

Designing OS for HPC Applications: Scheduling

Roberto Gioiosa¹, Sally A. McKee², Mateo Valero³

¹Barcelona Supercomputing Center

²Chalmers University of Technology

³Universitat Politècnica de Catalunya

roberto.gioiosa@bsc.es, mckee@chalmers.se, mateo@ac.upc.edu

IEEE Cluster 2010

IEEE International Conference on Cluster Computing 2010

20 - 24 September 2010 - Heraklion, Crete, Greece



Heraklion (GR), 21st Sep 2010



Outline

- 1 Introduction
- 2 Execution time variability
- 3 High Performance Linux
- 4 Experimental results
- 5 Summary



Outline

- 1 Introduction
- 2 Execution time variability
- 3 High Performance Linux
- 4 Experimental results
- 5 Summary



System designing

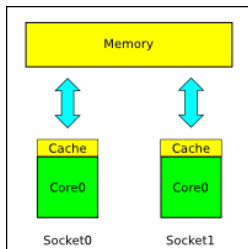
Operating systems have been developed as independent layers between hardware and applications.

Except for specific interfaces, layers are practically oblivious of each other.



This approach provides portability and transparency, but may not provide optimal performance.

CMP: Parallelism gone wild!

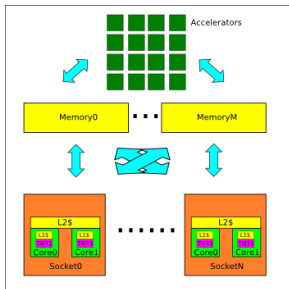
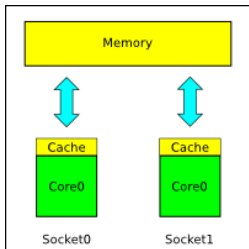


- CMPs share different levels of internal hardware resources
- With CMPs, each compute node runs several processes
- Parallel applications have evolved and use a mix of different programming models (MPI, OpenMP, UPC, OpenCL, CUDA)

Have OS evolved with hardware and applications?



CMP: Parallelism gone wild!



- CMPs share different levels of internal hardware resources
- With CMPs, each compute node runs several processes
- Parallel applications have evolved and use a mix of different programming models (MPI, OpenMP, UPC, OpenCL, CUDA)

Have OS evolved with hardware and applications?



The HPC world

In current HPC systems:

- sub-optimal performance is... not an option!
- power efficiency is a main design constraint
- low system utilization means money loss

Two possible approaches:

LW kernels

- Good performance/scalability - low OS noise
- Applications/Hardware specific
- **Limited number of services**

Monolithic kernels

- **Limited performance/scalability - high OS noise**
- Applications/Hardware independent
- Large number of services

Shall we go bottom-up or top-down?

Both approaches are fine but the OS needs more information on hardware and applications!



The HPC world

In current HPC systems:

- sub-optimal performance is... not an option!
- power efficiency is a main design constraint
- low system utilization means money loss

Two possible approaches:

LW kernels

- Good performance/scalability - low OS noise
- Applications/Hardware specific
- **Limited number of services**

Monolithic kernels

- **Limited performance/scalability - high OS noise**
- Applications/Hardware independent
- Large number of services

Shall we go bottom-up or top-down?

Both approaches are fine but the OS needs more information on hardware and applications!

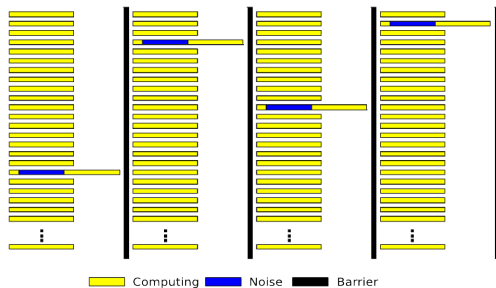


Outline

- 1 Introduction
- 2 Execution time variability**
- 3 High Performance Linux
- 4 Experimental results
- 5 Summary



OS noise

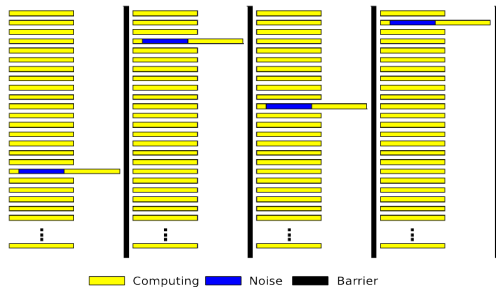


There are many sources of OS noise:

- Interrupts
- User and kernel daemons
- Scheduling
- Memory management

[E. Betti, M. Cesati, R. Gioiosa, F. Piermaria, "A Global Operating System for HPC Clusters" in *CLUSTER09*, August 2009].

OS noise

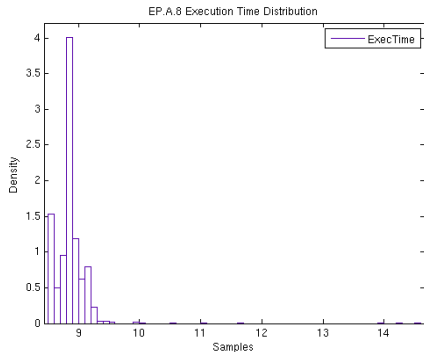


There are many sources of OS noise:

- Interrupts
- User and kernel daemons
- Scheduling
- Memory management

[E. Betti, M. Cesati, R. Gioiosa, F. Piermaria, "A Global Operating System for HPC Clusters" in *CLUSTER09*, August 2009].

Execution time variation



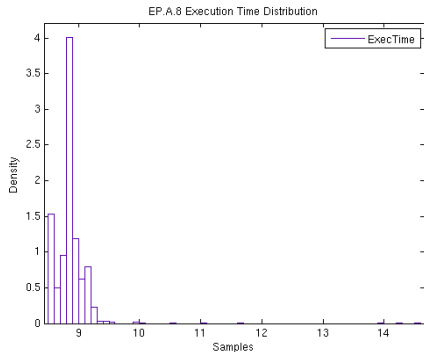
Linux scheduler

- NAS EP, class A, 8 MPI process
- 1000 repetitions

- Sensitive to OS noise (Small dataset)
- Easy to compose (embarrassing parallel)
- Execution time variation is way larger than expected



Execution time variation



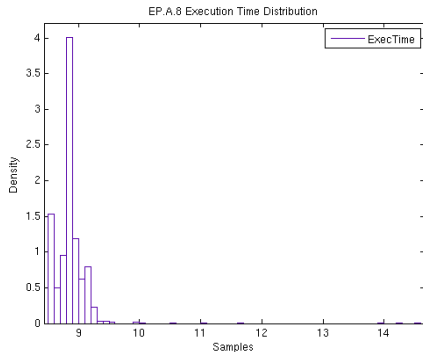
Linux scheduler

- NAS EP, class A, 8 MPI process
- 1000 repetitions

- Sensitive to OS noise (Small dataset)
- Easy to compose (embarrassing parallel)
- Execution time variation is way larger than expected



Execution time variation



Linux scheduler

- NAS EP, class A, 8 MPI process
- 1000 repetitions

- Sensitive to OS noise (Small dataset)
- Easy to compose (embarrassing parallel)
- Execution time variation is way larger than expected



NAS benchmarks ET variation

Bench	Min	Avg	Max	Var %
<i>cg.A.8</i>	0.69	1.04	46.69	6608.70
<i>cg.B.8</i>	36.98	42.04	126.48	242.02
<i>ep.A.8</i>	8.54	8.87	14.59	70.84
<i>ep.B.8</i>	34.14	34.69	53.34	56.24
<i>ft.A.8</i>	2.27	2.50	9.07	327.31
<i>ft.B.8</i>	22.56	22.91	41.78	85.20
<i>is.A.8</i>	0.35	0.57	3.27	832.29
<i>is.B.8</i>	1.82	1.88	4.82	164.84
<i>lu.A.8</i>	17.56	19.34	50.85	189.58
<i>lu.B.8</i>	71.93	79.37	140.03	94.68
<i>mg.A.8</i>	1.40	1.60	7.80	457.14
<i>mg.B.8</i>	4.48	4.96	28.35	532.81

- simple applications (*ep*) show ET variability up to 70%
- more complex applications (*cg*) show extremely high ET variability



NAS benchmarks ET variation

Bench	Min	Avg	Max	Var %
<i>cg.A.8</i>	0.69	1.04	46.69	6608.70
<i>cg.B.8</i>	36.98	42.04	126.48	242.02
<i>ep.A.8</i>	8.54	8.87	14.59	70.84
<i>ep.B.8</i>	34.14	34.69	53.34	56.24
<i>ft.A.8</i>	2.27	2.50	9.07	327.31
<i>ft.B.8</i>	22.56	22.91	41.78	85.20
<i>is.A.8</i>	0.35	0.57	3.27	832.29
<i>is.B.8</i>	1.82	1.88	4.82	164.84
<i>lu.A.8</i>	17.56	19.34	50.85	189.58
<i>lu.B.8</i>	71.93	79.37	140.03	94.68
<i>mg.A.8</i>	1.40	1.60	7.80	457.14
<i>mg.B.8</i>	4.48	4.96	28.35	532.81

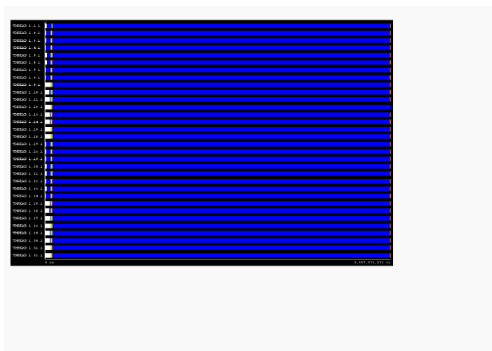
- simple applications (*ep*) show ET variability up to 70%
- more complex applications (*cg*) show extremely high ET variability



OS noise resonance

Some applications are more sensitive than others to OS noise, according to:

- 1 Application *complexity*
- 2 Noise *signature*

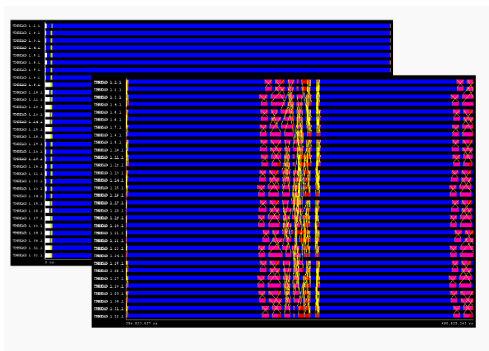


OS noise has greater impact on scalability if it resonates with the application.

OS noise resonance

Some applications are more sensitive than others to OS noise, according to:

- 1 Application *complexity*
- 2 Noise *signature*



OS noise has greater impact on scalability if it resonates with the application.

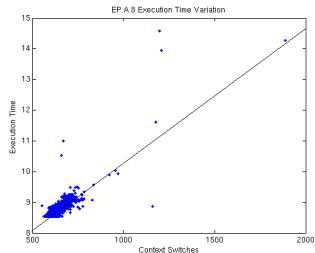
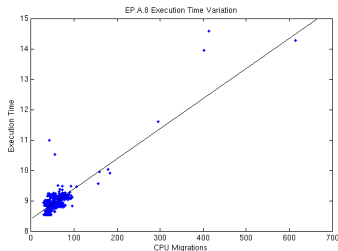
Under the hood

Bench	CPU Migrations			Context Switches		
	Min.	Avg.	Max.	Min.	Avg.	Max.
<i>cg.A.8</i>	30	63.61	2078	460	602.57	5755
<i>cg.B.8</i>	28	90.62	3499	1726	2011.80	8243
<i>ep.A.8</i>	29	52.41	615	550	652.62	1886
<i>ep.B.8</i>	28	69.02	2536	1198	1333.70	5239
<i>ft.A.8</i>	20	53.02	565	318	575.10	1609
<i>ft.B.8</i>	28	51.23	1163	1111	1222.50	3258
<i>is.A.8</i>	29	52.18	160	396	537.35	956
<i>is.B.8</i>	28	52.88	370	519	610.93	1213
<i>lu.A.8</i>	18	70.79	1368	219	1030.40	3870
<i>lu.b.8</i>	29	69.04	3657	2518	2933.50	9131
<i>mg.A.8</i>	29	54.73	590	91	556.24	1776
<i>mg.B.8</i>	29	54.68	853	531	660.43	2396

The OS scheduler frequently migrates and/or preempts processes.



Source of ET variation



There seems to be a correlation between execution time variation and

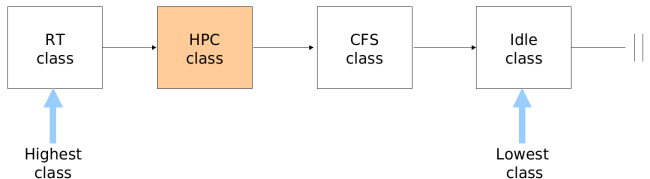
- CPU migrations
- process preemption

Outline

- 1 Introduction
- 2 Execution time variability
- 3 High Performance Linux**
- 4 Experimental results
- 5 Summary



HPL scheduler



High Performance Linux scheduler has been designed considering:

- 1 applications' requirements and common problems
- 2 hardware topology and characteristics

We introduced a new scheduling class (`SCHED_HPC`):

- We do not change the behavior of other tasks/daemons
- Critical drivers' components are still allowed to run



HPL scheduler features

Property/problem	Solution
CPU migrations	Smart load balancing
Process preemption	Class prioritization
Responsiveness	Class prioritization
Load balancing	Architecture characterization
Representativeness	Application characterization
Easy-to-use	<code>sched_setscheduler</code> <code>chrt</code>

How to run

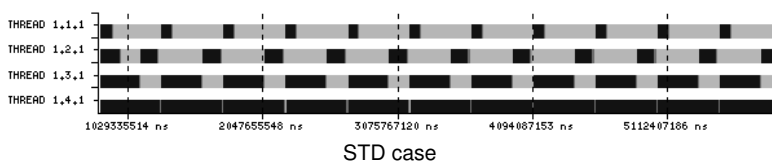
```

mpiexec -n 8 ep.A.8 Normal
chrt -c -p 0 mpiexec -n 8 ep.A.8 SCHED_HPC

```



Smart load balance



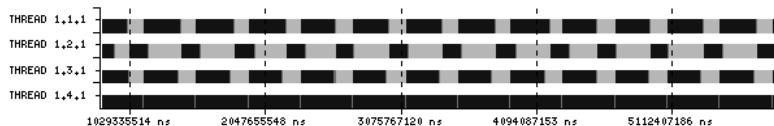
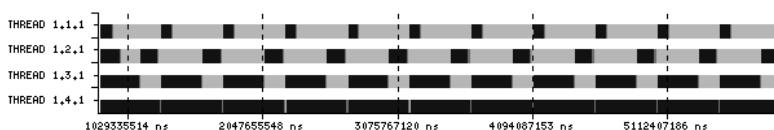
BT-MZ running on a IBM POWER5 (2 cores/2 hw thread)

- use hw thread prioritization
- completely automatic
- 18% performance improvement

[C. Boneti, R. Gioiosa, F.J. Cazorla, M. Valero, "A Danamic Scheduler for Balancing HPC Applications" in SC08, November 2008].



Smart load balance



BT-MZ running on a IBM POWER5 (2 cores/2 hw thread)

- use hw thread prioritization
- completely automatic
- 18% performance improvement

[C. Boneti, R. Gioiosa, F.J. Cazorla, M. Valero, "A Danamic Scheduler for Balancing HPC Applications" in SC08, November 2008].

Outline

- 1 Introduction
- 2 Execution time variability
- 3 High Performance Linux
- 4 Experimental results**
- 5 Summary



CPU migrations

Bench	STD case			HPL case		
	Min.	Avg.	Max.	Min.	Avg.	Max.
<i>cg.A.8</i>	30	63.61	2078	10	11.52	14
<i>cg.B.8</i>	28	90.62	3499	10	12.31	21
<i>ep.A.8</i>	29	52.41	615	10	12.02	18
<i>ep.B.8</i>	28	69.02	2536	10	12.04	19
<i>ft.A.8</i>	20	53.02	565	10	11.43	17
<i>ft.B.8</i>	28	51.23	1163	10	12.11	19
<i>is.A.8</i>	29	52.18	160	10	11.39	14
<i>is.B.8</i>	28	52.88	370	10	12.07	23
<i>lu.A.8</i>	18	70.79	1368	10	12.84	21
<i>lu.b.8</i>	29	69.04	3657	10	12.97	22
<i>mg.A.8</i>	29	54.73	590	10	11.94	22
<i>mg.B.8</i>	29	54.68	853	10	12.55	17

- CPU migrations drastically reduce
- 8 MPI process + `chrt` + `mpiexec` \geq 10 CPU migrations
- Small difference cause by `chrt` and `perf`



Process preemption

Bench	STD case			HPL case		
	Min.	Avg.	Max.	Min.	Avg.	Max.
<i>cg.A.8</i>	460	602.57	5755	333	356.32	391
<i>cg.B.8</i>	1726	2011.80	8243	343	374.72	484
<i>ep.A.8</i>	550	652.62	1886	315	344.77	436
<i>ep.B.8</i>	1198	1333.70	5239	329	365.39	472
<i>ft.A.8</i>	318	575.10	1609	331	361.32	413
<i>ft.B.8</i>	1111	1222.50	3258	337	365.29	414
<i>is.A.8</i>	396	537.35	956	326	347.37	382
<i>is.B.8</i>	519	610.93	1213	340	354.97	374
<i>lu.A.8</i>	219	1030.40	3870	325	361.81	604
<i>lu.b.8</i>	2518	2933.50	9131	340	381.46	455
<i>mg.A.8</i>	91	556.24	1776	357	386.60	423
<i>mg.B.8</i>	531	660.43	2396	357	386.44	422

- Process preemption drastically reduce and independent of input
- Small difference cause by `chrt` and `perf`



NAS benchmark suites

Bench	Std. Linux				HPL			
	Min.	Avg.	Max.	Var. %	Min.	Avg.	Max.	Var. %
<i>cg.A.8</i>	0.69	1.04	46.69	6608.70	0.68	0.69	0.70	2.94
<i>cg.B.8</i>	36.98	42.04	126.48	242.02	36.96	37.27	38.17	3.27
<i>ep.A.8</i>	8.54	8.87	14.59	70.84	8.54	8.55	8.57	0.35
<i>ep.B.8</i>	34.14	34.69	53.34	56.24	34.14	34.19	34.33	0.56
<i>ft.A.8</i>	2.27	2.50	9.07	327.31	2.05	2.06	2.08	1.46
<i>ft.B.8</i>	22.56	22.91	41.78	85.20	22.58	22.66	22.71	0.58
<i>is.A.8</i>	0.35	0.57	3.27	832.29	0.35	0.36	0.36	2.86
<i>is.B.8</i>	1.82	1.88	4.82	164.84	1.82	1.83	1.84	1.10
<i>lu.A.8</i>	17.56	19.34	50.85	189.58	17.71	17.79	18.00	1.64
<i>lu.B.8</i>	71.93	79.37	140.03	94.68	71.81	73.51	77.64	8.12
<i>mg.A.8</i>	1.40	1.60	7.80	457.14	0.96	0.97	0.97	1.04
<i>mg.B.8</i>	4.48	4.96	28.35	532.81	4.48	4.93	4.54	1.34

HPL considerably reduces ET variation for NAS benchmarks:

- ET variation < 3%
- *ep* ET variation < 1%
- *lu* still shows considerable ET variation



Outline

- 1 Introduction
- 2 Execution time variability
- 3 High Performance Linux
- 4 Experimental results
- 5 Summary**



Conclusions

Parallel applications' execution time variability is index of OS noise:

- Even simple applications are affected
- Variability can up to 66x

Several OS components may introduce OS noise (scheduler, timer interrupts, TLB misses, memory management, etc.)

In this work we focused on the OS scheduler and discovered an empirical relationship among CPU migrations, process preemption and execution time.

Execution time variation $< 3\%$ in general by reducing process preemption and CPU migrations.

Beyond performance: other metrics (power, temperature).... stay tuned!



Thank you!

Questions?

roberto.gioiosa@bsc.es, gioiosa@sprg.uniroma2.it
<http://www.sprg.uniroma2.it/home/gioiosa/>

CAOS'11

If you are interested to hardware/software co-design consider to submit your work to the 2nd edition of the **Workshop on Computer Architecture and Operating System co-design**:

<http://caos.csail.mit.edu/>



Thank you!

Questions?

roberto.gioiosa@bsc.es, gioiosa@sprg.uniroma2.it
<http://www.sprg.uniroma2.it/home/gioiosa/>

CAOS'11

If you are interested to hardware/software co-design consider to submit your work to the 2nd edition of the **Workshop on Computer Architecture and Operating System co-design**:

<http://caos.csail.mit.edu/>



Acknowledgements



IEEE Cluster 2010

IEEE International Conference on Cluster Computing 2010

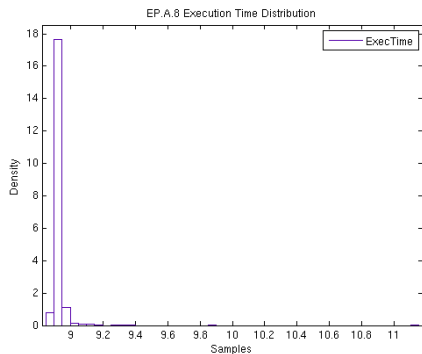
20 - 24 September 2010 - Heraklion, Crete, Greece



Backup slides



On the RT scheduler



Linux RT scheduler

- NAS EP, class A, 8 MPI process
- 1000 repetitions

- RT scheduler may help but doesn't solve the problem
- driver's components may not work properly
- critical OS activities may stop working

