Hardware Parallelism: Are Operating Systems Ready?

(Case Studies in Mis-Scheduling)

Eitan Frachtenberg
Los Alamos National Labs
eitanf@lanl.gov

Yoav Etsion
Hebrew University
etsman@cs.huji.ac.il

Hardware is Going Parallel

Hardware Parallelism is reaching the desktop

- Intel's CoreDuo
- AMD's Athlon64X2

It makes the operating system's life difficult:

- Complicates process scheduling
- Complicates resource management
- More locking
- More locking overhead

We'll focus on process scheduling...

Software is Going Parallel

Parallelism is already trickling to software:

- Servers (Web, DB, File)
- Applications (POSIX PThreads)

This trend is expected to grow as we witness advances in related research topics:

- Compiler level parallelism
- Locking mechanisms
- Transactional Memory

Parallel Software Workloads

Still too few applications to characterize...

...but enough to see there's a problem!

We'll use Flynn's categorization as a base:

- ***SIMD***: Workpile model
- ***MISD***: Systolic/Pipelined model
- ***MIMD***: Bulk-Synchronous model

Parallel Workloads, Serial Schedulers?

Scheduling parallel workloads is done in HPC

- Has not found its way to the desktop

Schedulers treat each process independently:

- Ignore inter-process dependencies
- Only attempt to balance the load between processors

Result: *Missing the Big Picture*

- *Separating interfering processes*
- *Co-Scheduling collaborating processes*

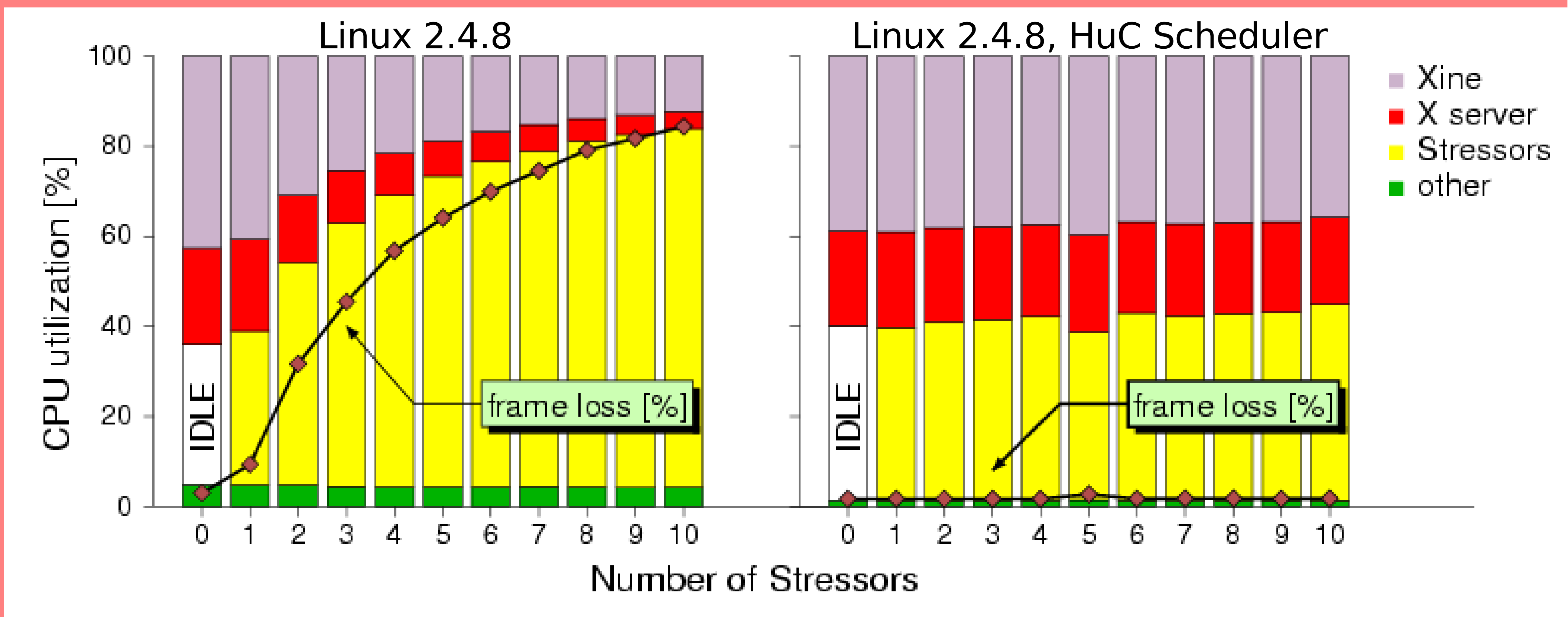
Consequence:

- Only the *Workpile* model is supported (and not well)

Pipelined App. vs. Uniprocessor: *A Smart Scheduler Can Do It!*

Xine multi-threaded player + increasing load

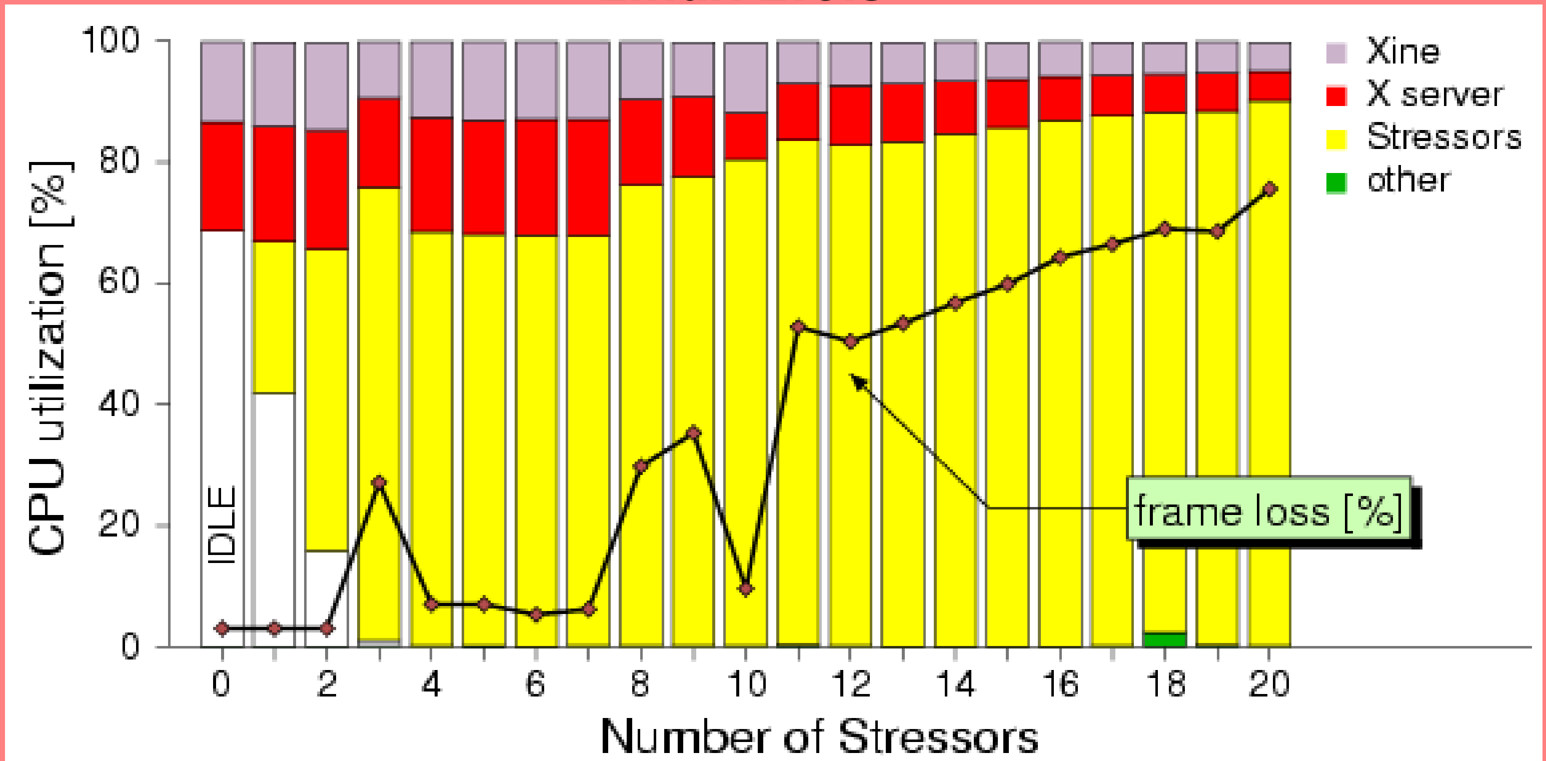
- App. Stages: Decoder + Displayer threads
- Hidden pipeline stage: X Windows server



Pipelined App. vs. Multiprocessor
How Good is the Scheduler?

- *Xine* running on a 4-way SMP
  - No single CPU can satisfy both *Xine* and the *X Server*
- Result: dependency incognizant scheduler migrates performance away…

Linux 2.6.9

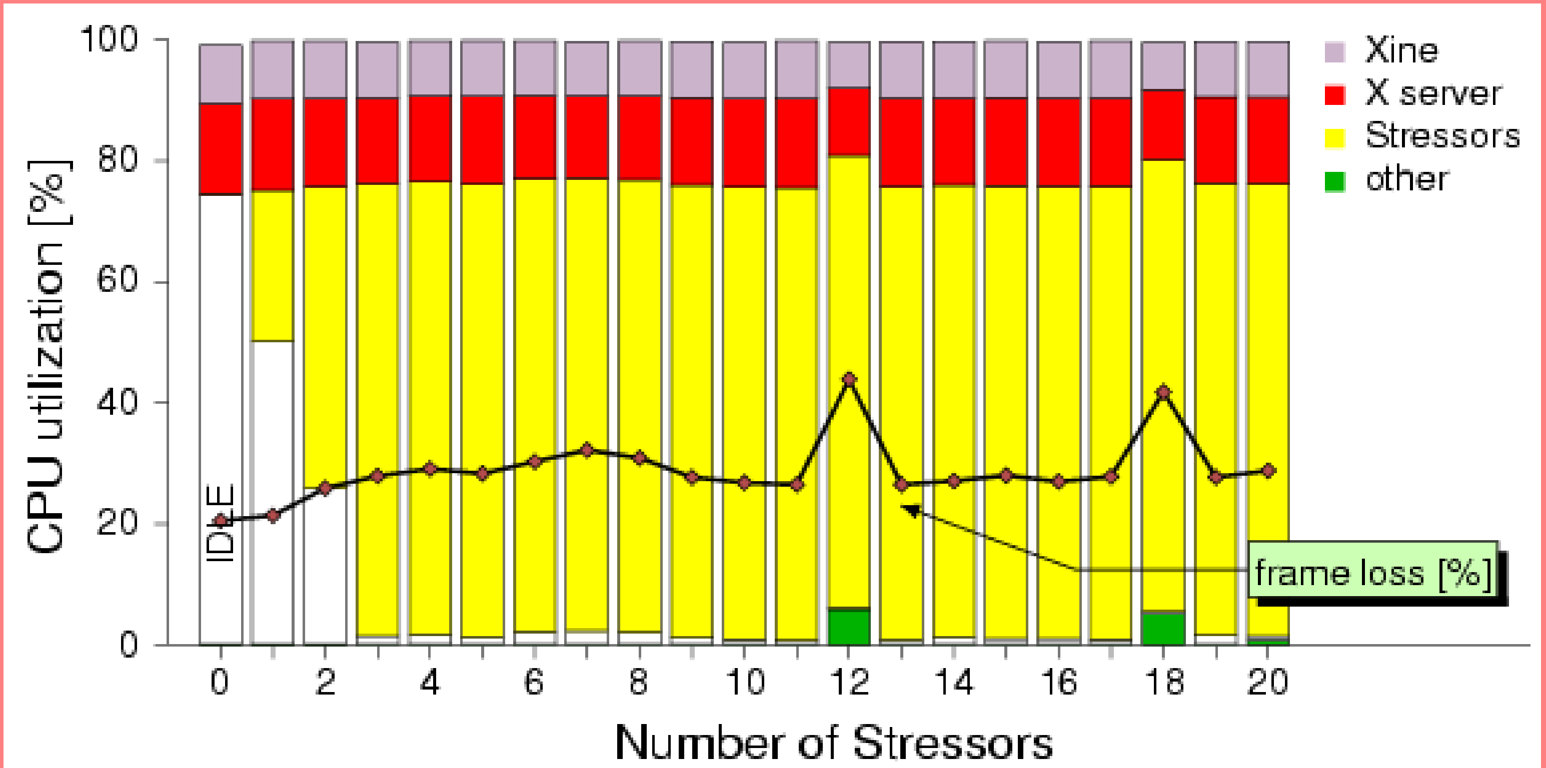


Pipelined App. vs. Multiprocessor: *Processor Containment*

This time we locked Xine+X on CPU 0

- Stressors limited to CPUs 1-3

Result: More consistent results, but poor performance.

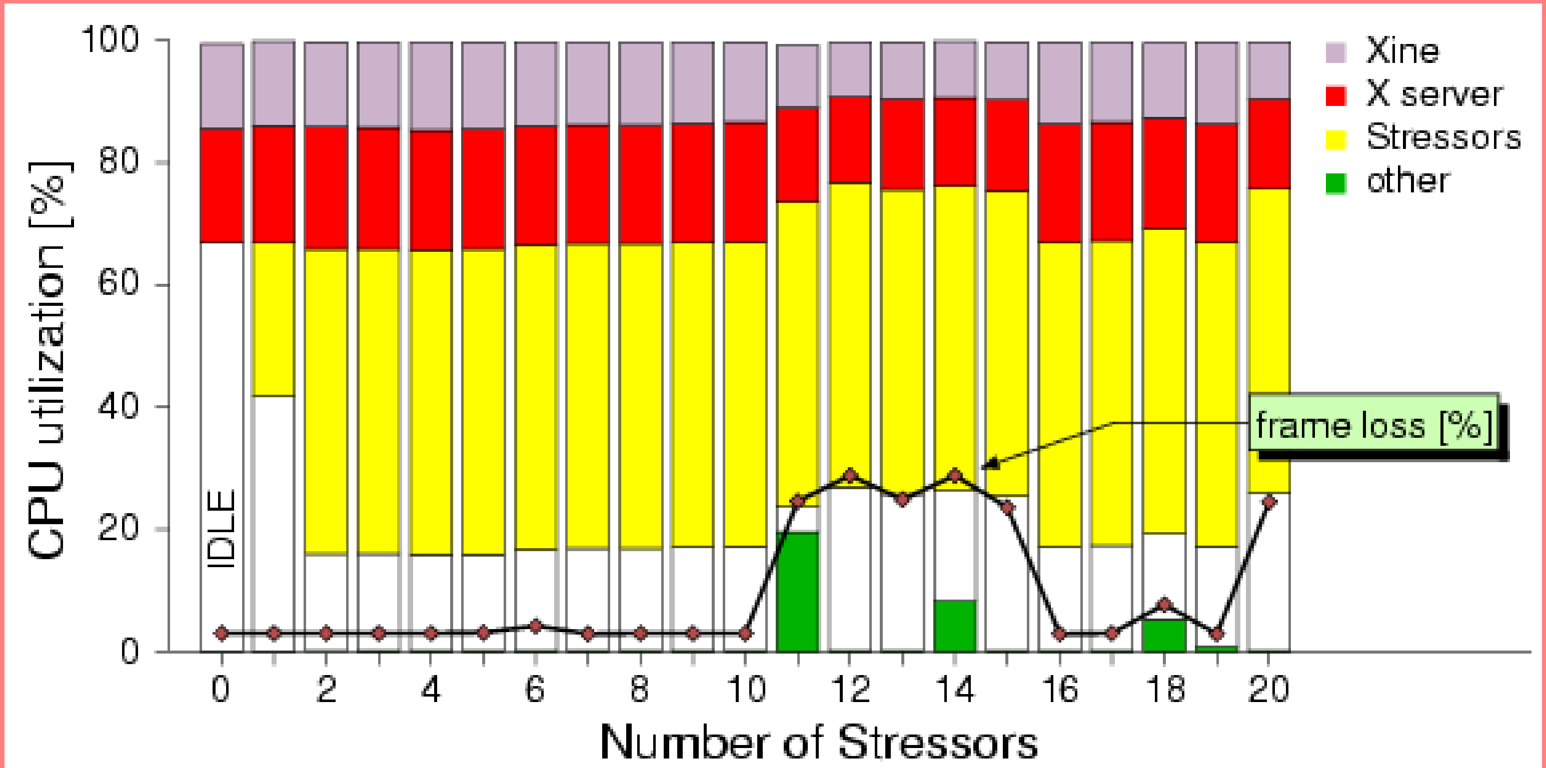


Pipelined App. vs. Multiprocessor: *Can the Scheduler Take a Hint?*

Hmm... let's try again: lock Xine+X on CPU 0-1

- Stressors limited to CPUs 2-3

<u>Result</u>: Better, but scheduler still gets confused when system daemons wake up.

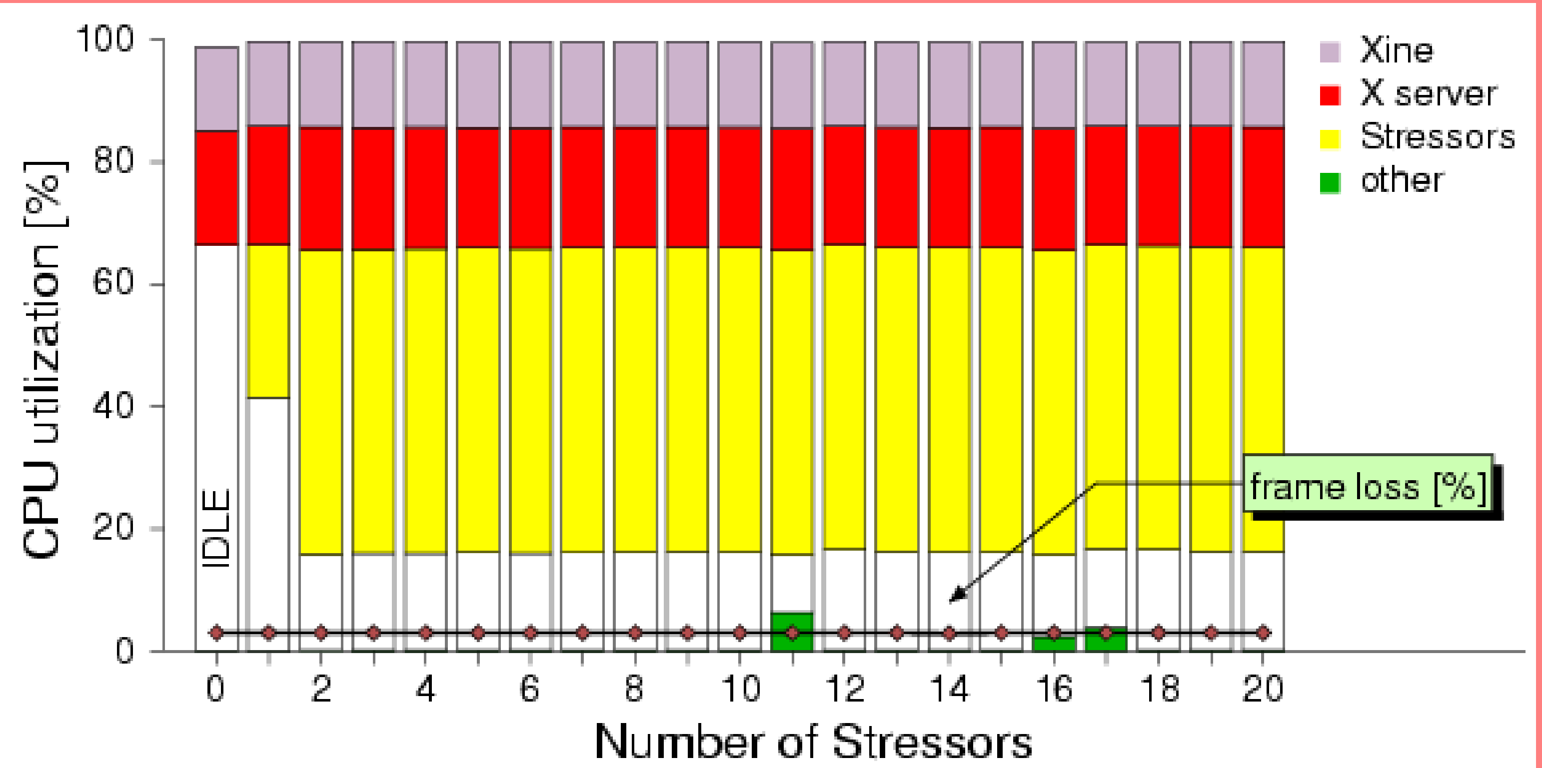


Pipelined App. vs. Multiprocessor: *Manual Scheduling*

OK, we'll do it ourselves: Xine on CPU 0, X on 1

- Stressors limited to CPUs 2-3

Result: Good results, but we had to manually configure the scheduler...



Pipelined App. vs. Multiprocessor: *Conclusions*

The scheduler's only guideline is load balancing:

- Not all jobs are equal, so imbalance can be useful

OS must be cognizant of whole job semantics

- Poses a counter force against load balancing
- Otherwise an application might compete against itself

Partial knowledge about a job does not help

- Invalidates APIs that enable application specific hints
- Calls for a deductive scheduler solution

Bulk Sync. vs. Multiprocessor

A synthetic bulk synchronous application:

- Processes perform some computation
- synchronize every few hundred iterations
- Using P-1 processes on P-way machine

Defaults scheduler vs. gang scheduling:

- One parallel job active at any time + CPU stressors

Metric: Jobs' completion time

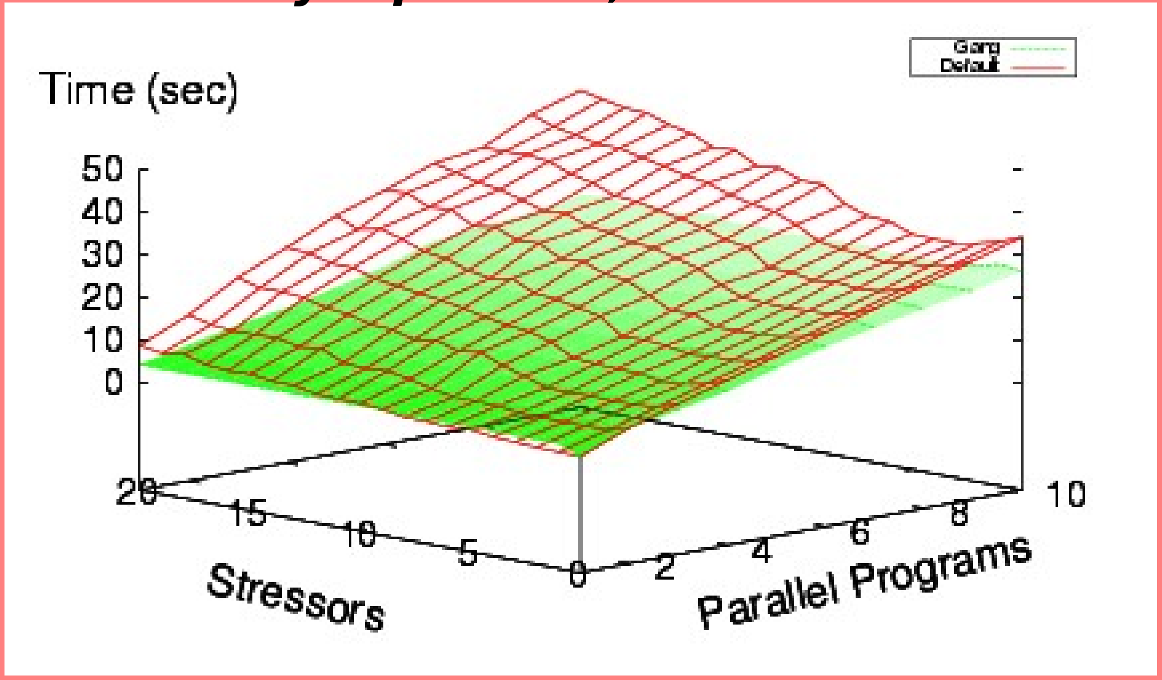
Goal: Evaluate the effect of the OS scheduler's incognizance on performance

Bulk Sync. vs. Multiprocessor: *Why Gang Scheduling is Better*

Base case: comparing default scheduler with gang scheduling

We see how the gap between surfaces increases with the load...

4-way Alpha EV6, Linux 2.4.21



Bulk Sync. vs. Multiprocessor: *Adequacy of Linux schedulers*

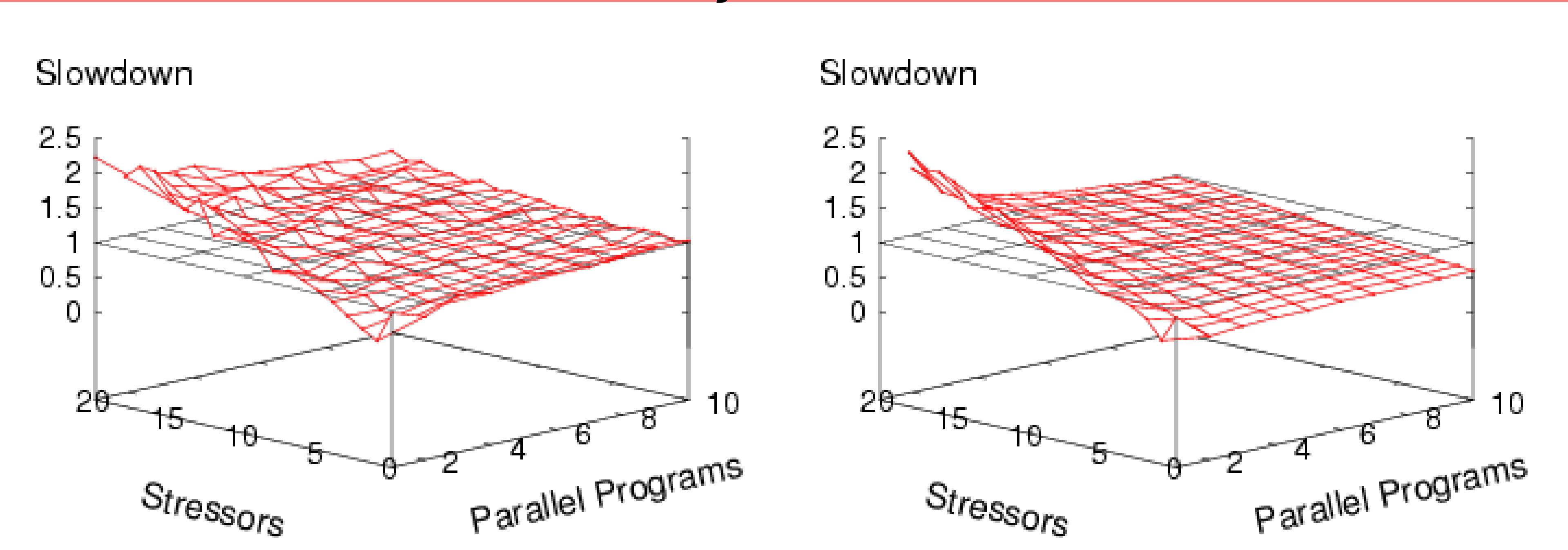
The Linux 2.6 scheduler targets SMPs:

- Per-CPU run queue, load balancing

But Improvement is attributed to multimedia...

(Results shown as slowdown compared to GS)

Linux 2.4.22 ***4-way Pentium3*** ***Linux 2.6.9***



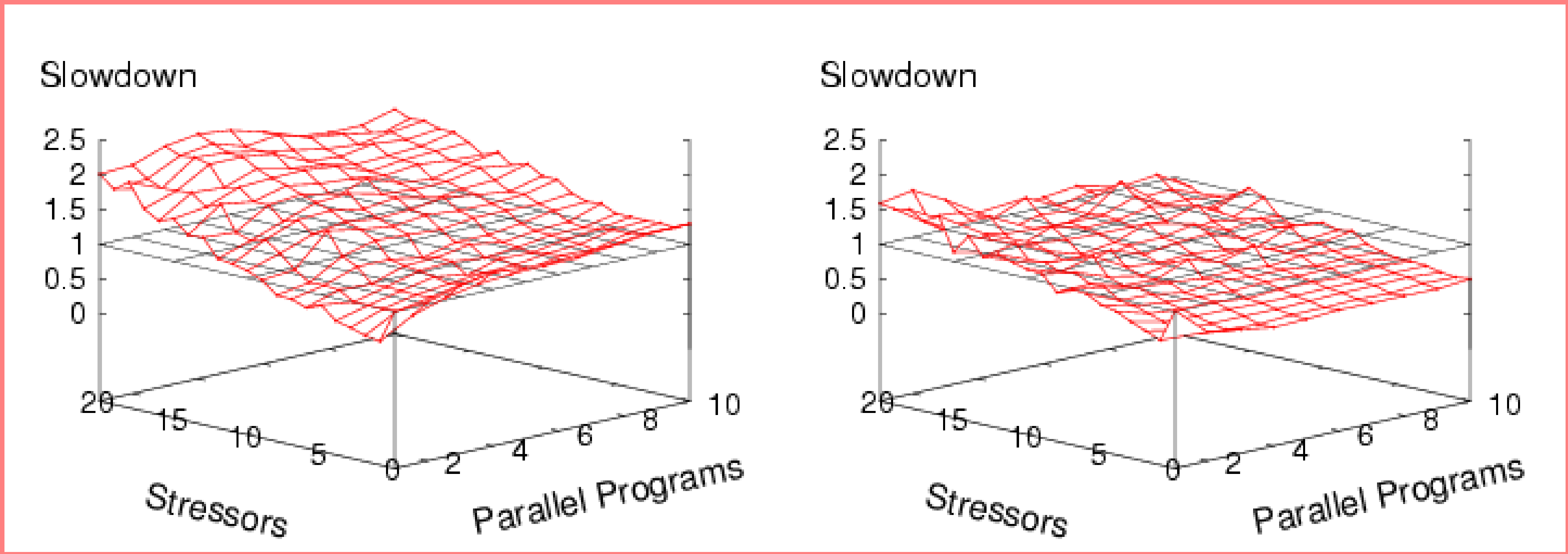
Bulk Sync. vs. Multiprocessor: *Industrial Strength or Desktop OS?*

Similar hardware, different operating systems:

- Industrial strength OS: Tru64
- Desktop OS: Linux 2.6

Tru64 is usually even better than GS, but inconsistent

Linux 2.4.21 ***4-way Alpha EV6*** ***Tru64 5.1***



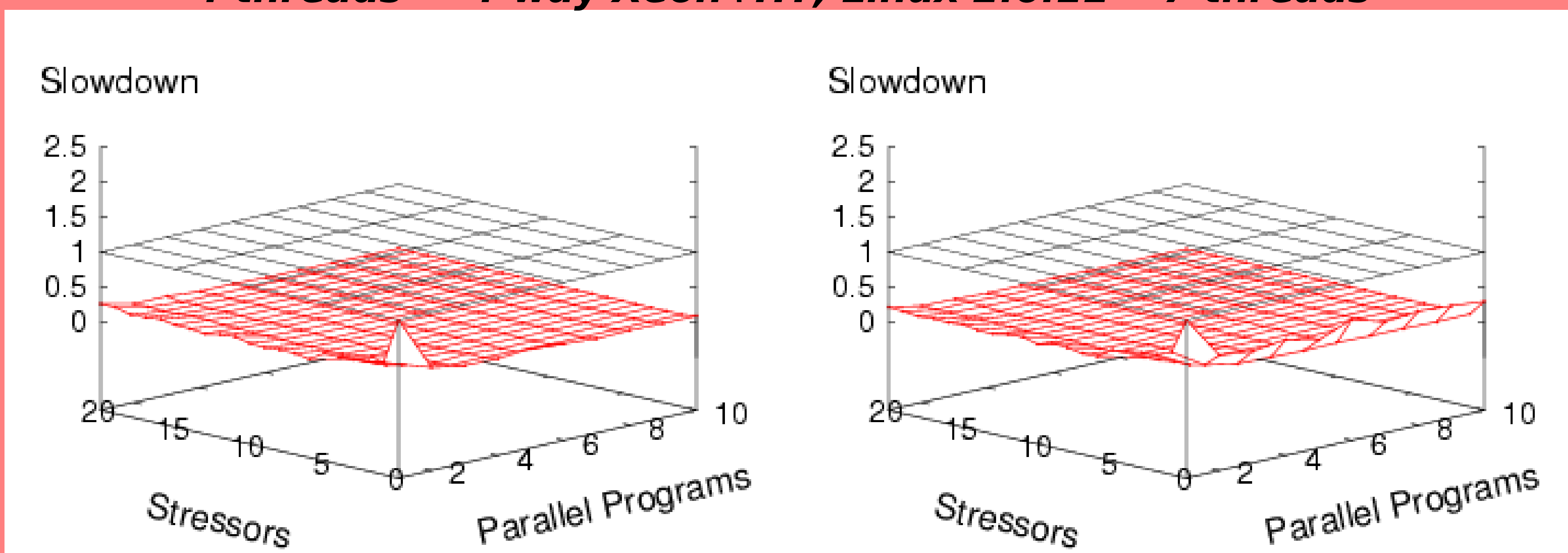
Bulk Sync. vs. Multiprocessor: *Combining SMP with SMT*

Gang scheduling is not the answer on SMTs

System with 4 physical/8 logical processors:

- 4-way parallelism: OS has ample resources
- 7-way parallelism: Job competes against itself anyway

4 threads ***4-way Xeon+HT, Linux 2.6.11*** ***7 threads***



Bulk Sync. vs. Multiprocessor: Conclusions

Desktop schedulers have problems mixing parallel and sequential loads.

- Auto-Parallelism can seriously suffer from this

Co-Scheduling can go a long way:

- Even in its rigid form of gang scheduling

SMT complicates process placement:

- The OS must be aware of resource sharing

Conclusions

Scheduling serial programs remained in 1970s

- Stagnation was compensated by CPU speeds

Scheduling parallel programs is used in HPC

- But mostly in homogeneous environments

Guidelines for combining parallel and serial workloads:

- ***Maximize Collaboration***: *co-schedule if needed*
- ***Minimize Interference***: *imbalance can be useful*

The Road Ahead

Scheduling 101: characterize the workload

- Are Flynn's models still relevant?

Scheduling metrics:

- Is CPU consumption still a relevant metric?
- How do we account for concurrent consumption?
- Handling heterogeneous workloads

The scheduler needs to know about grouping

- Explicitly, or will tracking IPC suffice?

Does SMT pose more trouble that its worth?