

# CS 610: Compiler Challenges for Parallel Architectures

**Swarnendu Biswas**

Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur

Sem 2024-25-I

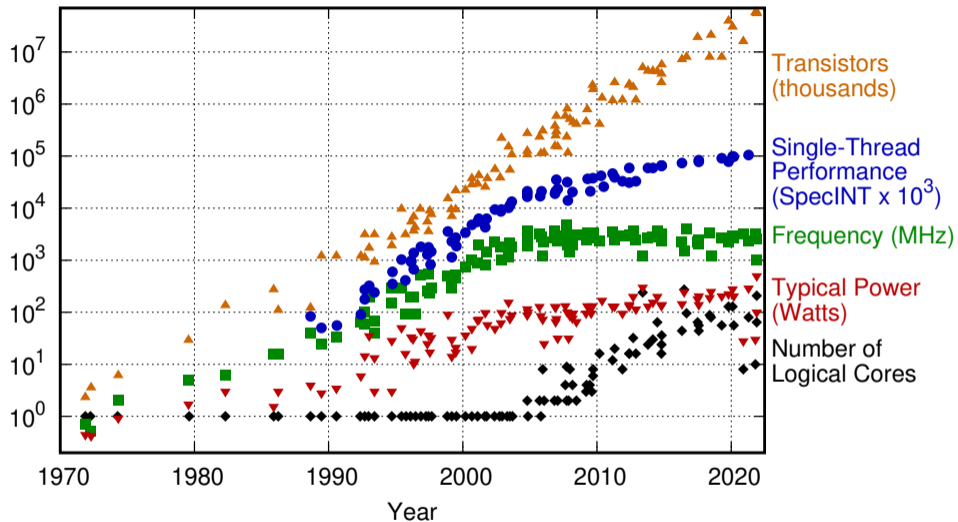


# Improvements in Computing Capabilities

- Last few decades have been exciting for the parallel computing community
- Sources of improvements in computing capabilities
  - (i) Improvement in underlying technology (leads to Moore's law)
  - (ii) Advances in computer architecture

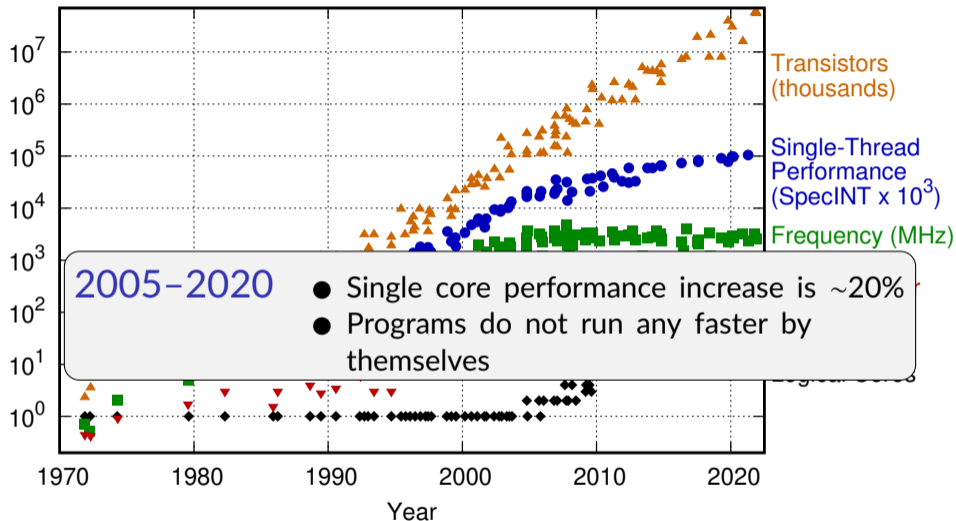
'50-65	Instruction-level parallelism (pipelining)	Vector operations	'75-90
'60-70	Multiple functional/execution units	Deeper and sophisticated memory hierarchies	'85-00
'65-75	Superscalar instruction issue and VLIW	Core-level parallelism	'90-...

# 50 Years of Microprocessor Trend Data



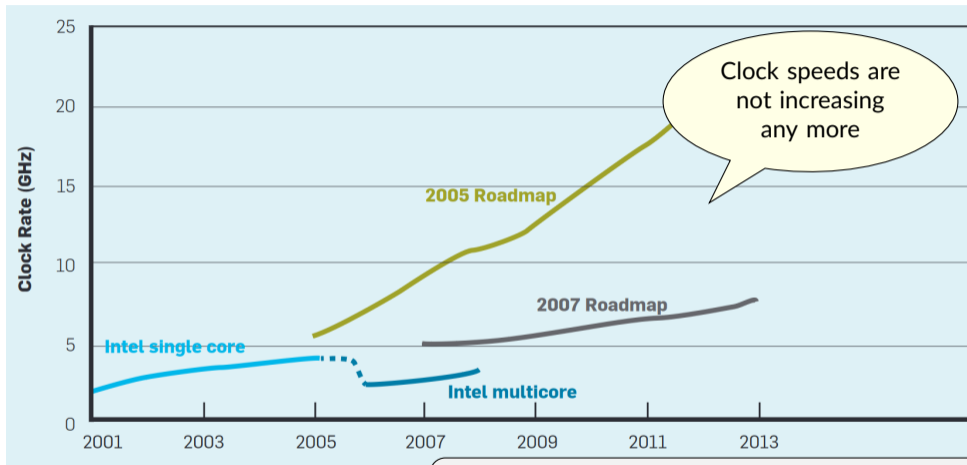
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2021 by K. Rupp

## 50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2021 by K. Rupp

# Challenges to Growth in Performance



Power, and not manufacturing, limits microarchitectural improvements - F. Pollack

# Programs Do Not Run Any Faster by Themselves!

## Microarchitectural techniques

Multiple functional units, superscalar architecture, VLIW, sophisticated cache structures, deeper pipelines

## Law of diminishing returns!

There is little or no more hidden parallelism (ILP) to be found

# Programs Do Not Run Any Faster by Themselves!

## Microarchitectural techniques

Multiple functional units, superscalar architecture, VLIW, sophisticated cache structures, deeper pipelines

## Complex systems are more difficult to program efficiently

Systems programmers now need to be aware of memory hierarchies and other architectural features to fully exploit the potential of the hardware

## Have you heard of ninja programmers?

Popular libraries like Intel oneDNN and NVIDIA cuDNN are hand-optimized for performance

What is the software side of the story?



# Develop Parallel Programs

From my perspective, parallelism is the biggest challenge since high-level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful.

...

Industry is building parallel hardware, assuming people can use it. And I think there's a chance they'll fail since the software is not necessarily in place. So this is a gigantic challenge facing the computer science community.

– David Patterson, ACM Queue, 2006.

## Develop Parallel Programs

To save the IT industry, researchers must demonstrate greater end-user value from an increasing number of cores.

– A View of Parallel Computing Landscape, CACM 2009.

# New Challenges in Software Development

- Adapt to the changing hardware landscape
- Many applications are single-threaded

How can we develop software that makes effective use of the extra hardware?

# Compilers to the Rescue!

A compiler is a **system software** that translates a program into a source language to an equivalent program in a target language.



## Role of a compiler

- Generate correct code
- Must improve the code according to some metric
- Provide feedback to the user

# Relevance of Compiler Technologies

- Compiler technology has become more important as machines have become more complex
- Success of architecture innovations depends on the ability of compilers to provide efficient language implementations on that architecture
- Excellent techniques have been developed for vectorization, instruction scheduling, and management of multilevel memory hierarchies
- Automatic parallelization has been successful only for shared-memory parallel systems with a few processors



# The Golden Age of Compilers

in an era of Hardware/Software co-design

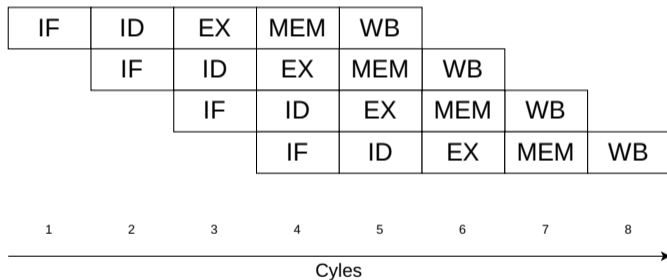
International Conference on  
Architectural Support for Programming Languages and  
Operating Systems (ASPLOS 2021)

**Chris Lattner**  
SiFive, Inc.

**April 19, 2021**

# Pipelined Execution

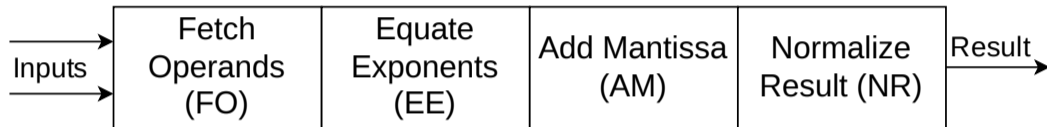
- Pipelining subdivides a complex operation into independent micro-operations
  - ▶ Assume the different micro-operations use different resources
  - ▶ Micro-operations can be overlapped by starting an operation as soon as its predecessor has completed the first micro-operation



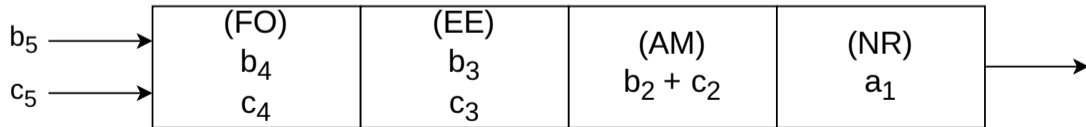
- A pipelined functional unit is effective only when the pipeline is full
  - ▶ Operands need to be available on each segment clock cycle

# Pipeline for Floating-Point Operations

## Steps in floating-point addition



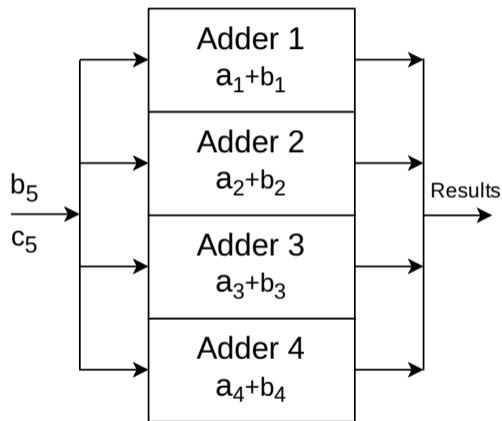
A pipelined execution unit computing  $a_i = b_i + c_i$





# Parallel Functional Units

- Multiple functional units
  - ▶ Assume  $n$  units and  $m$  cycles for an operation to complete
  - ▶ Can issue  $n/m$  operations per cycle
- Also called **fine-grained** parallelism
- More flexible in allocating operations compared to pipelining but costlier to implement

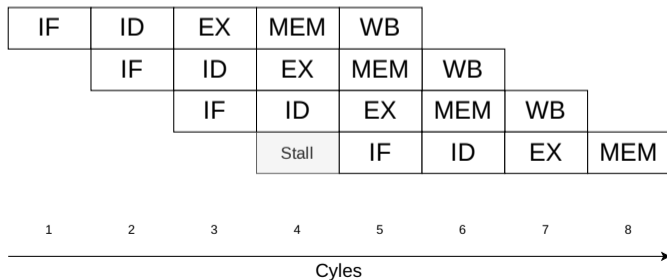


# Compiler Challenges with Pipelining

The key performance barrier is **pipeline stalls**, which occur when a new set of inputs cannot be injected into the pipeline because of a hazard

## Structural hazards

- Available machine resources do not support instruction overlap
  - ▶ For example, a machine cannot overlap instruction fetch with the fetch of data if there is only one memory port
- Such a hazard cannot be avoided through compiler strategies



# Compiler Challenges with Pipelining

The key performance barrier is **pipeline stalls**, which occur when a new set of inputs cannot be injected into the pipeline because of a hazard

**Data hazards** ● Result produced by one instruction is needed by a later one

```
ADD R1, R2, R3
SUB R4, R1, R5
```

```
LW R1, 0(R2)
ADD R3, R1, R4
```

● Compiler can schedule an instruction that does not use R1

**Control hazards** ● Occur during the processing of branch instructions

# Vector Instructions

- Apply the same operation to different positions of one or more arrays

**Goal:** Keep pipelines of execution units full

```
VLOAD V1, A
VLOAD V2, B
VADD V3, V1, V2
VSTORE V3, C
```

vectorize  
⇒

```
C(1:N) = A(1:N) + B(1:N)
```

## Challenges

- Increases processor state to support vector registers
- Increases the cost of processor context switching
- Expanded instruction set, complicates instruction decode
- Stresses memory bandwidth, can pollute the cache hierarchy

# Compiler Challenges with Vector Instructions

```
DO I = 1, 64  
  C(I) = A(I) * B(I)
```

vectorize  
⇒

```
C(1:64) = A(1:64) * B(1:64)
```

```
DO I = 1, 64  
  A(I+1) = A(I) + B(I)
```

vectorize  
⇒

```
A(2:65) = A(1:64) + B(1:64)
```

# Superscalar and VLIW Processors

Goal is to issue multiple instructions in the same cycle

**Superscalar** looks ahead in the instruction stream and issues instructions that are ready to execute

**VLIW** executes a “wide” instruction consisting of multiple regular instructions per cycle that utilize different functional units

## Challenges

- Finding enough parallel instructions
- Require more memory bandwidth for fetching instructions and data
  - Poor locality will waste memory bandwidth

# Compiling for Multiple-Issue Processors

Compiler must recognize when operations are not related by dependence

**Solution:** vectorization

Compiler must schedule instructions so that it requires as few total cycles as possible

**Solution:** instruction scheduling

# Importance of Instruction Scheduling

Assume that a memory access (i.e., LD/ST) takes 3 cycles and ADD takes 1 cycle.

## Naïve

```
LD   R1 , A
LD   R2 , B
ADD  R3 , R1 , R2
ST   X , R3
LD   R4 , C
ADD  R5 , R3 , R4
ST   Y , R5
```

How many  
cycles?

## Improved

```
LD   R1 , A
LD   R2 , B
LD   R4 , C
ADD  R3 , R1 , R2
ADD  R5 , R3 , R4
ST   X , R3
ST   Y , R5
```

How many  
cycles?



# Scheduling in VLIW

```
LD   R1, A
LD   R2, B
ADD  R3, R1, R2
ST   X, R3
LD   R4, C
LD   R5, D
ADD  R6, R4, R5
ST   Y, R6
```

Consider a VLIW system that can issue two memory accesses and two additions per cycle

## Schedule 1

LD R1, A	LD R4, C
LD R2, B	LD R5, D
delay	delay
ADD R3, R1, R2	ADD R6, R4, R5
delay	delay
ST X, R3	ST Y, R6

## Schedule 2

LD R1, A	LD R4, C
LD R2, B	LD R5, D
ADD R3, R1, R2	—
—	ADD R6, R4, R5
ST X, R3	—
—	ST Y, R6

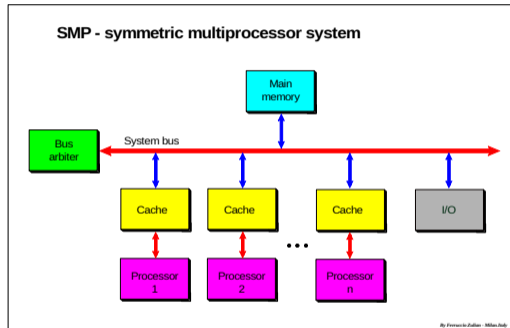
# Processor Parallelism

## Synchronous Parallelism

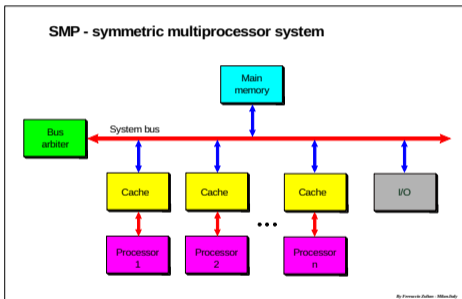
- Replicate processors, with each processor executing the same program on different data
- **Data Parallelism** – same task on different data

## Asynchronous Parallelism

- Replicate processors, but each processor can execute different programs
- Requires explicit synchronization
- **Task Parallelism** – independent tasks on the same or different data



# Compiling for Asynchronous Parallelism



```
PARALLEL DO I = 1, N  
  A(I+1) = A(I) + B(I)
```

```
PARALLEL DO I = 1, N  
  A(I-1) = A(I) + B(I)
```

```
PARALLEL DO I = 1, N  
  S = A(I) + B(I)
```

```
DO I = 1, M  
  DO J = 1, N  
    A(I, J) = 2*B(I, J) + 3*C(I, J)
```

# Granularity of Parallelism

## Vectorization

- Parallelism is **finer-grained**
- Synchronization overhead is small

```
DO I = 1, M
  DO J = 1, N
    A(I,J) = 2*B(I,J) + 3*C(I,J)
```

Compilers should **parallelize the outer loops** and **vectorize the inner ones**

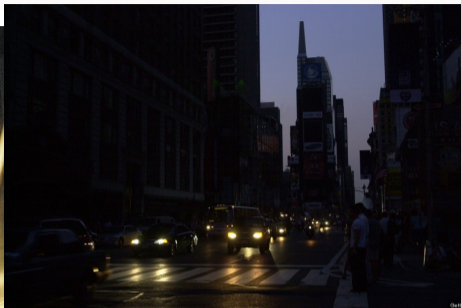
## Asynchronous Parallelism

- Parallelism is **coarser-grained**
- Larger start-up and synchronization overheads

# Challenges in Developing Parallel Programs

- Programmers tend to **think sequentially**
  - ▶ Correctness issues – concurrency bugs like data races and deadlocks
  - ▶ Performance issues – minimize communication across cores
- Overheads of parallel execution
  - ▶ Amdahl's law limits scalability
  - ▶ Other challenges like load balancing

# Writing Concurrent Programs is Hard



# Writing Concurrent Programs is Hard



**BUSINESS SOFTWARE**  
business

**BUSINESS READY**

## Nasdaq's Facebook Glitch Came From Race Conditions

Joab Jackson  
@Joab\_Jackson

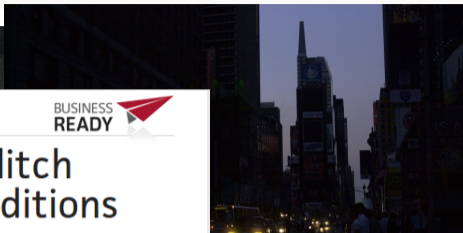
May 21, 2012

The Nasdaq computer system that delayed trade notices of the Facebook was plagued by race conditions, the stock exchange announced Monday. This technical glitch in its Nasdaq OMX system, the market expects to pay million or even more to traders.

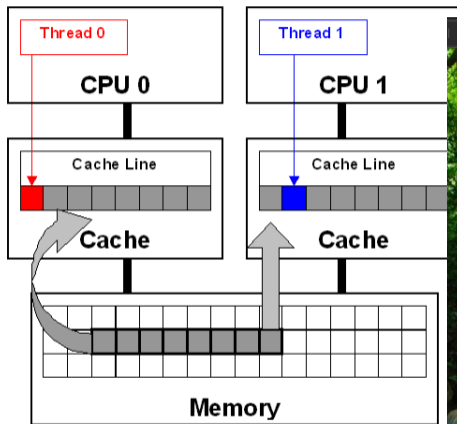
A number of trading firms **lost money** due to mismatched Facebook shares. 30 million shares' worth of trading were affected, the exchange estimated.

On Friday, Nasdaq **had delayed** Facebook's IPO by 30 minutes. For about the exchange stopped confirming trades placed by brokers, who were unable to see the results of their orders for more than two hours.

0  
Like  
1  
Tweet  
11  
g+1  
in  
in Share  
0  
Pin.it  
submit



# Performance Bugs



mikaeronstrom.blogspot.com/2012/04/

TUESDAY, APRIL 10, 2012

### MySQL team increases scalability by >50% for Sysbench OLTP RO in MySQL 5.6 labs release april 2012

A MySQL team focused on performance recently met in an internal meeting to discuss and work on MySQL scalability issues. We had gathered specialists on InnoDB and all its aspects of performance including scalability, adaptive flushing and other aspects of InnoDB, we had also participants from MySQL support to help us understand what our customers need and a number of generic specialists on computer performance and in particular performance of the MySQL software.

The fruit of this meeting can be seen in the MySQL 5.6 labs release april 2012 released today. We have a very interesting solution to the adaptive flushing problem. We also made a significant breakthrough in MySQL scalability. On one of our lab machines we were able to increase performance of the Sysbench OLTP RO test case by more than 50% by working together to find the issues and then quickly coming up with the solution to the issues. Actually in one particular test case we were able to improve MySQL performance by 6x with these scalability fixes.

In this blog I will provide some details on what we have done to improve the scalability of the MySQL Server on large servers.

MySQL have now reached a state where the solutions to the scalability is no longer only related to protected regions and their related mutexes and read-write locks or atomic variables. MySQL scalability is also affected by the type of scalability issues normally found in high-performance computing. When developing MySQL Cluster 7.2 and its scalability enhancements we encountered the same type of problems as we discovered in MySQL 5.6, so I'll describe the type of issues here.

In a modern server there are three levels of CPUs, there are CPU threads, there are CPU cores and there are CPU sockets. A typical high-end server of today can have 4 CPU sockets, 32 CPU cores and 64 CPU threads. Different vendors name this building blocks slightly differently but from a SW point of view it's sufficient to consider these 3 levels.

MYSQL CLUSTER 7.5 INSIDE AND OUT  
Buy the new book on MySQL Cluster  
Bound version

Paperback version

E-book version

INSPIRATIONAL MESSAGES OF THE WEEK

A call to democracy

Sense and Sensibility and Experiments

Periods in life

Achieving Perfection

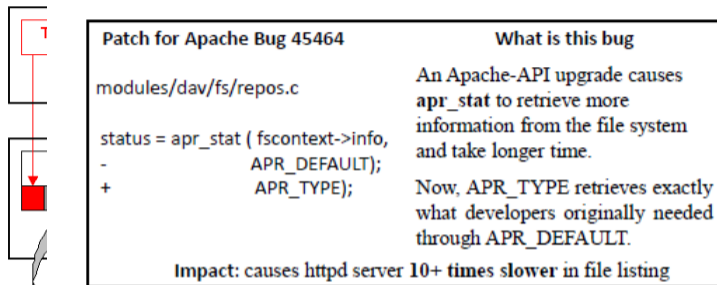
Easter Message

FOLLOWERS

Followers (120) Next



# Performance Bugs



**Patch for Apache Bug 45464**

```
modules/dav/fs/repos.c
```

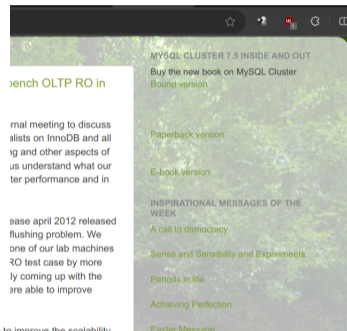
```
status = apr_stat ( fscontext->info,  
-                 APR_DEFAULT);  
+                 APR_TYPE);
```

**What is this bug**

An Apache-API upgrade causes `apr_stat` to retrieve more information from the file system and take longer time.

Now, `APR_TYPE` retrieves exactly what developers originally needed through `APR_DEFAULT`.

**Impact: causes httpd server 10+ times slower in file listing**



MYSQL CLUSTER 7.5 INSIDE AND OUT

Buy the new book on MySQL Cluster Bound version

Paperback version

E-book version

INSPIRATIONAL MESSAGES OF THE WEEK

A call to democracy

Sense and Sensibility and Experiments

Periods in life

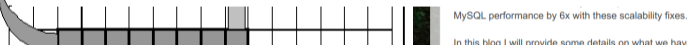
Achieving Perfection

Easter Message

bench OLTP RO in

mal meeting to discuss  
lists on InnoDB and all  
tg and other aspects of  
us understand what our  
ter performance and in

base april 2012 released  
flushing problem. We  
one of our lab machines  
RO test case by more  
ly coming up with the  
are able to improve



**MySQL Bug 38941 & Patch**

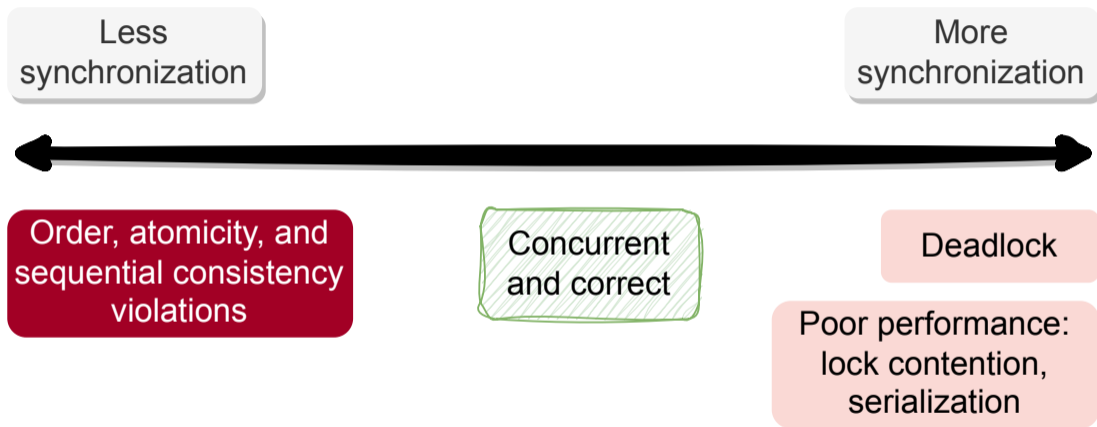
```
int fastmutex_lock (fmutex_t *mp){  
...  
- maxdelay += (double) random();  
+ maxdelay += (double) park_rng();  
...  
}
```

**What is this bug**

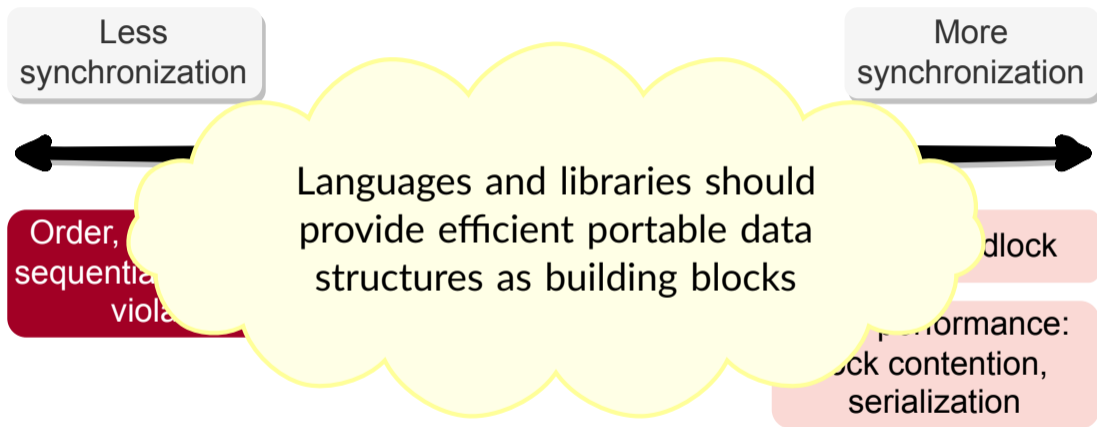
`random()` is a serialized global-mutex-protected libc function.

Using it inside `fastmutex` causes **40X slowdown** in users' experiments.

# Challenges with Concurrent Programming



# Challenges with Concurrent Programming



We will focus on performance aspects!

# Automated Parallelization with Compiler Support

```
// Disable optimizations
void serial(const float *A, const float *B, float *C) {
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
        C[i] = C[i] + C[i];
    }
}

void omp_parallel(const float * A, const float * B, float * C) {
    // Enable auto-parallelization with threads with OpenMP
    #pragma omp parallel for num_threads(omp_get_num_procs())
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
        C[i] = C[i] + C[i];
    }
}
```

# Automated Parallelization with Compiler Support

```
// Disable optimizations
void serial(const float *A, const float *B, float *C) {
    for (int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
        C[i] = C[i] + C[i];
    }
}
```

```
) g++ -O0 -fopenmp omp-parallelization.cpp
```

```
) ./a.out
```

```
Reference Version: Vector Size = 268435456, Approximately 0.582 GFLOPS; Time = 0.923 sec
```

```
OpenMP Version: Vector Size = 268435456, Approximately 2.228 GFLOPS; Time = 0.241 sec
```

```
for (int i = 0; i < N; i++) {
    C[i] = A[i] + B[i];
    C[i] = C[i] + C[i];
}
}
```

# Loop Transformations to Enable Parallelization

## Thread Parallelism

```
// N and M are very large values  
  
// Parallelize loop j with threads  
for (int j = 1; j < N; j++) {  
    for (int i = 1; i < M; i++) {  
        A[i][j] = A[i-1][j] + B;  
    }  
}
```

## Data Parallelism

```
// N and M are very large values  
  
for (int i = 1; i < M; i++) {  
    // Parallelize loop j with SIMD  
    // instructions  
    for (int j = 1; j < N; j++) {  
        A[i][j] = A[i-1][j] + B;  
    }  
}
```

# How to Write Efficient and Scalable Programs?

Good choice of algorithms and data structures

Determines the number of operations executed

Code that the compiler and architecture can effectively optimize

Determines the number of instructions executed

Proportion of parallelizable and concurrent code

Amdahl's law

Specialize to the target architecture platform

Memory hierarchy, cache sizes, new features like AMX



# References



R. Allen and K. Kennedy. *Optimizing Compilers for Multicore Architectures*. Sections 1.1–1.6, Morgan Kaufmann.