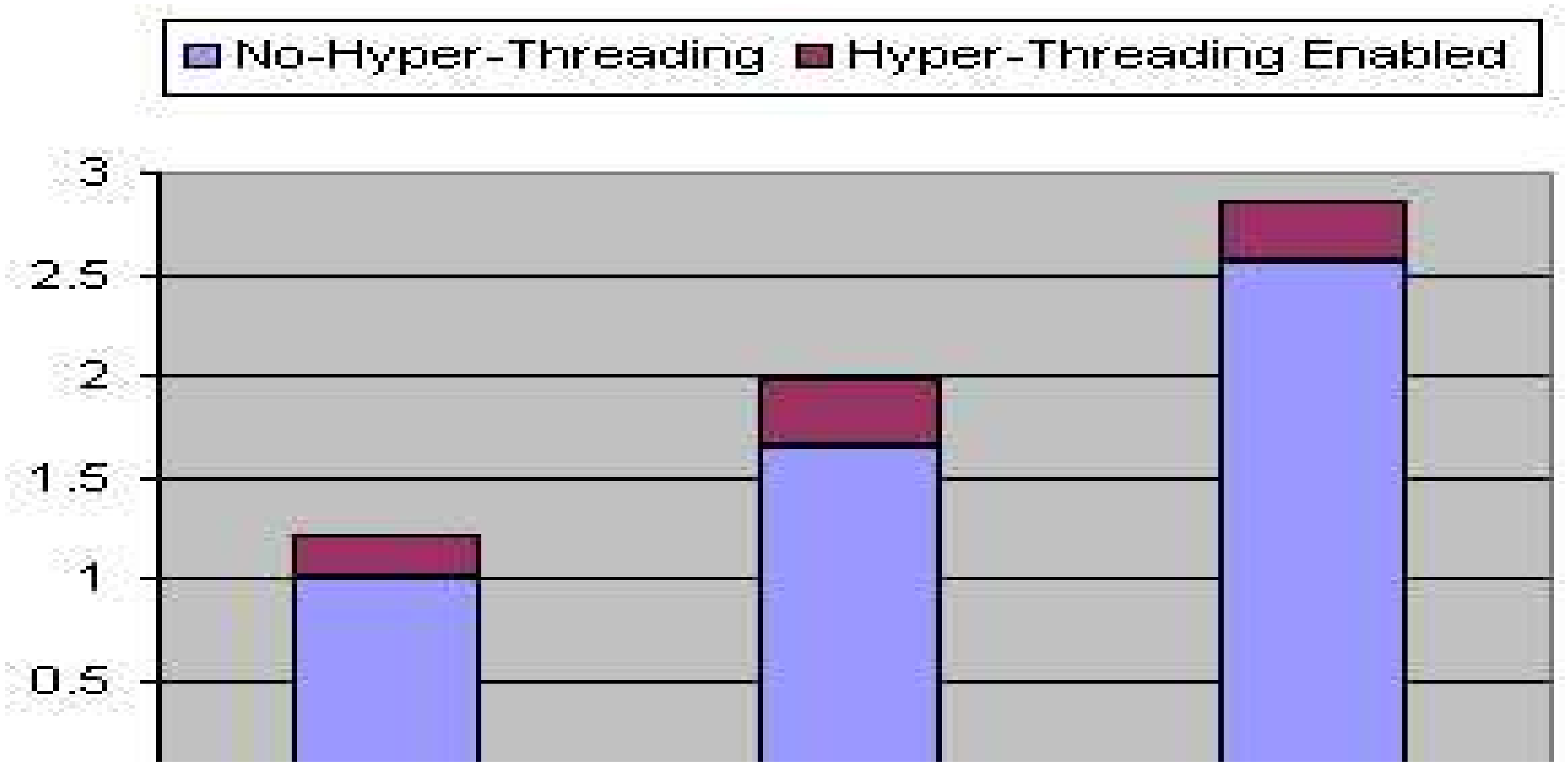
Good speedups in multi-threaded workloads, whether filesystem and socket calls, or just socket calls.





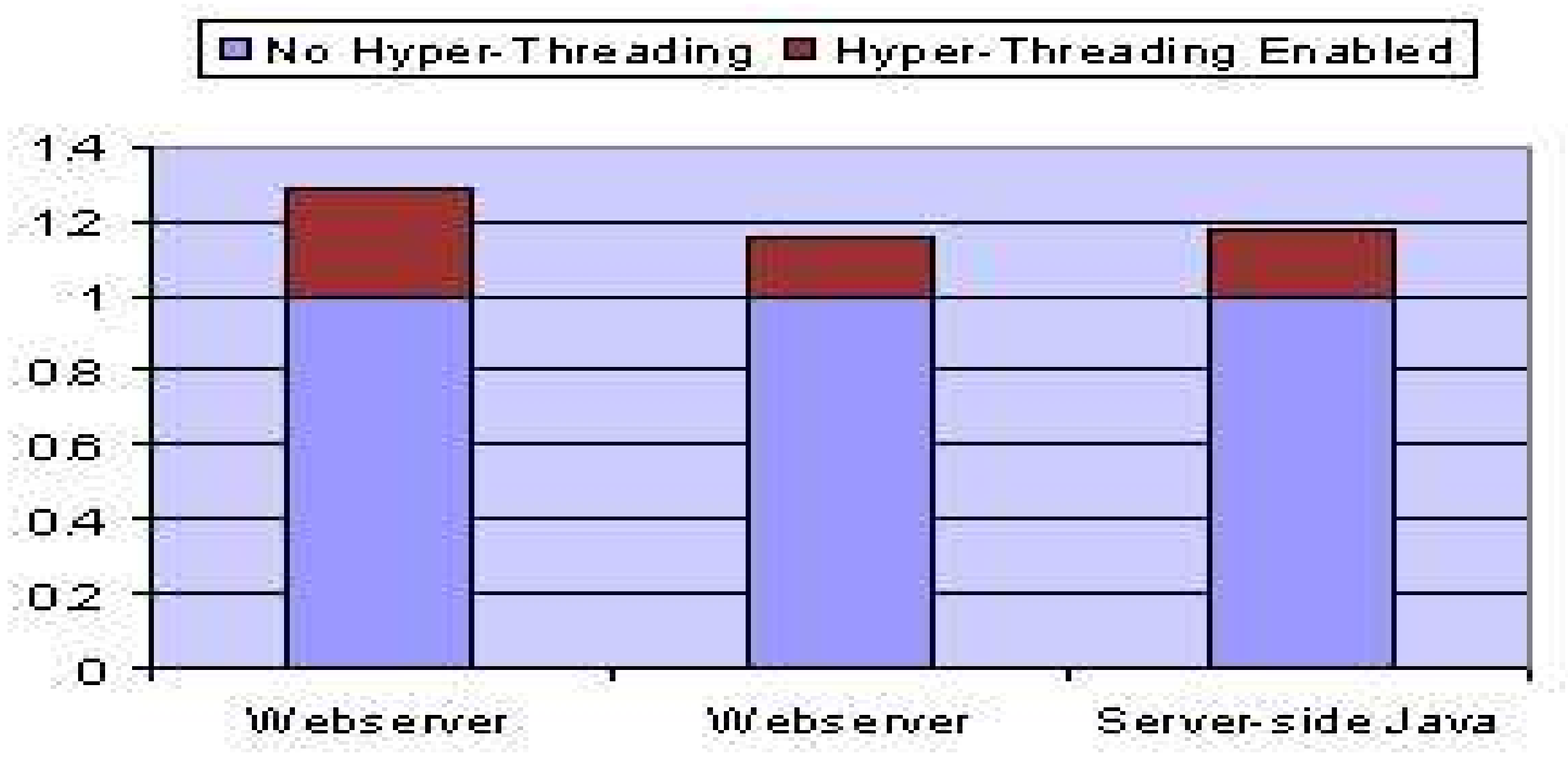
Performance in Online Transaction Processing

21% performance gain in the case of 1 and 2 processors.



Performance in Web Serving

16 to 28% performance gain.



Conclusions

- Hyper-Threading enables thread-level parallelism by duplicating the architectural state of the processor, while sharing one set of processor execution resources.
- When scheduling threads, the OS sees two logical processors.
- While not providing the performance achieved by adding a second processor, Hyper-Threading can offer a 30% improvement.
- Resource contention limits the performance benefits for certain applications.
- Performance gains are evident in multi-threaded workloads, which are usually found in servers.

References

1. D. Marr et al., “Hyper-Threading Technology Architecture and Microarchitecture”, Intel Technology Journal, Volume 06-Issue 01, 2002.
2. D. Tulsen et al., “ Simultaneous Multithreading: Maximizing On-Chip Parallelism”, ISCA, 1995.
3. J. Stokes, “Introduction to Multithreading, Superthreading and Hyperthreading”, Ars Technica, 2002.
4. K. Smith et al., “Support for the Intel Pentium 4 Processor with Hyper-Threading Technology in Intel 8.0 Compilers”, Intel Technology Journal, Volume 08-Issue 01, 2004.
5. D. Vianney, “Hyper-Threading speeds Linux”, IBM Linux developerWorks, 2003.
6. J.Hennessy, D. Patterson, “Computer Architecture: A Quantitative Approach”, 3rd Edition, pp. 608–615, 2003.
7. “Hyper-Threading Technology on the Intel Xeon Processor Family for Servers”, Intel White Paper, 2004.
8. K. Flautner et al., “Thread-level Parallelism and Interactive Performance of Desktop Applications”, ASPLOS, 2000.
9. L. Carter et al., “Performance and Programming Experience on the Tera MTA”

Thank you!

Questions/Comments?