Scheduling: today and tomorrow

Larry Carter

Apologies for

- Misrepresenting your work
- Not knowing very much
- Taking extreme positions
- ...

Market-based systems are inevitable

A "convergence of technologies" is needed:

– Electronic money
– Allocatable resources
– Trust
– ...

from 2005 workshop

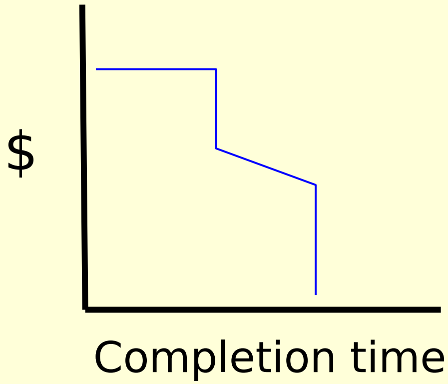
Once this happens, compute power will either be:

- Abundant
- Scarce

Moore's law will eventually fail; people's imagination won't

.

Issues for market-based scheduling



User's quality measure:

- ~~Makespan, stretch, or steady-state throughput~~
- User-specific "utility"

Server's goal:

- ~~Maximize throughput~~
- Maximize profit ("structural unemployment" may be beneficial)

System wide goal:

- ~~Fairness~~
- Equal access to market, but "money talks"

User's motivation:

- ~~Altruism~~ Profit

New opportunities:

- Brokers, publicists, insurers, …

from 2005 workshop

Towards more generality

Computational platform

- Homogeneous → heterogeneous platform
- Processor time → communication, memory, … times
- Centralized → distributed decision making
- Reliable → unreliable or collusive processors
- One → multiple administrative domains

red = new since Aussois 2004

Applicaton model

- Independent tasks → Job = DAG of tasks
- Constant → changing resource needs
- Uniform tasks → multiple bags of tasks

Objective of optimization

- single goal → individual utility functions

Combating NP-completeness

NP complete result → adaptive approximation algorithms → (simgrid) simulations

Makespan → Steady state → Trim analysis

Non-traditional features

	Platform model	Application model	Objective function	Other features
Rosenberg		DAG	maximize ||-ism	
Weinberg	multi-level memory			symbiosis
Lee	supercomputer	(real)	user-specified	
Jeannot			reduce errors	colluding users
Dongarra	supercomputer	(real)		
Trystram	multiple domains		multiple	
Agrawal	varying allocation			adversarial allocator
Beaumont	heterogeneous	multiple, divisible		
Detti			reliability	crashes possible
Marchal	heterogeneous	multiple bags	stretch	on-line versus off-line
…		ET CETERA		…

What we're accomplishing

Breadth-first search of new models

- Driven by technology changes
  - Multicore, unreliable processors, …
- ~~Improving constant from 9/7 to 5/4~~

Introducing (potentially important) new paradigms, e.g.

- IC-optimality
- Symbiosis
- Collusion-resistance
- Nash equilibrium
- …

What we’re not accomplishing

New algorithms implemented in “real” system

(Perhaps if we were that successful, we wouldn’t be attending this workshop)

What should we be accomplishing ??

- Computer science is not a natural science
  - We get to invent our own models
  - Discovering properties of random models isn't nearly as interesting as discovering "nature"
- We should work towards having an influence
  - (Well, that's my opinion)

How to have an influence

"Throw great idea over the wall"

- i.e. publish paper
- If it's good enough, people will pick it up
  - example: randomized routine

But what's on the other side?

- "Not invented here"

Usually, we must do (much) more

A not-yet influential idea

Bandwidth-centric scheduling

“A parent node responding to requests from multiple children should give first priority to child with highest bandwidth.”

A not-yet influential idea

Bandwidth-centric scheduling

- "A parent node responding to requests from multiple children should give first priority to child with highest bandwidth."

Why hasn't this been adapted by BOINC ??

- Bandwidths don't follow one-port model
- BOINC doesn't even know the bandwidths

The other side of the wall

“Not invented here”

- Learning our language and sifting many papers is very difficult
- People have their own ideas they think are good

To overcome these barriers, we need to

- Learn about their world
- Demonstrate effectiveness on their data
- My experience: this effort benefits <u>me</u>
  - new problems
  - new ideas

Other ways to have influence

Often, our ideas are discovered independently by others

- At best, our theory can help assure others that the ideas are valid (a constructive interaction)
- At worst, we can get into big fights over who deserves the credit or patents

Our work can suggest what general directions are more or less promising (if we can get ourselves in an advisory position).

Perhaps we can demonstrate value of:

- Collecting extra information
- Providing new capability

Influence-aware research

Suppose we want to do research on desktop grids for DAG applications

- What is a potential application?
- What information would be readily available to scheduler in such an application.
- Can we argue that our technique is so good it's worth the effort to collect needed parameters?
- Can we envision a path towards implementation

Possible target: Chess or Go on BOINC

Multi-core: a new opportunity

Jack Dongarra’s problem (LAPACK)

- Even easily parallelized applications will need to tolerate variable execution times and failures

Scheduling for more general progams (e.g. threaded programs) will be needed

- Adaptive, self-scheduling techniques are easiest to get adapted

Locality will be very important in future

- The cost of moving data is MUCH more than the cost of computation

We must learn to live with unreliable cores

Conclusion

We’re doing excellent work

- I’m not suggesting you totally change your research

Perhaps we could do more to get work used

Discussion

Backup slides

Symbiotic Scheduling

Symbiosis: Two applications run concurrently take less time than running one then the other.

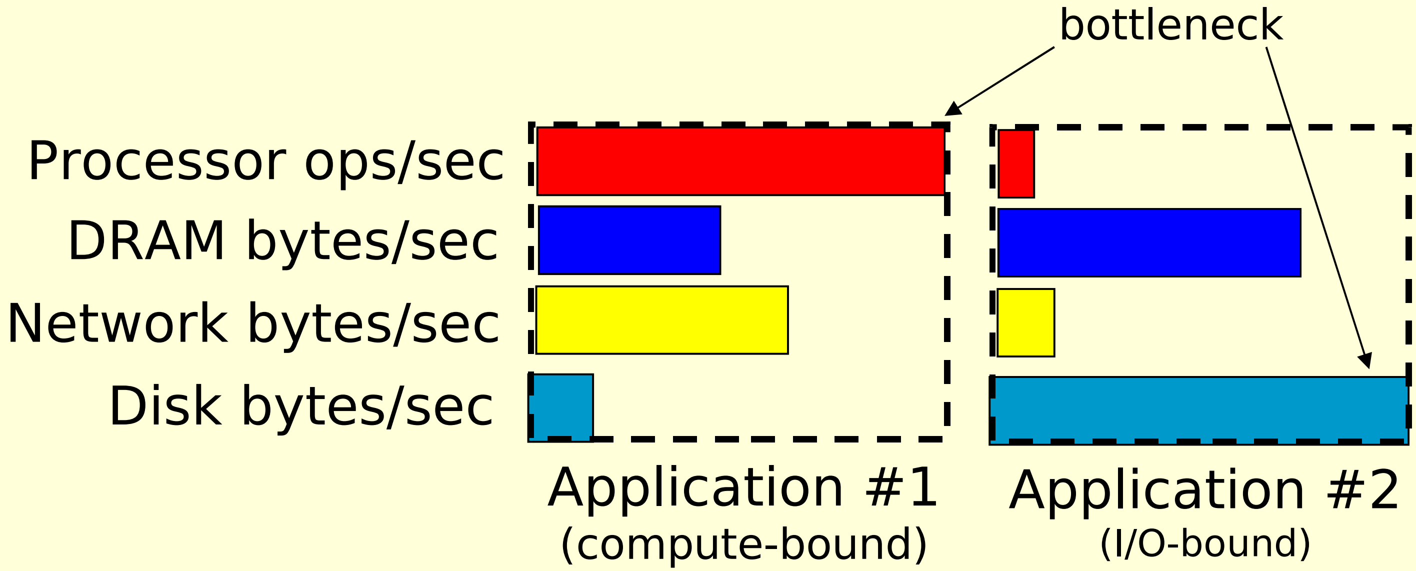
- "Timesharing" on uniprocessors usually isn't symbiotic.
- Symbiosis has been demonstrated for multithreaded processors (Snavely)

Typical node:

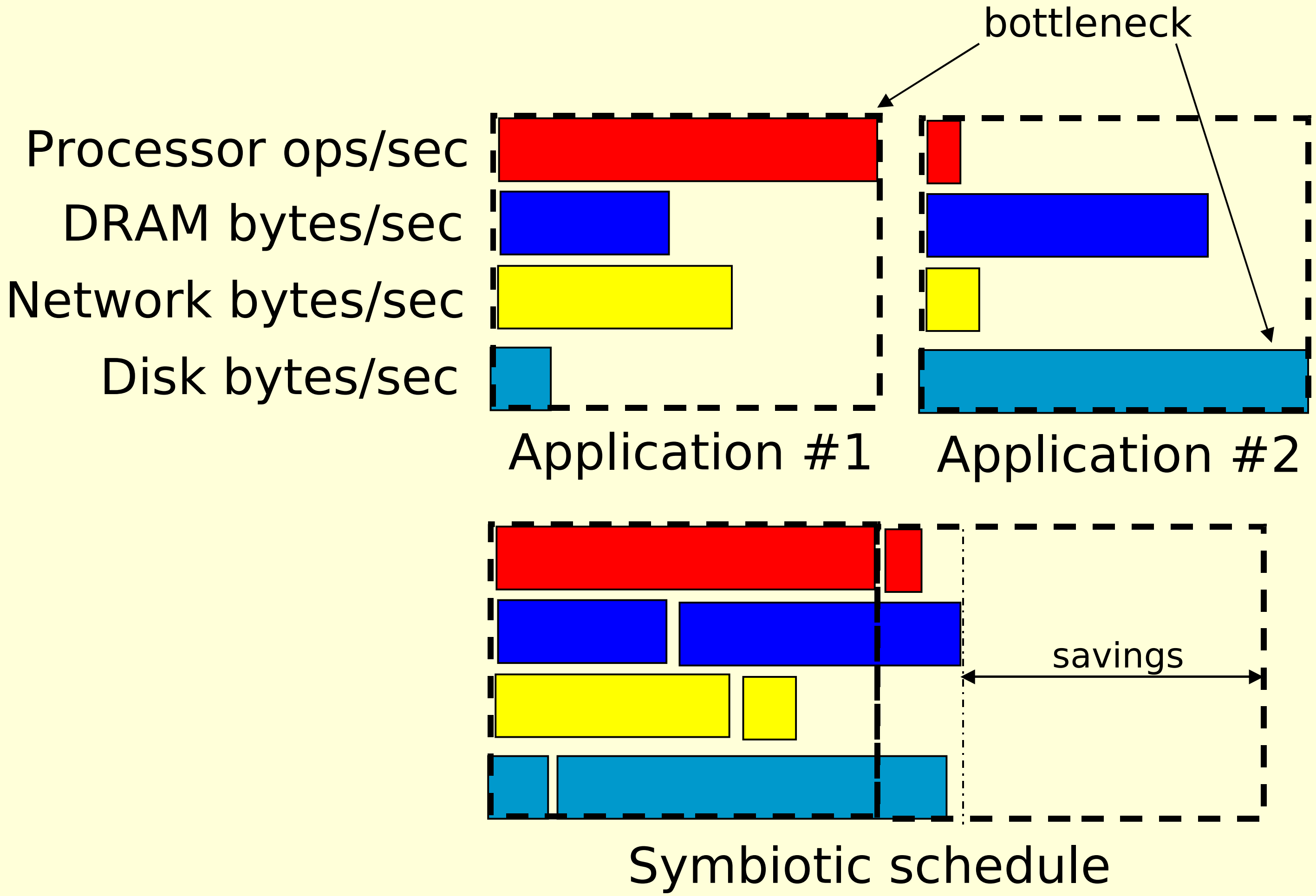
- Multiple processors
- Shared memory (but separate caches)
- Communication network to other nodes
- I/O channel to disks

Opportunity for symbiosis when different applications have different bottlenecks.

Symbiosis


bottleneck
Processor ops/sec
DRAM bytes/sec
Network bytes/sec
Disk bytes/sec
Application #1
(compute-bound)
Application #2
(I/O-bound)

Symbiosis



bottleneck
Processor ops/sec
DRAM bytes/sec
Network bytes/sec
Disk bytes/sec
Application #1
Application #2
savings
Symbiotic schedule