Scheduling on current multi-core clusters

Emílio Francesquini

emilio@ime.usp.br

Alfredo Goldman

gold@ime.usp.br

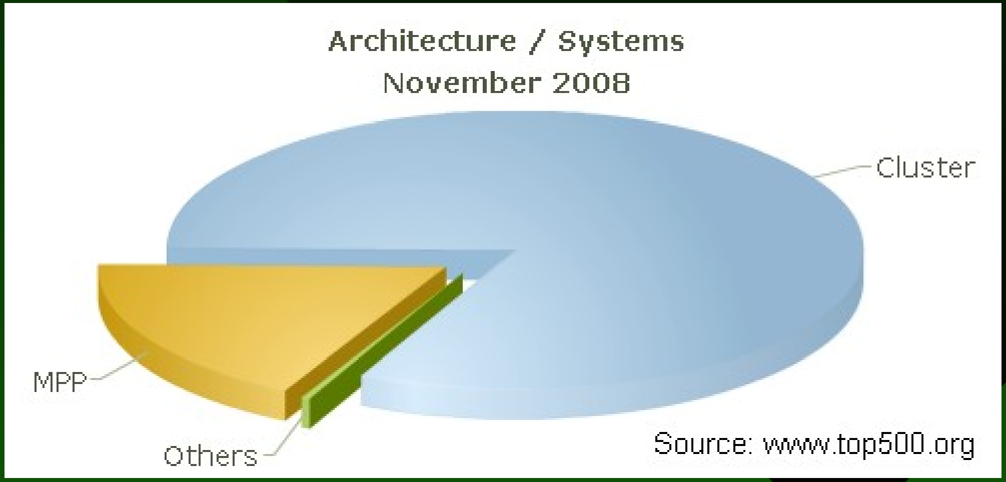
University of São Paulo, Brazil

Agenda

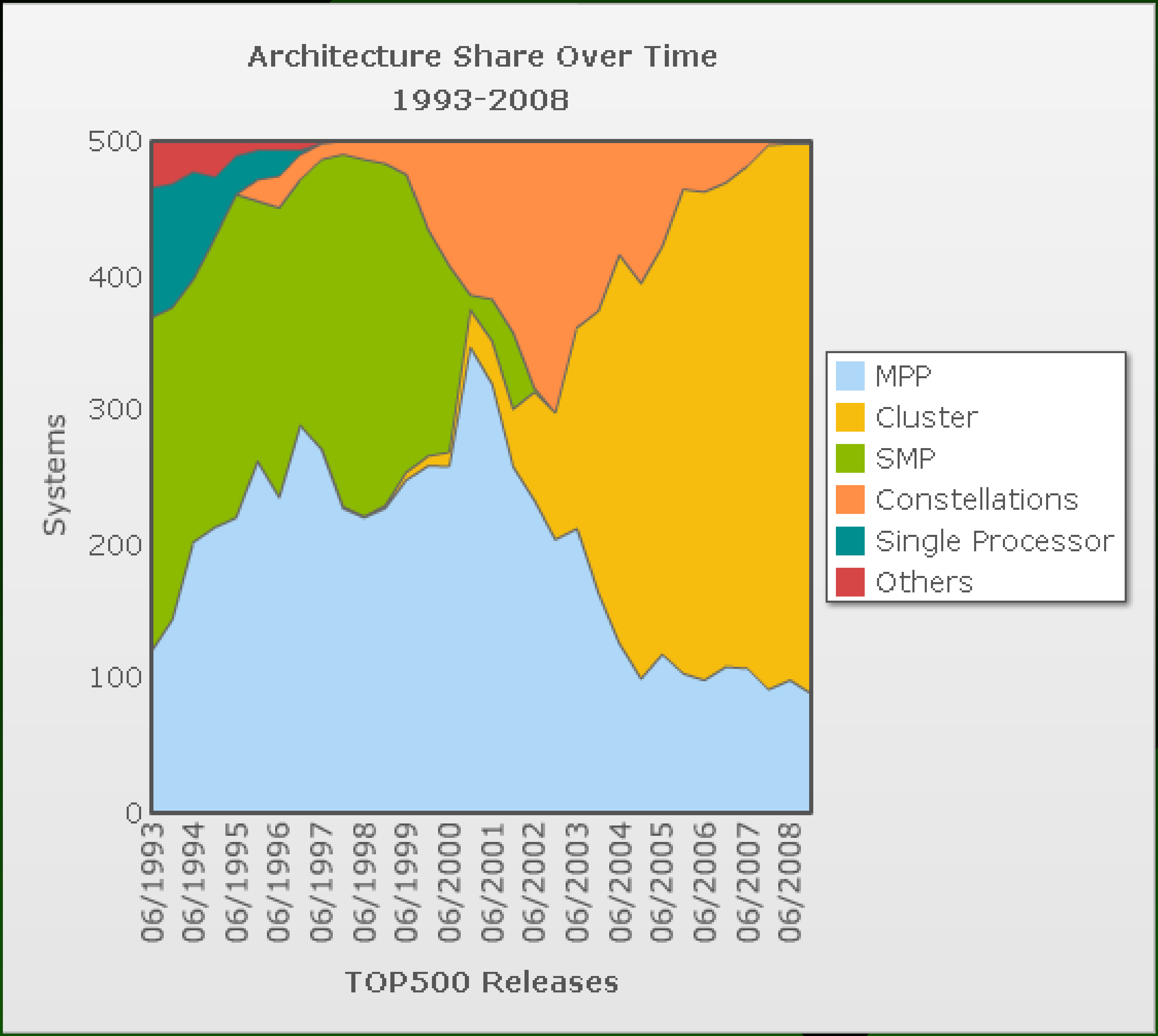
- Current scenario overview
- Some existing approaches
- Weaknesses of the current approaches
- Some of our ideas to improve the current solutions

Current scenario overview

- Multi-core processors are becoming cheaper and more common every day
- 9 out of the Top 10 (www.top500.org) computers use multi-core processors*
  - 8 of them have more than 2 cores
- Most of the Top 500 (410/500) computers are already clusters

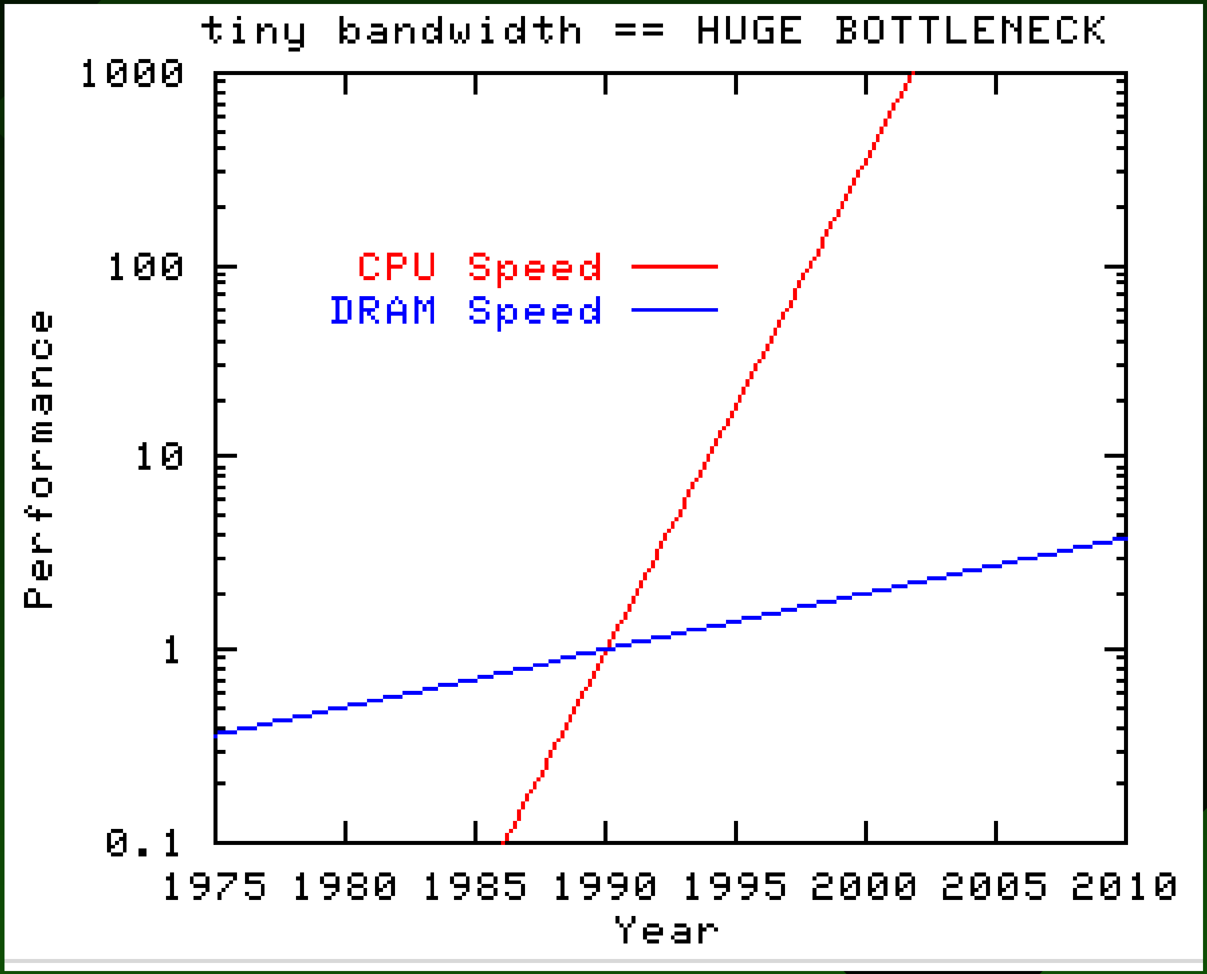


Current scenario (cont.)



Current scenario (cont.)

- Memory Bottleneck



Source: www.cs.virginia.edu/stream/

Some existing approaches

How to efficiently use all that computational power?

- Message passing
  - SLURM
    - Heavy use of process pinning
- NUMA/DSM
  - Memory pinning
  - Process pinning

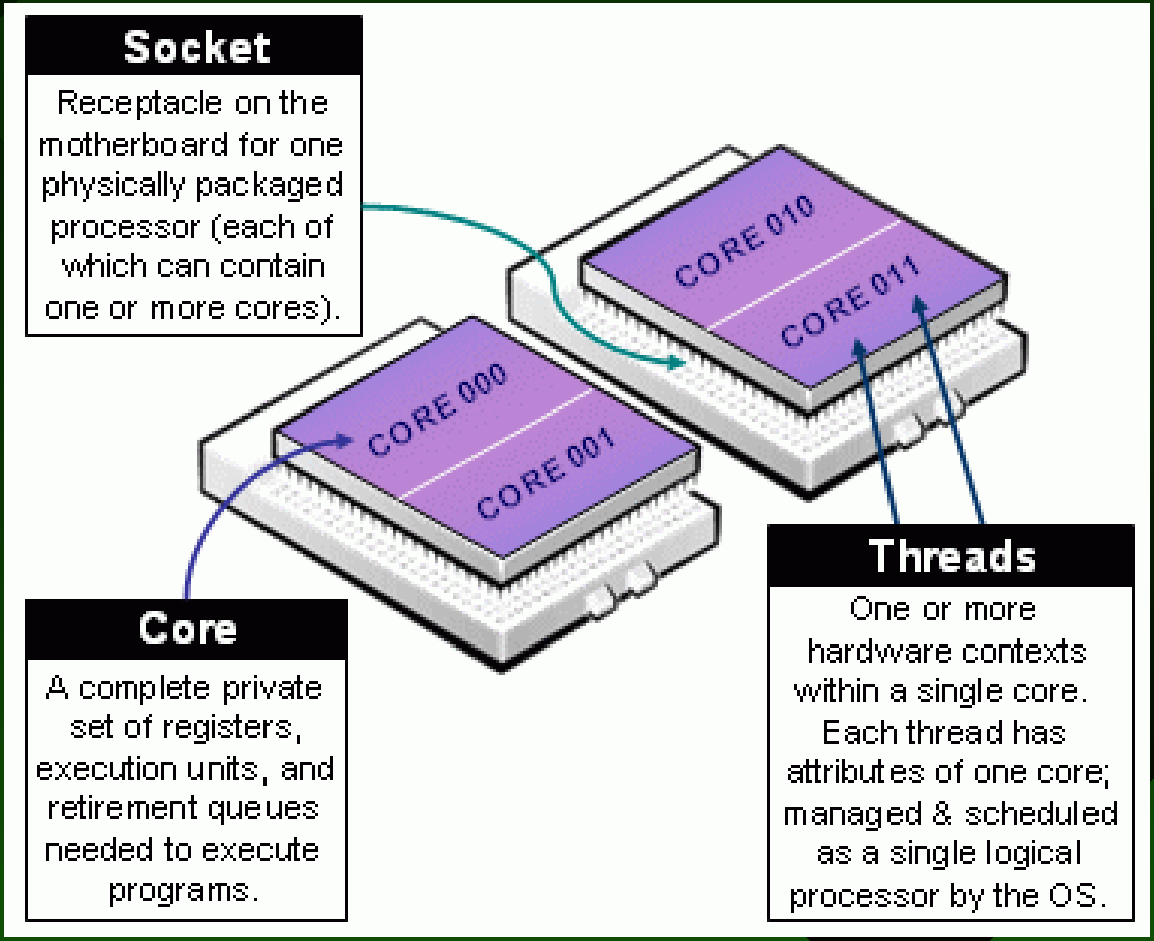
SLURM

Simple Linux Utility for Resource Management

- Open source
- Manages resources and controls queues for exclusively reserved resources
- Allows users to dispatch, to execute and to monitor jobs
- BlueGene/L at LLNL with 106,496 dual-core processors

SLURM (Cont.)

- Has three levels of hierarchy for each processing unit in the system



https://computing.llnl.gov/linux/slurm/

SLURM

- Works with the concept of process pinning
- Has low and high level flags to assert process scheduling to specific cores
- Low-level flags gives more control whereas high-level flags are much more user-friendly

SLURM
Low-level flags

- Allows process pinning to the cores
- User must be aware of the numeration scheme for their system

	c0	c1
p0	0	1
p1	2	3
p2	4	5
p3	6	7

Block numeration

	c0	c1
p0	0	4
p1	1	5
p2	2	6
p3	3	7

Cyclic numeration

SLURM
Low-level flags

- --cpu-bind=
  - mask_cpu
  - map_cpu
- Examples
  - Block numbering
    - srun -n 8 -N 4 -cpu_bind=mask_cpu:0x1,0x4 prog
    - srun -n 8 -N 4 -cpu_bind=map_cpu:0,2 prog

SLURM
High-level flags

- Created to simplify the usage
- Automatically generates the task masks
  - --sockets-per-node=S
  - --cores-per-socket=C
  - --threads-per-socket=T
  - Shortcut: -B S[:C[:T]]
- Example:
  - srun -n 8 -N 4 -B 2:1:1 prog
  - srun -n 8 -N 4 -B 2-2:1-1:1-1 prog

SLURM
Multi-core performance results
[Balle and Palermo, JSSP'07]

- Linpack on 16 cores
  - 4 nodes X 2 sockets X 2 cores

Configuration	CPUs	Time (sec)	% Speedup
No affinity control used	16	467.16	
taskset 0xf	16	481.83	-3.04%
taskset 0x1; 0x2; 0x4; 0x8	16	430.44	8.53%
–cpu_bind=map_cpu:0,1,2,3 -B 1:1	16	430.36	8.55%

SLURM Multi-core performance results

[Balle and Palermo, JSSP'07]

- LSDyna – Simulates the nonlinear dynamic response of three-dimensional inelastic structures
- Simulation of three cars collision
- Executed on a 16 core machine
  - 4 nodes X 2 sockets X 2 cores

SLURM – LSDyna performance results

Cores	Nodes	Binding	CPU binding option low-level flag high-level flag	Time (sec)	Time (D:H:M:S)	Speedup	% Speedup vs. no binding
1	1	No		194,809	2:06:06:49	1.00	
1	1	Yes	–cpu_bind=map_cpu:0 -B 1:1	194,857	2:06:07:37	1.00	-0.02%
2	1	No		104,994	1:05:09:54	1.86	
2	1	Yes	–cpu_bind=map_cpu:0,1 -B 1:1	110,702	1:06:45:02	1.76	-5.16%
2	1	Sockets	–cpu_bind=map_cpu:0,2 -B 1:1-1	104,620	1:05:03:40	1.86	0.36%
2	2	No		102,336	1:04:25:36	1.90	
2	2	Yes	–cpu_bind=map_cpu:0 -B 1:1	100,266	1:03:51:06	1.94	4.72%
8	2	No		33,616	0:09:20:16	5.80	
8	2	Yes	–cpu_bind=map_cpu:0,1,2,3 -B 1:1	31,996	0:08:53:16	6.09	5.06%
8	4	No		28,815	0:08:00:15	6.76	
8	4	Yes	–cpu_bind=map_cpu:0,1 -B 1:1	28,532	0:07:55:32	6.83	0.99%
8	4	Sockets	–cpu_bind=map_cpu:0,2 -B 1:1-1	26,081	0:07:14:41	7.47	10.48%

NUMA
Non-Uniform Memory Access

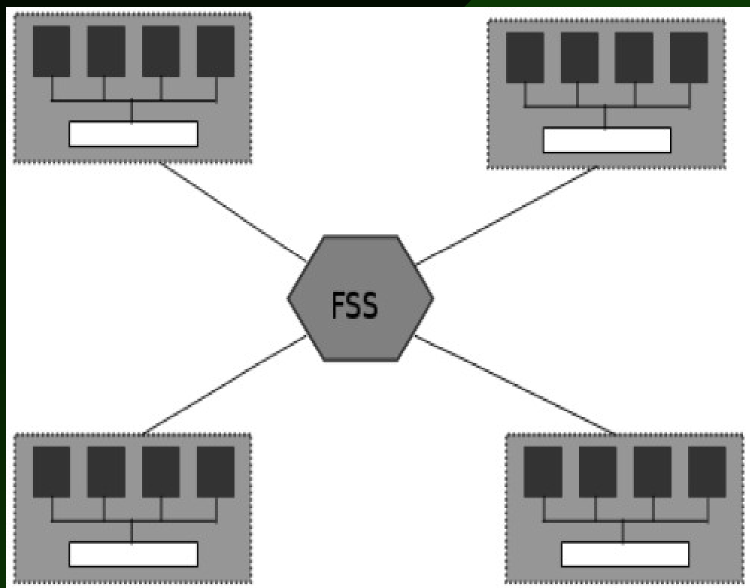
- OpenMP and pthreads
- NUMA support (Linux kernel >= 2.6)
  - Memory pinning
  - Process pinning
- Manual control over memory and process pinning

NUMA

- Techniques
  - First touch initialization/Parallel initialization → no guarantees
  - Memory/Process pinning
    - `sched_setaffinity`
    - `Mbind`
      - `bind/interleave/preferred`

NUMA performance test architecture

[Pousa, Méhaut et al. , WSCAD'08]



- NUMA factor: 2 → 2.5
- 16 Itanium2 at 1.6 GHz
- 64 GBytes of RAM

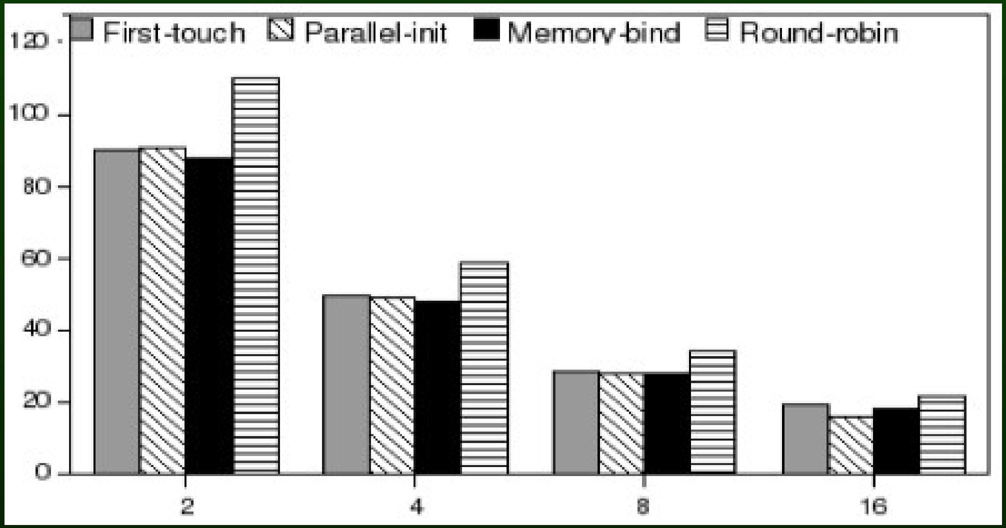
- NUMA factor: 1.2 → 1.5
- 8 dual-core Opteron at 2.2 GHz
- 32 Gbytes of RAM

NUMA Performance comparison

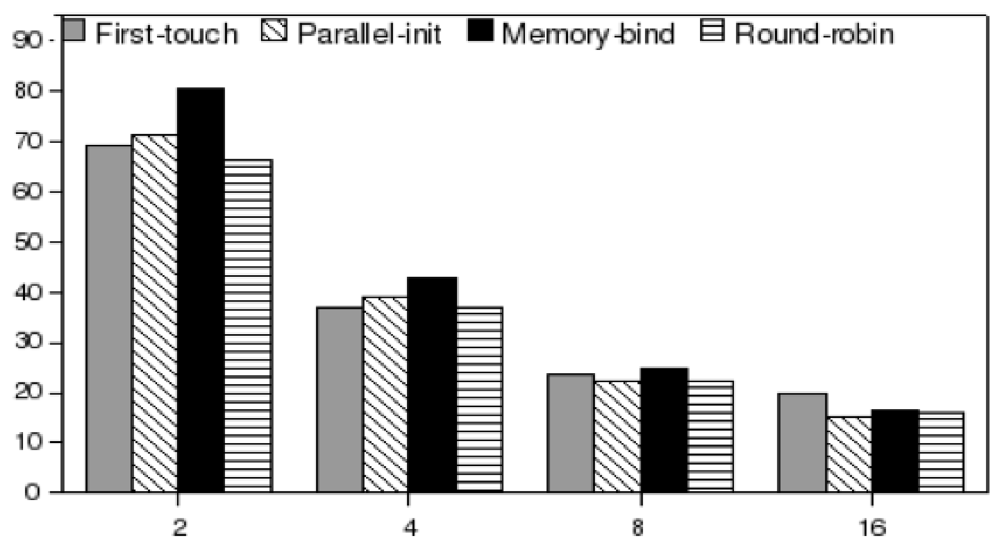
Ondes 3D

- Application for seismic wave propagation simulation

- Regular data access pattern



Itanium Cluster



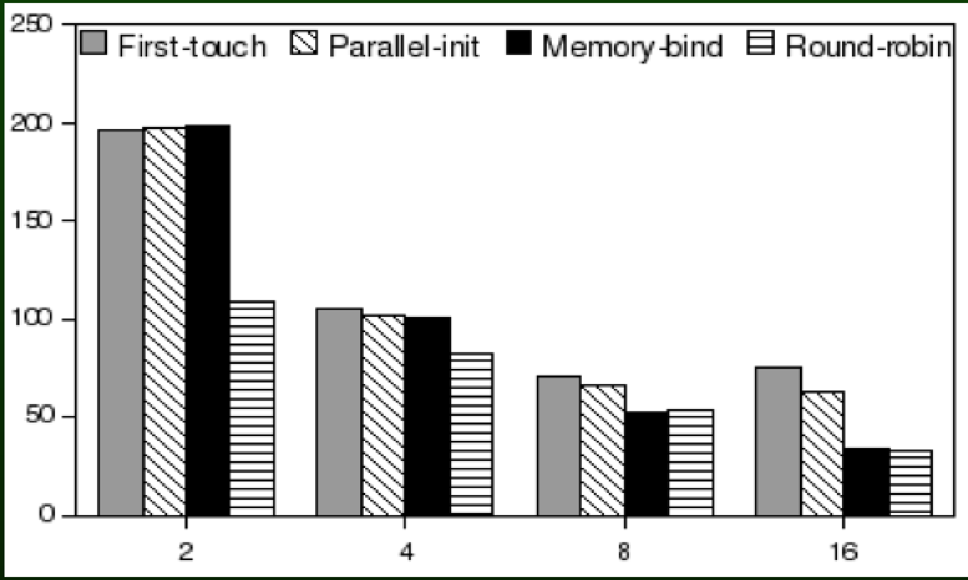
Opteron cluster

NUMA Performance comparison

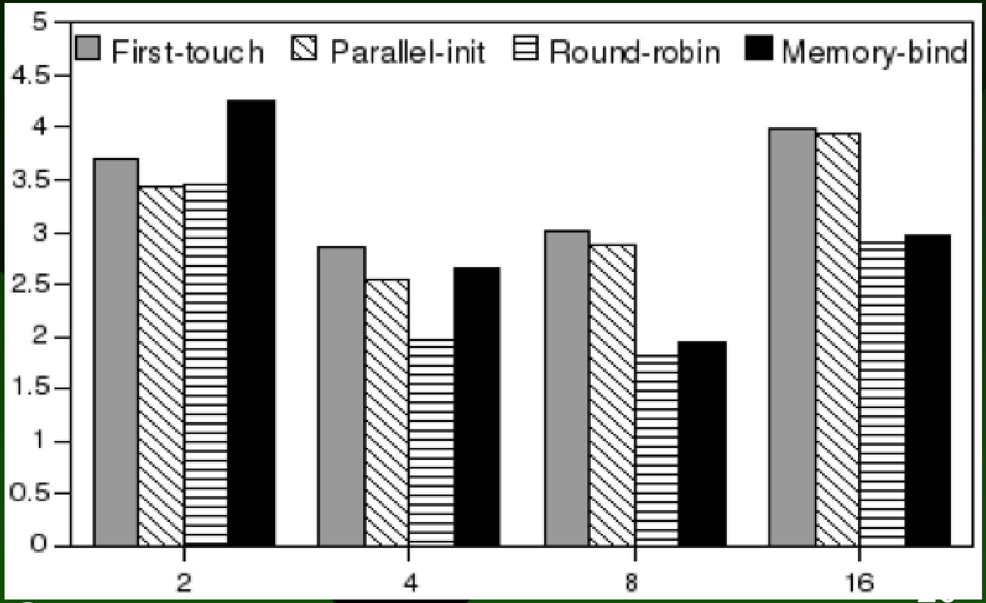
Benchmark NAS

- Simulation of fluid dynamics

- CG Kernel
    - Large memory footprint
    - Irregular data access pattern



Itanium



Opteron

Weaknesses of the current approaches

- Lack of portability
- Not suitable (or even usable) for heterogeneous clusters
- Demands expertise from the developer and the executor (not always the same person)
- Scheduling relies too much on the users

Some of our ideas...

- Profiling
- Dynamic Scheduling using online profiling and profiles obtained from previous runs
- Let the user specify the architecture/topology of his network. But also try to discover what is possible without user intervention

Why ?

- To allow the developer to focus on the problem, and not on architectural details
- Portability
- Deal with node idiosyncrasies seamlessly
- We believe the simplicity pays off the eventual losses in performance in most cases

Why? (cont.)

- Application behavioral patterns may change
  - During execution
  - From inputs
  - During its lifecycle

Conclusion

- We've presented a current problem
- Future steps
  - To propose a theoretical model
    - Cache proximity
  - To evaluate it

Thank you!

A small quotation...

So why should I be so happy about the future that hardware vendors promise? They think a magic bullet will come along to make multicores speed up my kind of work; I think it's a pipe dream. No!—that's the wrong metaphor! "Pipelines" actually work for me, but threads don't. Maybe the word I want is "bubble."

Donald Knuth

www.informIT.com