A Thread Performance Comparison:

Windows NT and Solaris on A Symmetric Multiprocessor

Fabian Zabatta and Kevin Ying

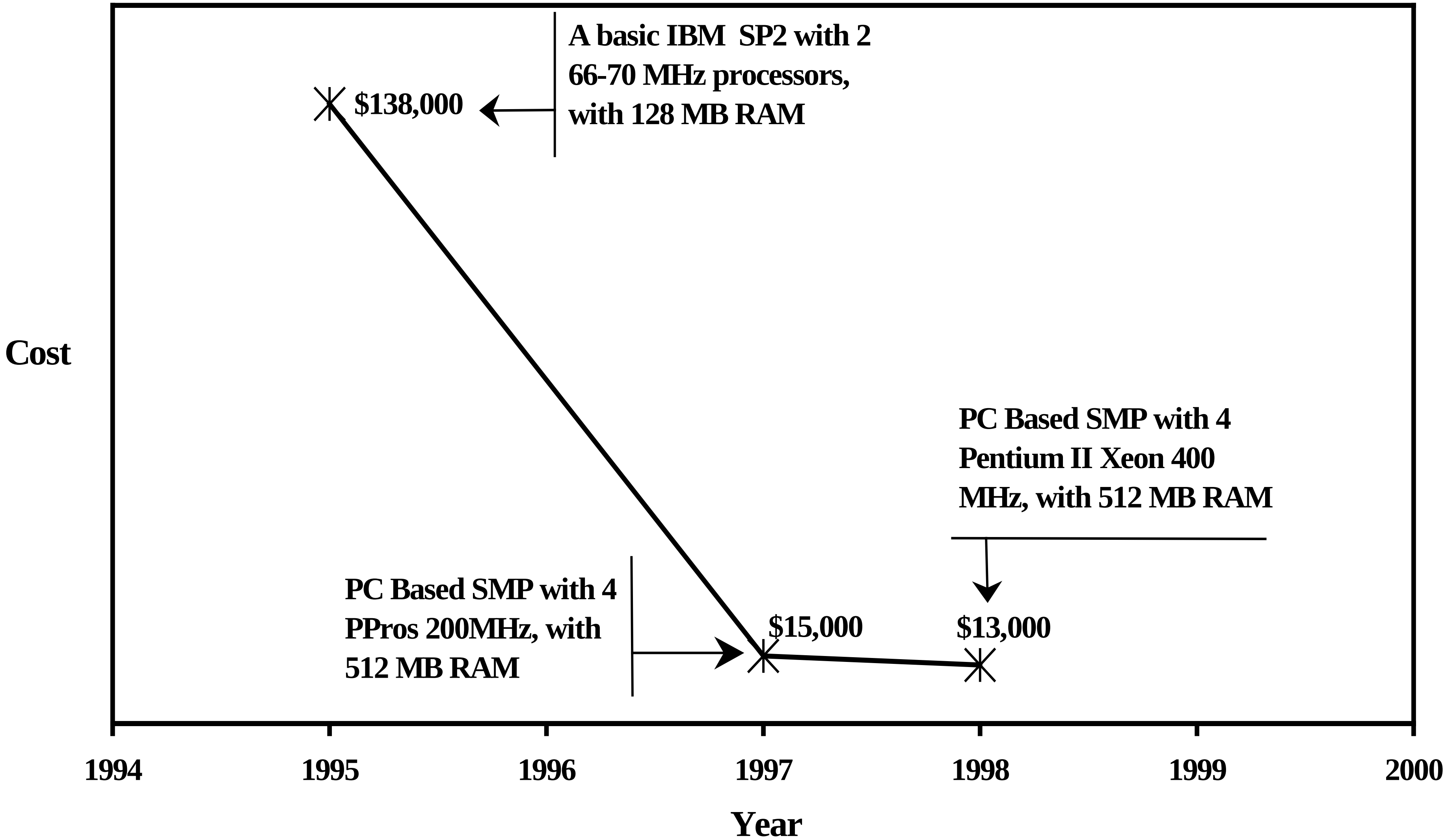
Brooklyn College and CUNY Graduate School

{fabian,kevin}@sci.brooklyn.cuny.edu

Outline

- Introduction
- NT and Solaris Implementation
- Experiments
- Conclusions

Decreasing Cost of Parallel Hardware



Kernel-Level Objects of Execution

Classic Process

- One unit of control

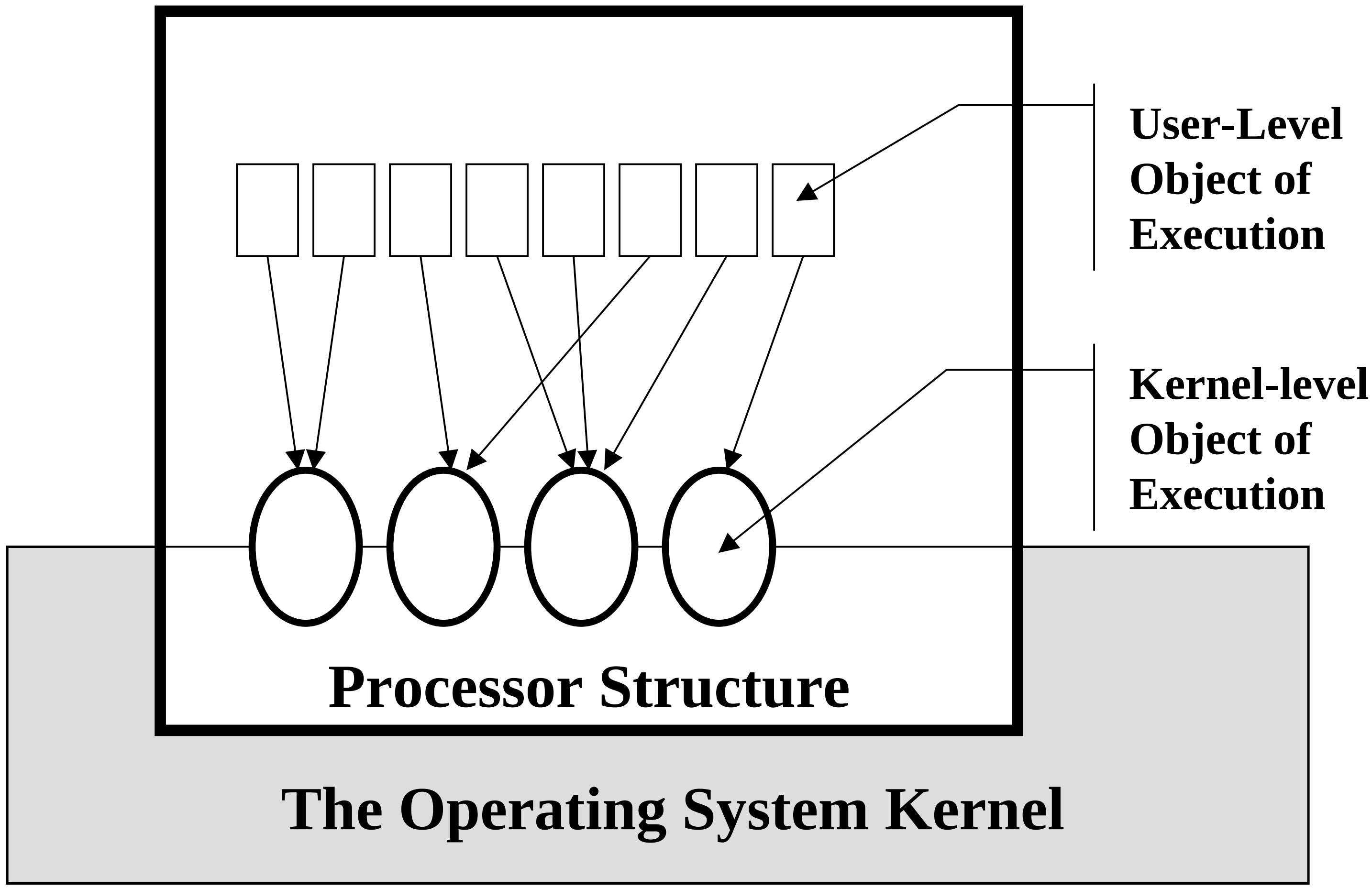
Modern Process

- 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

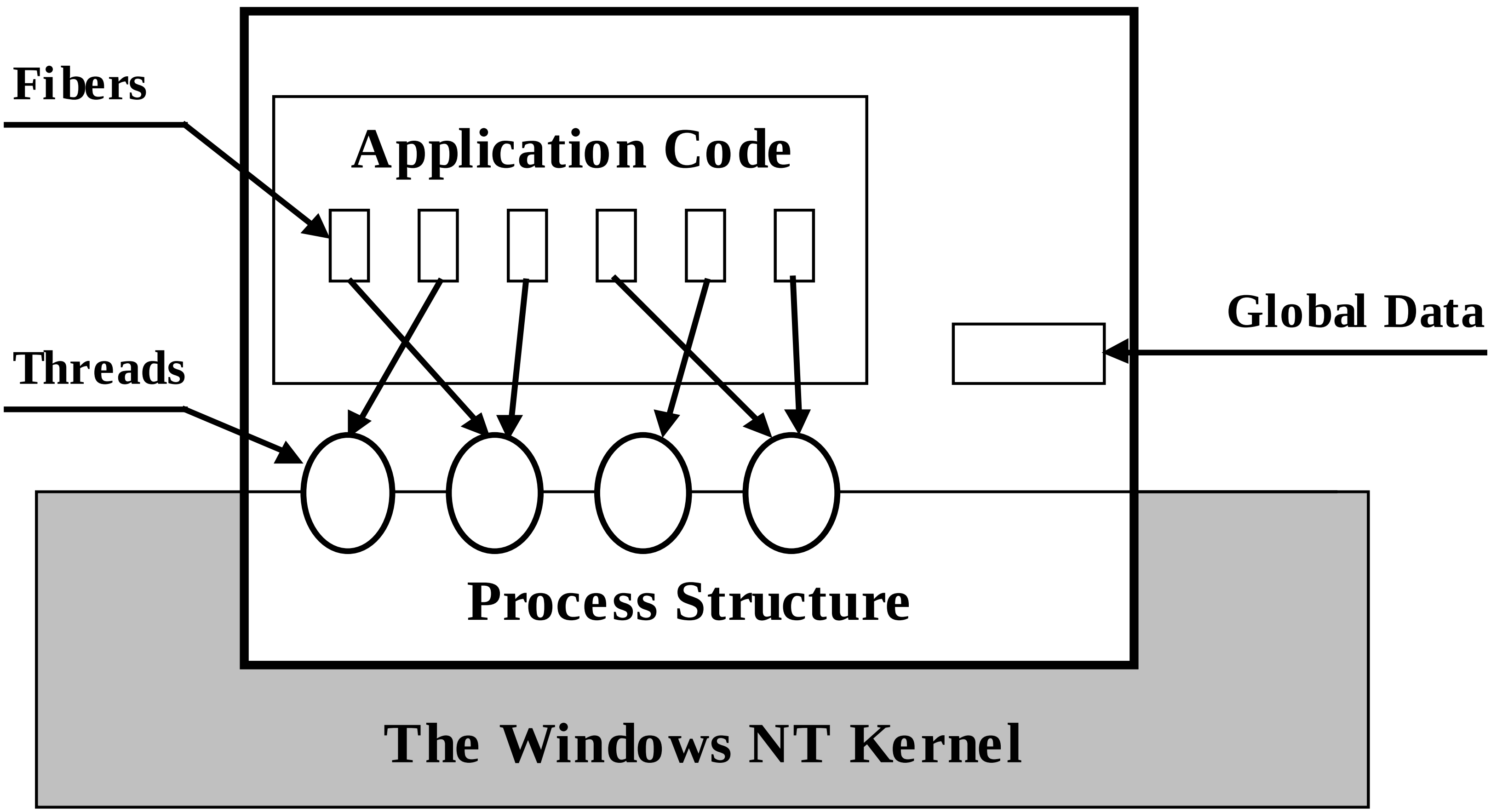
Advantages of Design

- Overlap Processing
- Parallel Execution
- Scalability
- Communication
- Inexpensive
- Well Structured Programming Paradigm.

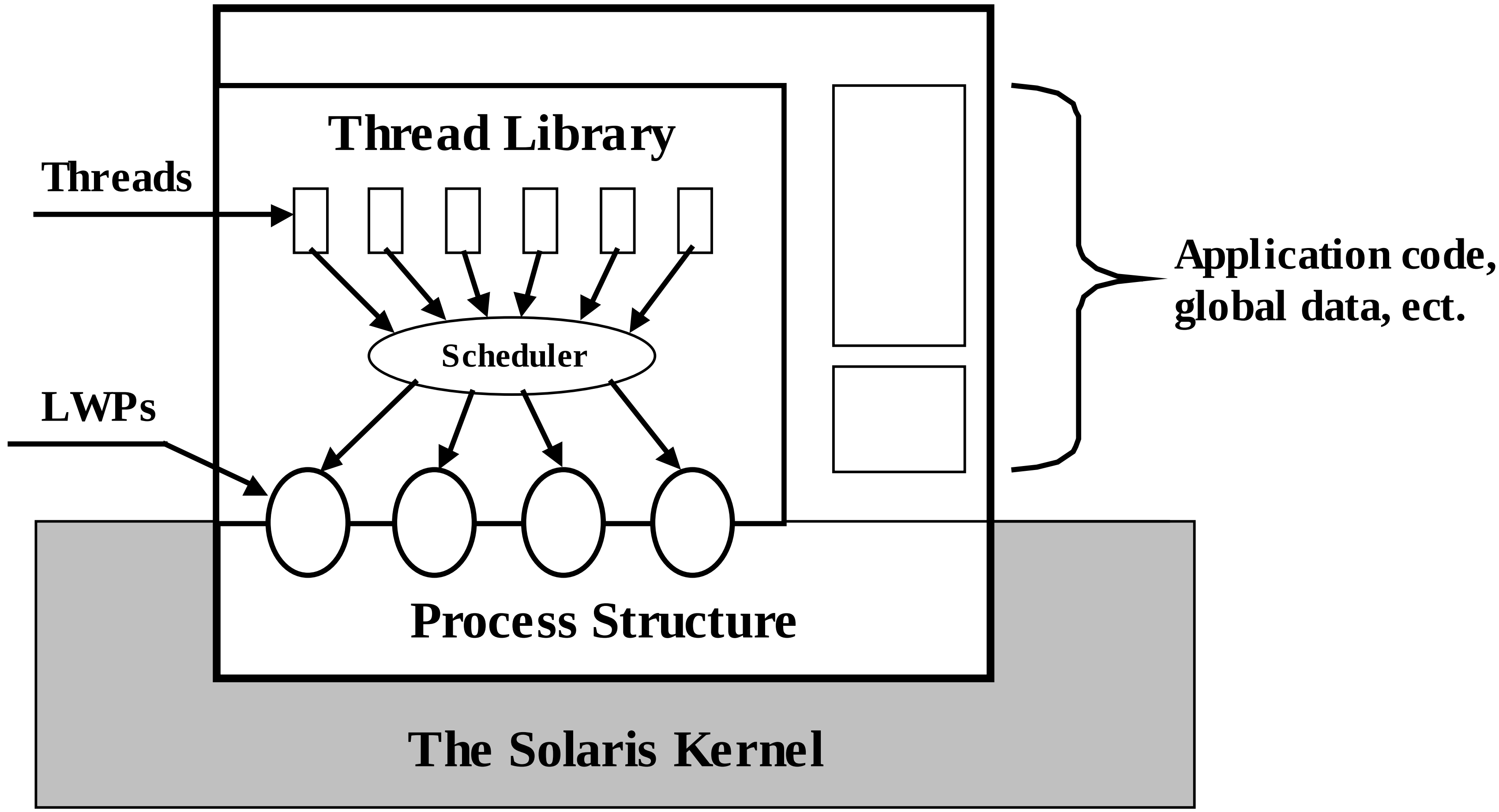
Generic Thread Architecture



NT's Thread Architecture



Solaris's Thread Architecture



Implementation Comparison

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

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.
6. Performance of threads on a parallel search.
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

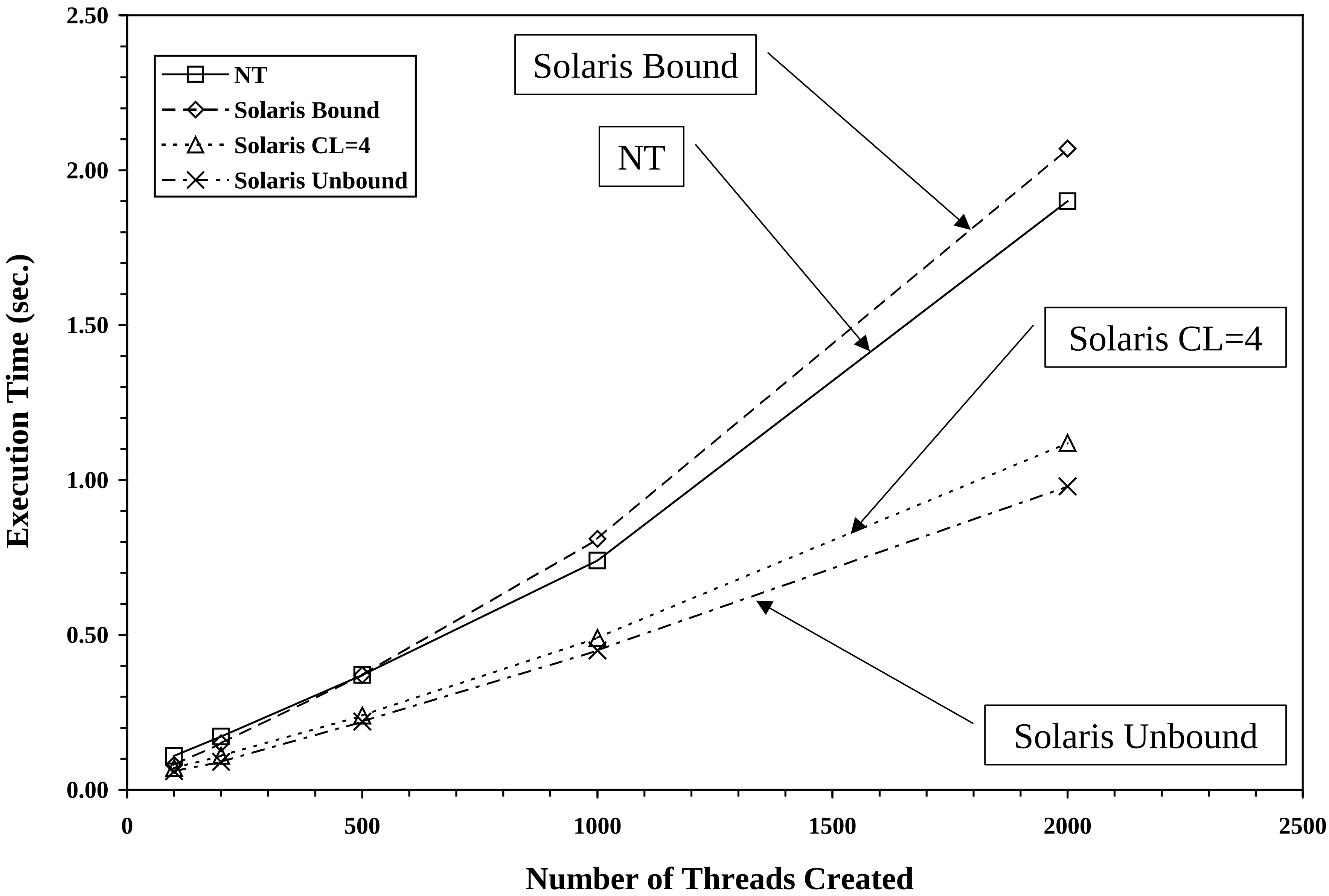
•NT Server 4.0 (Service Pack 3) & Solaris 2.6

•GNU gcc Version 2.8.1 Compiler

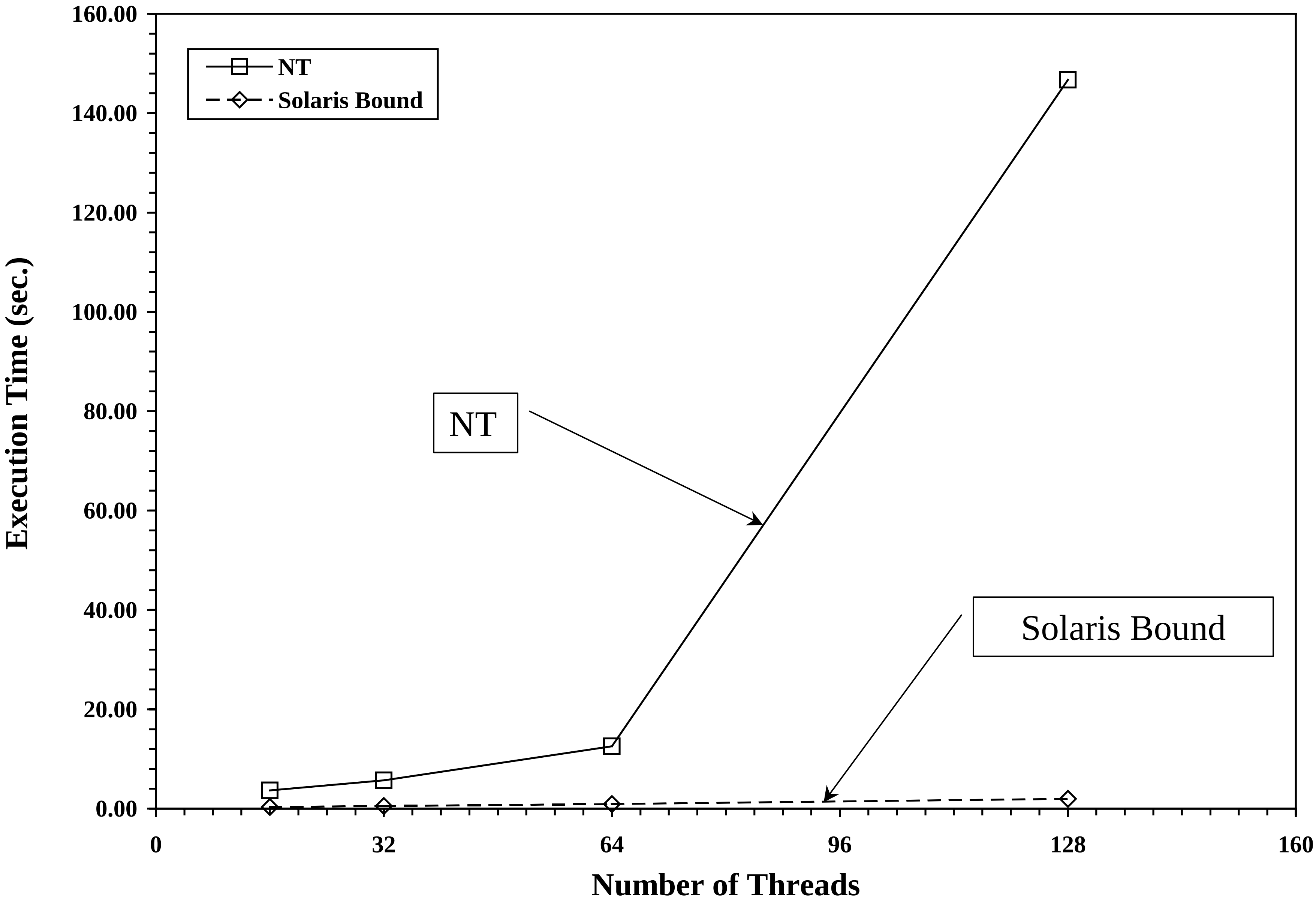
1. Thread Limits

Description	**NT**	**Solaris**
# of Threads Created	9817	2294
Memory Usage	68MB	19MB
Execution Time (sec.)	24.12	2.68

2. Thread Creation Speed



3. Thread Creation of CPU Intensive Threads



4. No Synchronization-CPU Intensive Threads

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

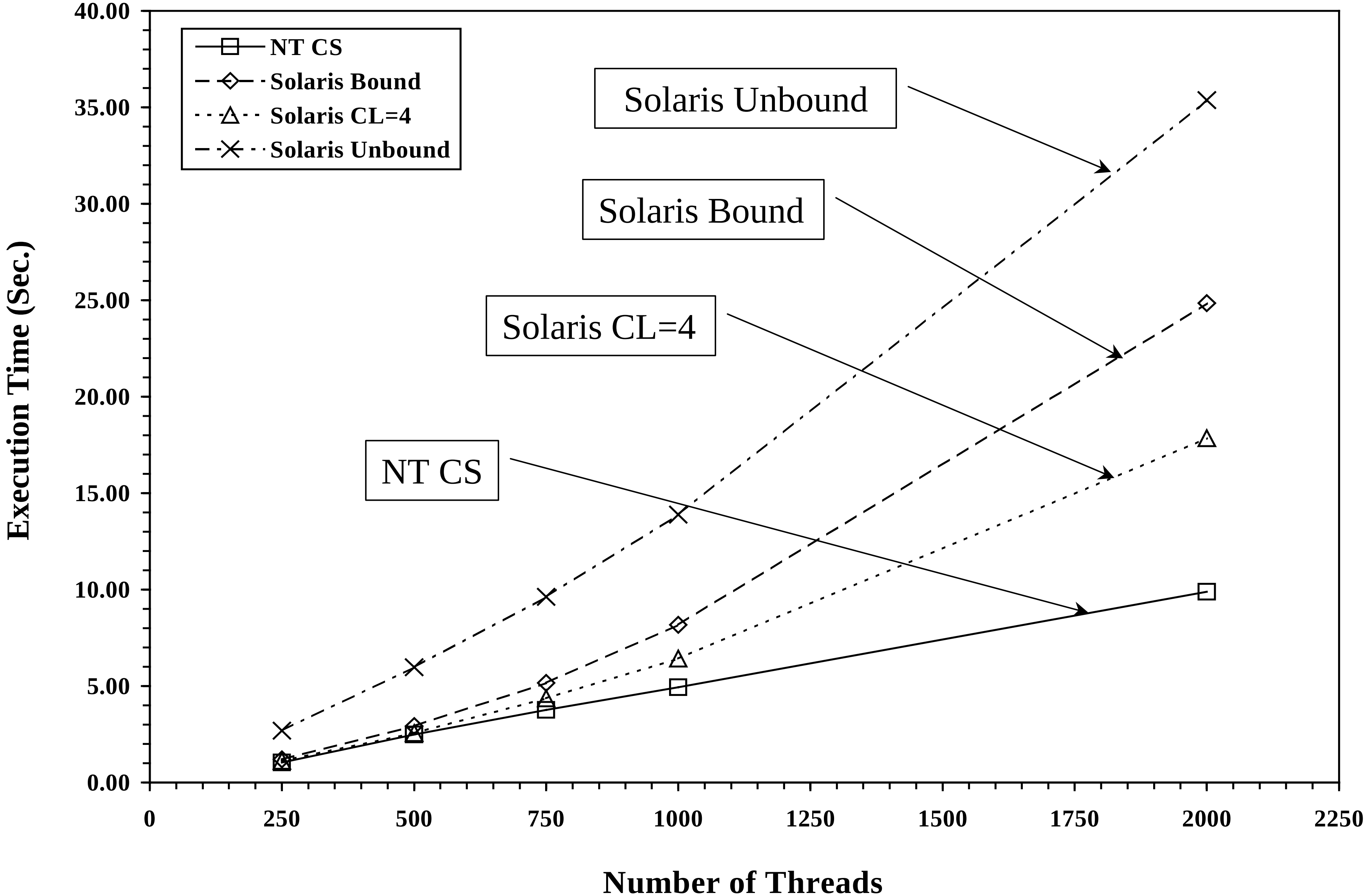
• Solaris thread library did not increase nor decrease the size of LWP pool for CL=4 and unbounded threads.

• CL=4 has equivalent performance to that of the bound threads.

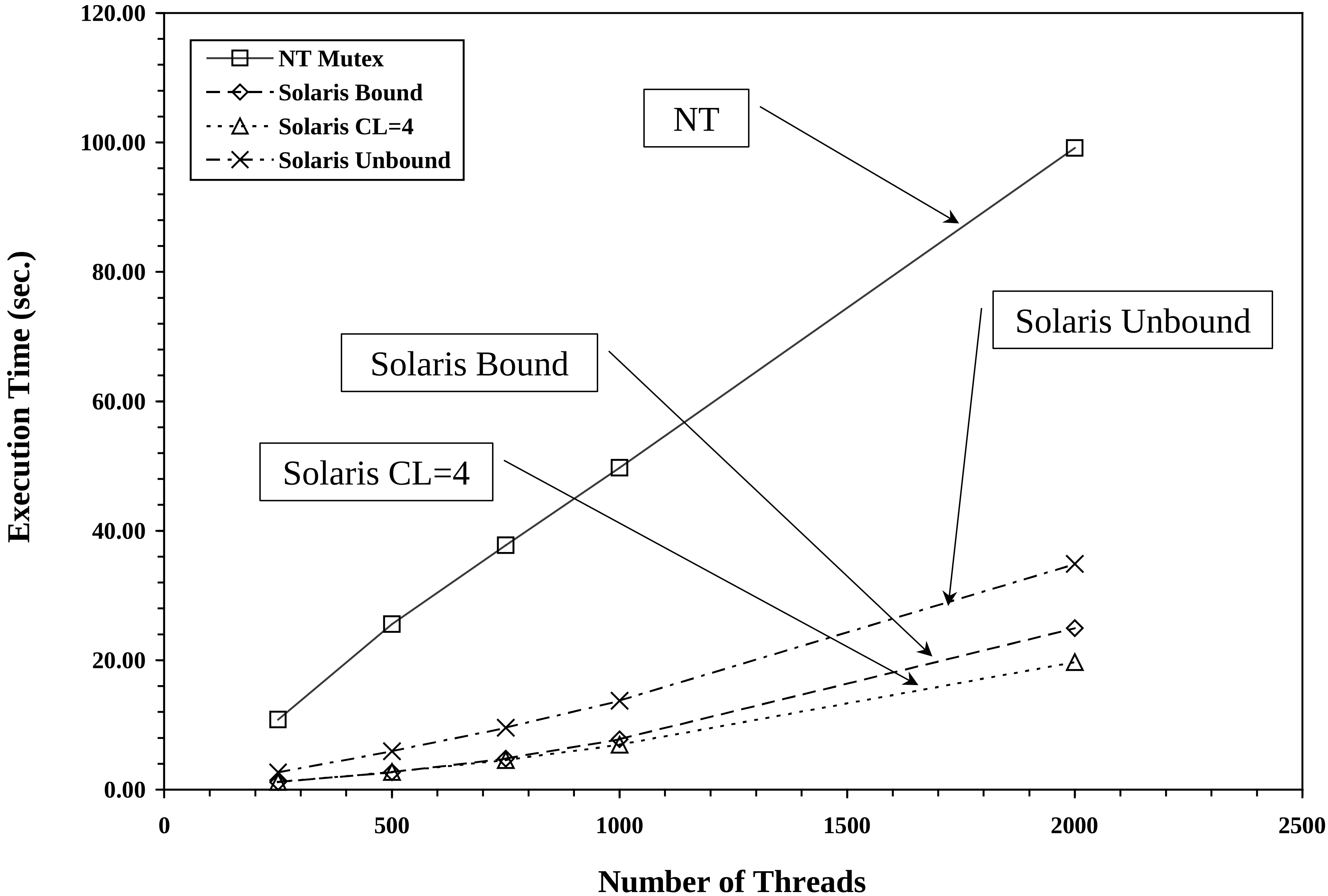
- This implies that additional LWPs did not increase the performance.

- 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)



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+).

7. Threads with CPU Bursts

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.