

[after taking the five-minute break]

### Administrivia

- Why have a five-minute break?
  - break up the monotony of lecture
  - give us a chance to stretch, use the bathroom
  - come back refreshed
- Do I really want to give up five minutes of lecture?
  - I'll try to end on time.
  - In return, please don't rustle backpacks, etc., until I'm done.
  
- office hours
  - Tu 1-3 upstairs at Za's for now
  - may push back and move into lab at some point

- introductory handout
  - introduction [pass around handout]
  - ask about balance of grads/undergrads
  - course objectives and rationale
    - understand how and when to use modern language abstractions
    - understand the engineering design process behind language evolution
    - be able to enumerate and explain the challenges for parallel programming
    - be aware of the wide range of efforts to meet these challenges and why today we have no polished answers
  - work for the class
    - four homework + lab problems
    - two exams
  - sharing answers
    - ok on “homework”
    - ok within team on lab problems
  - intent: not 391++, but I’ll try to leave projects open-ended enough for ambitious students
  - grading & extra hour option (project, presentation, report)
  - web board + page
  - tentative syllabus
    - ~1/3 language abstractions from an engineer’s perspective
    - ~1/6 challenges/difficulties in parallel programming
    - ~1/2 overview and evaluation of approaches
  - [ask for comments/other objectives of student interest]

**Motivation: Language-Architecture Interaction**

- Who cares about how programming languages and architecture interact?
- data courtesy of Saman Amarasinghe@MIT (+ Martin Rinard + Charles Leiserson)
- application
  - dense matrix multiply (  $C_{ij} = \sum_k A_{ik} B_{kj}$  )
  - on dual quad-core Intel machines
  - 1024x1024 matrices → 2^30 multiple-adds → a few seconds?

<u>if you...</u>	<u>...you lose</u>	
ignore processor parallelism	3.5x	
don't use Intel hand-optimized assembly library	2.7x	
don't bother to vectorize (MMX/SSE)	2.8x	
ignore cache size	1.7x	
ignore data org. in memory (don't transpose matrix)	3.4x	
use Java!	2.1x	
use objects	2.2x	
allow double & integer matrices	2.4x	
<u>use immutable objects!</u>	<u>220x</u>	
	<u>~300000x!</u>	
		<hr/> "real" parallelism μarch!  arch  μarch! μarch! <hr/> language... sort of (extra insts?) <hr/> branches (μarch!) <hr/> language

- note that 2.1x is the kind of number that managed language proponents claim as the cost of using managed code
- the real cost is having no way to get at the remaining 100x, even if you're willing to do the work
- [MMX = multimedia extensions, SSE = streaming SIMD extensions]

## Motivation: Parallel Programming

- “Parallelism is not cheaper or easier, it’s *faster*.” –Jim Gray, 1995  
[quoted from Soumen Chakrabarti’s thesis; CHECK ORIGINAL SOURCE!  
VLDB Tutorial on “A Survey of Parallel Database Techniques and Systems”]
- may no longer be true!
  - Moore’s law continues:  $2\times$  transistors every 18 months. [fixed area]
  - Performance growth of single core flatlined ~3-4 years ago (not entirely true, but sharply reduced).
  - What will we do with the transistors?
- easy (relatively) hardware answer
  - build copies
  - let the software worry about it
- Jim Gray ca. 1996, after moving to Microsoft: Performance is not relevant to Microsoft; functionality is our primary concern.
- Will the software industry figure out how to overcome the additional problems inherent to parallel code?
- Maybe...
- It’s an exciting time to be a young engineer!

## **Thoughts on Potential Labs**

- memory debug implementation
- reference-counting garbage collection implementation
- profiling and optimization exploration/experimentation
- exploration of algorithm-architecture interaction in parallel kernels
- implementation of compiler/user taks library
  
- parallelizing applications?
  - probably not as a main class project
  - but could be a good extra hour for grad students
  
- nothing ruled out yet; can be flexible to accommodate student desires for class

## **Why C++?**

- illustrates most/all modern language abstractions
- reasonably transparent (although more complex than C)
- flexible enough to be used in systems programming
- [because] no requirement to use opaque abstractions
- easily linked with other code (e.g, hand-optimized assembly)