A Middleware for Concurrent Programming in MPI Applications

Tobias Berka, Helge Hagenauer and Marian Vajteršic

September 13, 2011

Outline

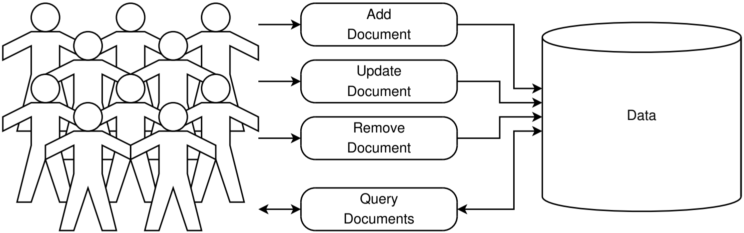
Introduction

Emergent Parallel Applications

- Parallelism is abundant in today's data centers:
  - Multi-core CPUs,
  - High-bandwidth low-latency interconnection networks,
  - Accelerator hardware.
- Exciting new applications in today's information economy:
  - Information retrieval (i.e. search),
  - Online analytical processing,
  - Recommender systems,
  - Data mining.

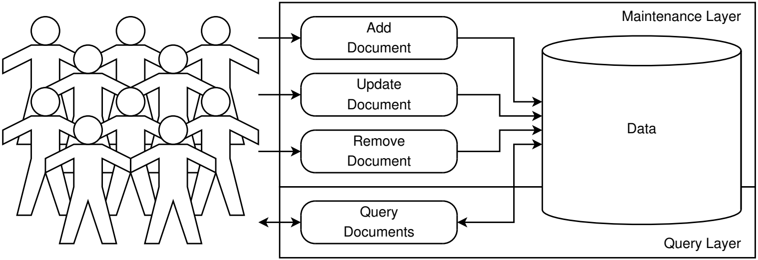
Use Case: Parallel Search Engine

- Requirements beyond the classic batch-job operation:



Use Case: Parallel Search Engine

- We group similar operations – short and long:



The Need for Concurrency

- Multi-User Operation
  - Multiple users,
  - Single back-end,
  - Single data base...

  $\Rightarrow$ We need concurrency!
- Both layers can be used concurrently,
  - At the same time:
    - Answer queries,
    - Modify the data base,

  $\Rightarrow$ We need concurrency!

Programming Model

How do we implement concurrent activities?

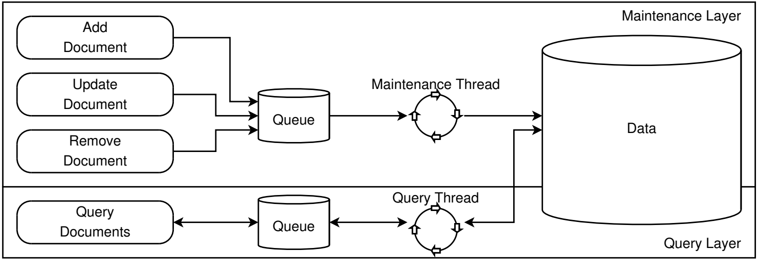
- Operations and queues:
  - Data structure to describe operations,
  - Queue holds operations,
  - "Main loop" pops operations and processes them.
- Threads:
  - One activity = one thread.

The pros and cons...

- Operations and queues:
  - \+ Efficient,
  - \- No true concurrency,
  - \- Cannot process operation and receive independent messages,
- Threads:
  - \- Context switching overhead,
  - \- Shared data requires locking,
  - \+ Very tidy abstraction,
  - \+ Compositional (can always add more threads).

Use Case: Parallel Search Engine

- Let's use threads to implement these concurrent activities:

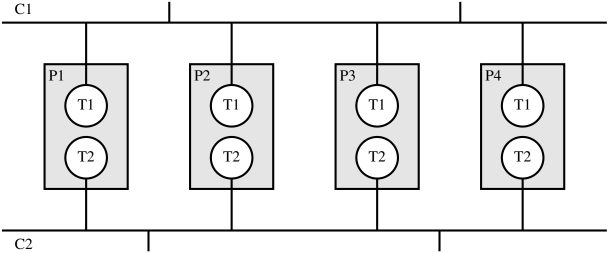


Programming Abstraction

- Key abstraction: thread collective,
- Goals:
  - Encapsulate concurrent activities,
  - Isolate concurrent communication,
  - Unify and simplify the design.
- Conflicting objectives:
  - Safety and ease of programmability,
  - Performance.

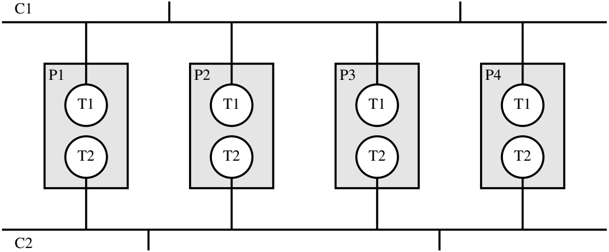
Thread Collectives

- Creates a new thread within every MPI process (T1 $\rightarrow$ T2),
- Assigns a copy of the MPI communicator ($C1 \rightarrow C2$),



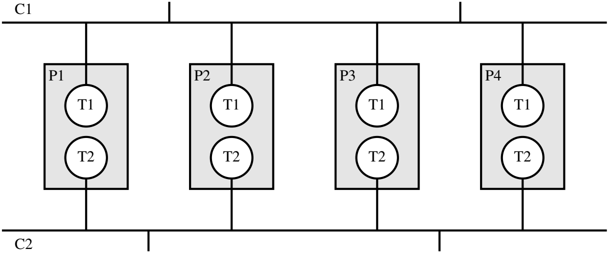
Thread Collectives

- Encapsulates computation: thread function(s) for T2,
- Isolates communication: communicator $C2$.



Parallel Search Engine

- $P1$–$P4$ each hold a part of all documents,
- $T1$s: Answer queries (query layer),
- $T2$s: Add, remove or update documents (maintenance layer).



What do we get?

- Simple, ready-made abstraction,
- Encapsulate & isolate,
- Compositional,
- Caveat: synchronization and locking.

The MPI Threads API

The MPIT Interface Definition

- Additional layer of middleware to provide what we need,
- Designed as a library for compatability
  (not a new programming lanugage),
- The "MPI Threads" (MPIT) interface definition,
- Written as a single C header file (157 physical SLOC[1]).

[1] According to David A. Wheeler's "SLOCCount".

Constructs and Features

- Thread collectives,
  - One thread within every MPI process,
  - Separate MPI communicator,
- Conventional threads,
  - Portable thread interface,
  - We get it "for free" – we have all of the machinery.
- Process-local synchronization constructs,
  - Mutex locks, condition variables, semaphores and barriers,
  - Specifies reliable semantics.

Performance Overhead

- Additional layer of indirection (1 additional function call),
- Additional error and consistency checks:
  - Condition variable checks spurious wake-up,
  - Barrier verifies thread identity.

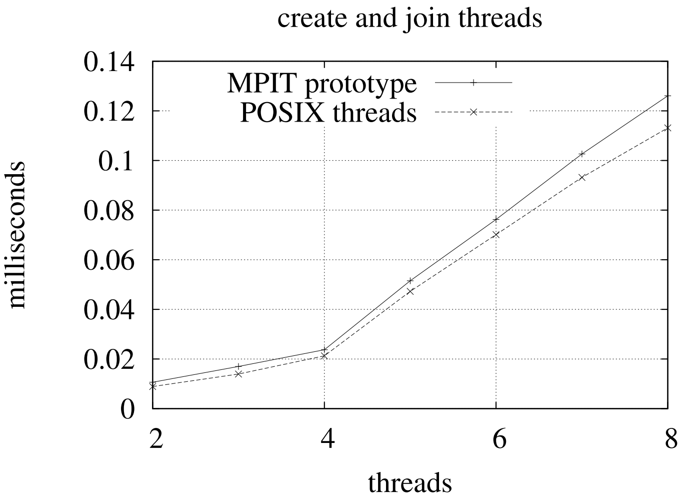
⇒ Execution time overhead.

- MPIT prototype implemented on top of POSIX threads (2,650 physical SLOC[2]).

[2] According to David A. Wheeler's "SLOCCount".

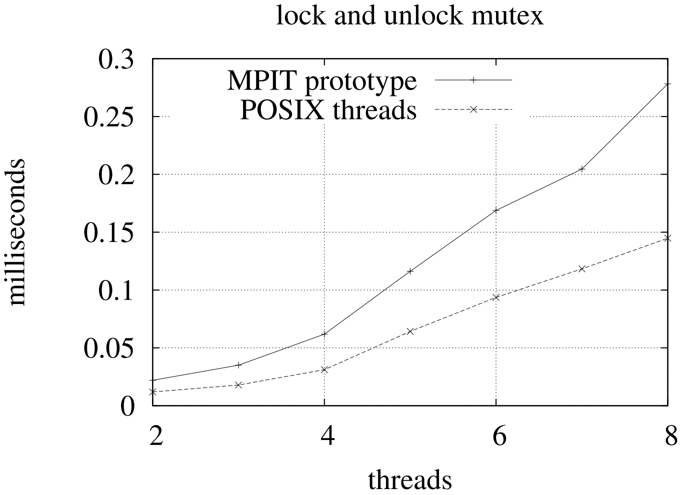
Thread Creation

- Difference: additional indirection.



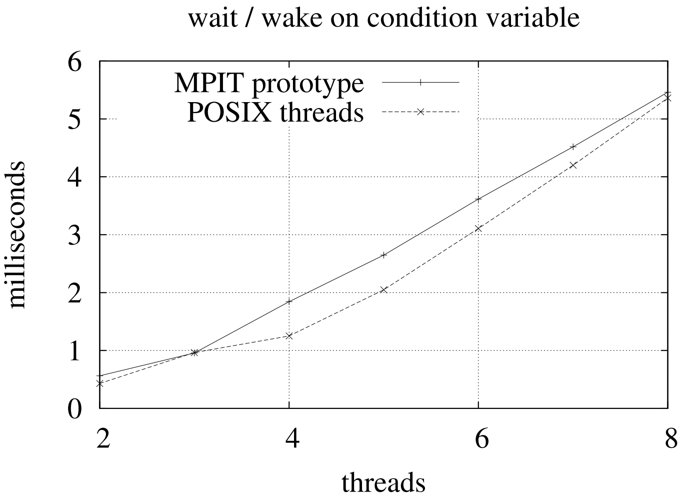
Lock / Unlock Mutex

- Difference: additional checks.



Wait / Wake on Condition Variable

- Difference: indirection & checks (different time scale).



Summary & Conclusions

Summary & Conclusions

- Parallel programs may require additional concurrency,
- New abstraction: thread collectives,
- MPIT interface specification,
- Performance penalty is acceptable...
- ...unless too many locks are used.

Thank you!

Thank you!