A Thread Performance Comparison:
Windows NT and Solaris on A Symmetric Multiprocessor

Fabian Zabatta and Kevin Ying
Brooklyn College and CUNY Graduate School
{fabian,kevin}@sci.brooklyn.cuny.edu
Outline

• Introduction
• NT and Solaris Implementation
• Experiments
• Conclusions
Decreasing Cost of Parallel Hardware

- **PC Based SMP with 4 PPros 200MHz, with 512 MB RAM**: $138,000 in 1994
- **PC Based SMP with 4 Pentium II Xeon 400 MHz, with 512 MB RAM**: $15,000 in 1997
- **A basic IBM SP2 with 2 66-70 MHz processors, with 128 MB RAM**: $13,000 in 1999
Kernel-Level Objects of Execution

Classic Process

• One unit of control

Modern Process

• Divided into sub-objects
• Each sub-object has its own context
• Each sub-object functions independently
• Each sub-object shares the same address space and resources with sub-objects of the same process
Advantages of Design

• Overlap Processing
• Parallel Execution
• Scalability
• Communication
• Inexpensive
• Well Structured Programming Paradigm.
Generic Thread Architecture

The Operating System Kernel

Processor Structure

User-Level Object of Execution

Kernel-level Object of Execution
NT’s Thread Architecture

The Windows NT Kernel

Process Structure

Application Code

Threads

Fibers

Global Data
Solaris’s Thread Architecture

The Solaris Kernel

Process Structure

Scheduler

Thread Library

Threads

LWPs

Application code, global data, etc.
## Implementation Comparison

<table>
<thead>
<tr>
<th>Implementation Model</th>
<th>Windows NT</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel-level</td>
<td>Thread</td>
<td>LWP</td>
</tr>
<tr>
<td>User-level</td>
<td>Fiber</td>
<td>Thread</td>
</tr>
<tr>
<td>Hybrid model</td>
<td>User Must Implement</td>
<td>Thread Library</td>
</tr>
<tr>
<td>Scheduling Model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>Fibers (user controlled)</td>
<td>Preemptive Priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non Time Sliced</td>
</tr>
<tr>
<td>Kernel</td>
<td>Preemptive Priority</td>
<td>Preemptive Priority</td>
</tr>
<tr>
<td></td>
<td>Time Sliced</td>
<td>Time Sliced</td>
</tr>
<tr>
<td>Programming Interface</td>
<td>Win32</td>
<td>UNIX International</td>
</tr>
<tr>
<td>Multiplexing Model</td>
<td>One-One (Fiber: many-many)</td>
<td>Variation of many-many</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Coexist of one-one)</td>
</tr>
</tbody>
</table>
Motivation

Test each system’s chosen thread API to discover the performance impact of each design on various applications.

• NT: Thread

• Solaris: Bound, Unbound and CL=4
Experiments

*Measure and Compare:*

1. Number of allowable kernel threads.
2. Execution time of thread creation.
3. Execution time of thread creation on a heavily loaded system.
4. Performance of CPU intensive threads that do not require synchronization.
5. Performance of CPU intensive threads that require extensive synchronization.
7. Performance of threads that have bursty processor requirements.
Parameters

**Hardware**

• SMP Machine (Sag Electronics) with 4-200 MHz Pentium Pros (256K Cache Each)

• 512 MB RAM & 4 GB SCSI Hard Drive

**Software**

• NT Server 4.0 (Service Pack 3) & Solaris 2.6

• GNU gcc Version 2.8.1 Compiler
## 1. Thread Limits

<table>
<thead>
<tr>
<th>Description</th>
<th>NT</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Threads Created</td>
<td>9817</td>
<td>2294</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>68MB</td>
<td>19MB</td>
</tr>
<tr>
<td>Execution Time (sec.)</td>
<td>24.12</td>
<td>2.68</td>
</tr>
</tbody>
</table>
2. Thread Creation Speed

![Graph showing thread creation speed comparison between Solaris Bound, Solaris CL=4, Solaris Unbound, and NT systems. The x-axis represents the number of threads created, while the y-axis represents execution time in seconds. Different markers distinguish each system type.](image-url)
3. Thread Creation of CPU Intensive Threads

The graph illustrates the execution time (in seconds) for creating threads as a function of the number of threads. The execution time increases significantly as the number of threads increases, with two distinct curves:

- The NT curve, represented by squares, shows a relatively linear increase in execution time until a certain point, after which the execution time increases more rapidly.
- The Solaris Bound curve, represented by diamonds, shows a sharper increase in execution time from the start.

The graph highlights the performance impact on both NT and Solaris systems as the number of threads grows, indicating potential bottlenecks or resource limitations with increased parallel processing.
4. No Synchronization-CPU Intensive Threads

• There is very few differences between NT threads and Solaris bound threads.

• Solaris thread library did not increase nor decrease the size of LWP pool for CL=4 and unbounded threads.

• CL=4 has equivalent performance to that of the bound threads.
  
  - This implies that additional LWPs did not increase the performance.

  - The time it takes Solaris’s thread library to schedule threads on LWPs is not a factor in performance.
5a. Extensive Synchronization (Process Scope)
5b. Extensive Synchronization (System Scope)

![Graph showing execution time vs. number of threads for different systems and thread bounds.]

- **NT Mutex**
- **Solaris Bound**
- **Solaris CL=4**
- **Solaris Unbound**
6. Parallel Search

We explored the classic symmetric traveling salesman problem (TSP). The problem was modeled with threads that required limited synchronization to perform a parallel depth-first branch and bound search.

- NT version of the TSP slightly outperformed the Solaris version. Both systems were able to achieve an almost linear speed up (3.9+).
This experiment tested the performance of threads that have bursty processor requirements. This is analogous to applications that involve any type of I/O, e.g. Networking or client/server applications.

- CL=4 showed a slightly better performance in comparison to NT’s threads or Solaris bound and unbound threads. This can be directly attributed to Solaris’s two-tier thread architecture.
Comparison Conclusions

• Both utilized multiprocessors and scaled well.
• Solaris’s design was more flexible at the cost of complexity.
• NT’s critical section outperformed Solaris’s mutex.
• Solaris’s mutex outperformed NT’s mutex.
• Solaris’s design excelled on tasks with bursty processor requirements.
Independent Performance Conclusions: Solaris

- Thread library’s automatic control of concurrency level is limited.

- Set the concurrency level to the number of processors and create unbounded threads when needed.
Independent Performance Conclusions: NT

• The number of threads should be roughly equal to the number of CPUs.

• When extensive intra-process synchronization is required use a “critical section”.
Closing Notes

• Threads are important and powerful programming tools.

• Differences exist on how they should be implemented.

• Differences in implementations are tradeoffs.

• Pthreads (POSIX): Standard thread API.