Simplifying Parallel Programming

Ulrich Drepper
Consulting Engineer, Red Hat
2010-6-25
The Problem
The Problem
The Problem
The Problem
The Problem
The Problem
The Reason

\[ E = C \times V^2 \times f \]
More Correctly

\[ E = C \times V (f)^2 \times f \]
Use of Transistors

- Increasing frequency is out
- Two uses
  - More complex architecture
    - Handle existing instructions faster
    - More specialized instructions
  - Horizontal growth
    - More execution cores; or
    - Only more execution contexts

Requires Parallelism!
Cost of Too Little Parallelism

- Idealized Amdahl's Law

\[ S = \frac{1}{(1-P) + \frac{P}{N}} \]

- Problems
  - \( P \) too small
  - \( N \) is steadily growing
- Formula is unrealistic though…
A More Realistic Formula

- Extended Amdahl's Law with Overhead

\[ S = \frac{1}{(1-P)(1+O_s) + \frac{P}{N}(1+O_p)} \]

- Parallelization is not free
  - Most of the time not even for serial code
  - The results are not *that* bad…
Even with Overhead P=0.6

- Even with 40% overhead not that much slower
- Speed-up from two threads on
  - Eleven threads for 10x slowdown
Programming Goals

\[ S = \frac{1}{(1-P)(1+O_S) + \frac{P}{N}(1+O_P)} \]

- Two goals: 1. ease parallel programming to increase \( P \)
  2. reduce \( O_S \) and \( O_P \)
Getting Parallelism

- Multi-process Pipeline
Problems with Pipelines

- Marshalling needed for transmission
- Protocol standardization required
- Limited buffer sizes
  - Lots of scheduling needed
- Program need to be designed for pipeline
  - Extending an existing program not easy
  - Major code restructuring needed
Problems with Pipelines

- Marshalling needed for transmission
- Protocol standardization required
- Limited buffer sizes
  - Lots of scheduling needed
- Program need to be designed for pipeline
  - Extending an existing program not easy
  - **Major code restructuring needed**
Simple Program Structure

Process

Common Data 1

Function 1

Function 2

Dataset 1

Dataset 2

Dataset 3

Common Data 2
“Easy” Fix

Process

Common Data 1

Common Data 2

Thread 1

Thread 2

Dataset 1

Dataset 2

Dataset 3
“Easy” Fix

Process

Common Data 1

Common Data 2

Thread 1

Thread 2

Dataset 1

Dataset 2

Dataset 3
It seems easy...
It seems easy...

Mutexes are hard to use right!!!
Explicit Multi-Threading

• Ill-conceived solution
  • Yes
    • Existing code can be reused, easier to set up
    • High-bandwidth inter-thread communication
    • On some OSes context switching faster
  • But:
    • Fragile programming model (one thread dies, the process dies)
    • Memory handling mistakes have global effects
    • Unix model initially not designed for multiple threads
Explicit Multi-Threading

- Ill-conceived solution
  - Yes
    - Existing code can be reused, easier to set up
    - High-bandwidth inter-thread communication
    - On some OSes context switching faster
  - But:
    - Fragile programming model (one thread dies, the process dies)
    - Memory handling mistakes have global effects
    - Unix model initially not designed for multiple threads

Hard to write correct code! High Cost!
Measures

Process

Common Data 1

Common Data 2

Thread 1

Thread 2

Mutex

Mutex

Dataset 1

Dataset 2

Dataset 3

Reuse

Bandwidth

Context Cost

Ease Program

Fragile

Overwrites

Unix model

Error Prone

SUMMIT JBoss WORLD
PRESENTED BY RED HAT
Alternative 1: fork and Shared Memory

- All in POSIX:

```c
int fd = shm_open(name, O_RDWR|O_CREAT);
ftruncate(fd, size);
p = mmap(NULL, size, PROT_READ|PROT_WRITE,
         MAP_SHARED, fd, 0);
if (fork() == 0)
    ...
```
fork and Shared Memory

Diagram:
- State Data
- Process 1
- Dataset 1
- Mutex
- State Data
- Process 2
- Dataset 2
- Dataset 3
fork and Shared Memory

- State Data
- Process 1
- Dataset 1
- Mutex
- Dataset 2
- Process 2
- Dataset 3
- State Data
fork and Shared Memory

- Dataset 1
- Dataset 2
- Dataset 3

- Process 1
- Process 2

- State Data
- State Data

- Mutex

- Reuse
- Bandwidth
- Context Cost
- Ease Program

- Fragile
- Overwrites
- Unix model
- Error Prone

- Unix model
- Error Prone
Alternative 2: fork and Linux Pipes

- Linux extensions, not POSIX (yet 😊)
- Can be zero-copy
- Use if just transferring data without inspection
- splice: transfer from file descriptor to pipe
- tee: transfer between pipes and keep data usable
- vmsplice: transfer from memory to pipe
fork and Linux Pipes

State Data

Process 1

Dataset 1

State Data

Pipe

Process 2

Dataset 3
fork and Linux Pipes

State Data → Process 1 → Dataset 1

State Data → Process 2 → Dataset 3

Pipe

Reuse
Bandwidth
Context Cost
Ease Program

Fragile
Overwrites
Unix model
Error Prone

Unix model
Ease Program

Error Prone
Alternative 3: Thread Local Storage

- Use thread-local storage
  - Very much simplifies use of static variables
  - No more false sharing of cache lines

```c
__thread struct foo var;
```
Thread Local Storage

- Dataset 1
- Dataset 2
- Dataset 3
- Thread 1
- Thread 2
- Common Data 1
- Common Data 2
- Mutex

Process
Thread Local Storage

Process

- Common Data 1
- Common Data 2
- Thread 1
- Thread 2
- Dataset 1
- Dataset 2
- Dataset 3

Mutex
Thread Local Storage

- Common Data 1
- Common Data 2
- Thread 1
- Thread 2
- Mutex
- Dataset 1
- Dataset 2
- Dataset 3

Process

- Reuse
- Bandwidth
- Context Cost
- Ease Program
- Fragile
- Overwrites
- Unix model
- Error Prone
Alternative 4: OpenMP

- Language extension to C, C++, Fortran languages
- Implements many thread functions with very simple interface for
  - Thread creation (controlled)
  - Exclusion
  - Thread-local Data
OpenMP
OpenMP

Process

Common Data 1
Common Data 2
Thread 1
Thread 2
Dataset 1
Dataset 2
Dataset 3
OpenMP

- Dataset 1
- Dataset 2
- Dataset 3

- Thread 1
- Thread 2

- Common Data 1
- Common Data 2

- Annotation

- Reuse
- Bandwidth
- Context Cost
- Ease Program

- Fragile
- Overwrites
- Unix model
- Error Prone
Alternative 5: Transactional Memory

- Extensions to C and C++ languages
- Can help to avoid using mutexes
  - Just source code annotations
  - No more deadlocks!!
  - Fine-grained locking without the problems
- Slow as pure software solutions
  - Hardware support on the horizon
Transaction System

Portfolio Data

- Person 1
- Person 2
- Person N

Bank 1
Bank 2
Bank N
Transaction System

Deduct Shares from Person 1

Portfolio Data

Person 1

Person 2

Person N

Bank 1

Bank 2

.......... Bank N
Transaction System

- Deduct Shares from Person 1
- Add Shares to Person 2

Portfolio Data → Person 1 → Person 2 → Person N

Bank 1
Bank 2
Bank N
Transaction System

- Deduct Shares from Person 1
- Add Shares to Person 2
- Add to Person 1 Account

Portfolio Data

Person 1

Person 2

Person N

Bank 1

Bank 2

Bank N
Transaction System

- Deduct Shares from Person 1
- Add Shares to Person 2
- Subtract from Person 2 Account
- Add to Person 1 Account

Portfolio Data

Bank 1

Bank 2

Bank N
Trying to Parallelize

Portfolio Data → Person 1 → Bank 1
Portfolio Data → Person 2 → Bank 2
Portfolio Data → Person N → Bank N
Trying to Parallelize

Portfolio Data

Bank 1

Person 1

Bank 2

Person 2

Bank N

Person N

Lock Domain
Not What We Want

![Graphs showing runtime vs. threads for Single Core i7 and Opteron NUMA processors.](image-url)
Trying to Parallelize

Portfolio Data

Person 1

Person 2

Person N

Bank 1

Bank 2

.............

Bank N
Trying to Parallelize Portfolio Data

Bank 1

Bank 2

Bank N

Portfolio Data

Person 1

Person 2

Person N

Lock Domain
Somewhat Better But…

![Graph comparing runtimes between Single Core i7 and Opteron NUMA for different thread counts. The graph shows a decrease in runtime for Single Core i7 as the number of threads increases, whereas Opteron NUMA shows an increase in runtime with more threads.]
Transactional Memory

Process

Common Data 1

Mutex

Thread 1

Mutex

Thread 2

Mutex

Dataset 1

Dataset 2

Dataset 3
Transactional Memory

- Dataset 1
- Dataset 2
- Dataset 3

Common Data 1
Common Data 2
Thread 1
Thread 2

Process
Conclusion

- Abilities to exploit hardware are there
  - Explicit threading only for experts
- But there is a lot of help
  - Use processes, not threads; or
  - If threads are used combine
    - Thread-local storage
    - Implicit thread creation
      - OpenMP
      - Futures
    - Transactional memory
Questions?