DEVELOPMENT ENVIRONMENT, BENCHMARKS, & OPTIMIZATIONS

Ms. Moran Tzafrir

TAU

Introduction

- This lecture is about the development cycle, coding, testing, & optimization.
- I’ll try to share my experience & education on those issues.
- But this lecture is also an *open discussion*, if you have something to contribute from your experience, please do so !
- In general this is a technical lecture, but the small details are less important. Please concentrate on the big picture. What to look for, what the important issues, and where are the pitfalls.

Lecture outline I

Hardware overview:

- Multicore machine we can access
- How to access
- Working at home

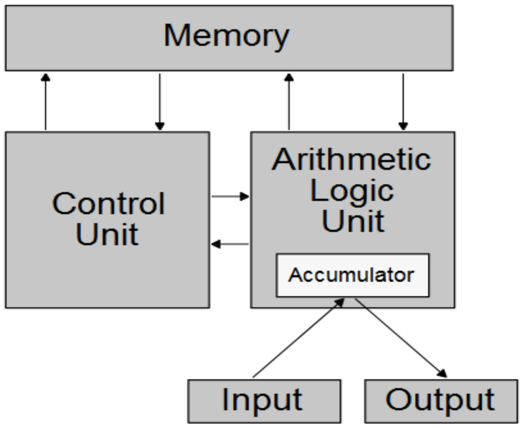
Software overview:

- **Java**
  - Compilation
  - Memory Model
  - Performance
  - Pitfalls
- **C++**
  - Compilation
  - Memory Model
  - Performance
  - Pitfalls
  - C++ Framework

Lecture outline II

- **Benchmarks:**
  - What we want to benchmark
  - What we can benchmark
  - Designing the benchmark
  - Different configurations
  - Different Hardware
  - Preparing the data to the benchmark
  - Pitfalls

Hardware - Von Neumann architecture



Know the limits

The von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data.

Von Neumann bottleneck The separation between the CPU and memory leads to the *von Neumann bottleneck*, the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory. In most modern computers, throughput is much smaller than the rate at which the CPU can work. This seriously limits the effective processing speed when the CPU is required to perform minimal processing on large amounts of data. The CPU is continuously forced to wait for needed data to be transferred to or from memory. Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem.

von Neumann syndrome most applications in massively parallel computing systems with thousands or tens of thousands of processors the performance can be less than hoped. Sometimes called a "supercomputing crisis" it is believed to be due to two factors. Firstly a hardware barrier in the efficiency in moving data, called the memory wall or von Neumann bottleneck. Secondly a fall in programmer productivity when faced with systems that are massively parallel, the difficulties in developing for parallelism (or thread-level parallelism in multi-core CPUs) when previously this was not an issue.

Reconfigurable computing paradox

use of reconfigurable computing on FPGAs produces faster results despite a lower speed, where, instead of moving data around, the locality of execution is optimized by placement and routing. This is also called a data-stream-driven anti machine paradigm.

Hardware: Multicore machine we can access

Two SPARC machines:

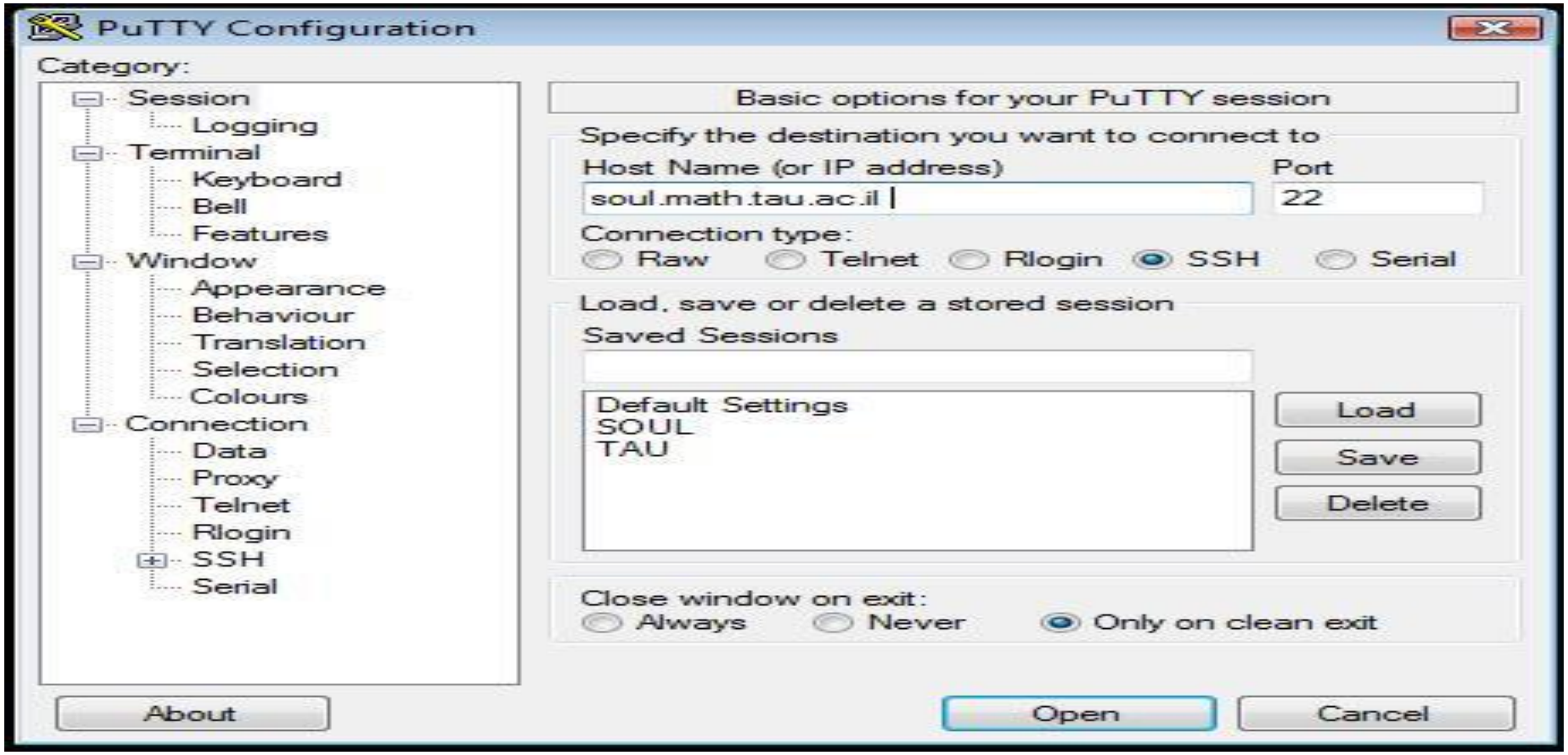
1. **"aries"** - UltraSPARC T1 - Sun Fire T2000.
2. **"capricorn"** - UltraSPARC T2 Plus - Sun SPARC Enterprise T5140 Server .

One INTEL:

1. **"pc-shanir4"** - *i7* 965 Extreme Edition, *Nehalem micro architecture.*
2. *You* Core 2 Duo can be helpful too.

Hardware: remote access I

1. **Using SSH Client** such as **PuTTY.**
2. Connect to the host **"soul.math.tau.ac.il"**, or **"nova.math.tau.ac.il"**
3. There are more productive SSH Clients.



PuTTY Configuration

Category:

- Session
  - Logging
- Terminal
  - Keyboard
  - Bell
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
  - Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): soul.math.tau.ac.il

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
SOUL
TAU

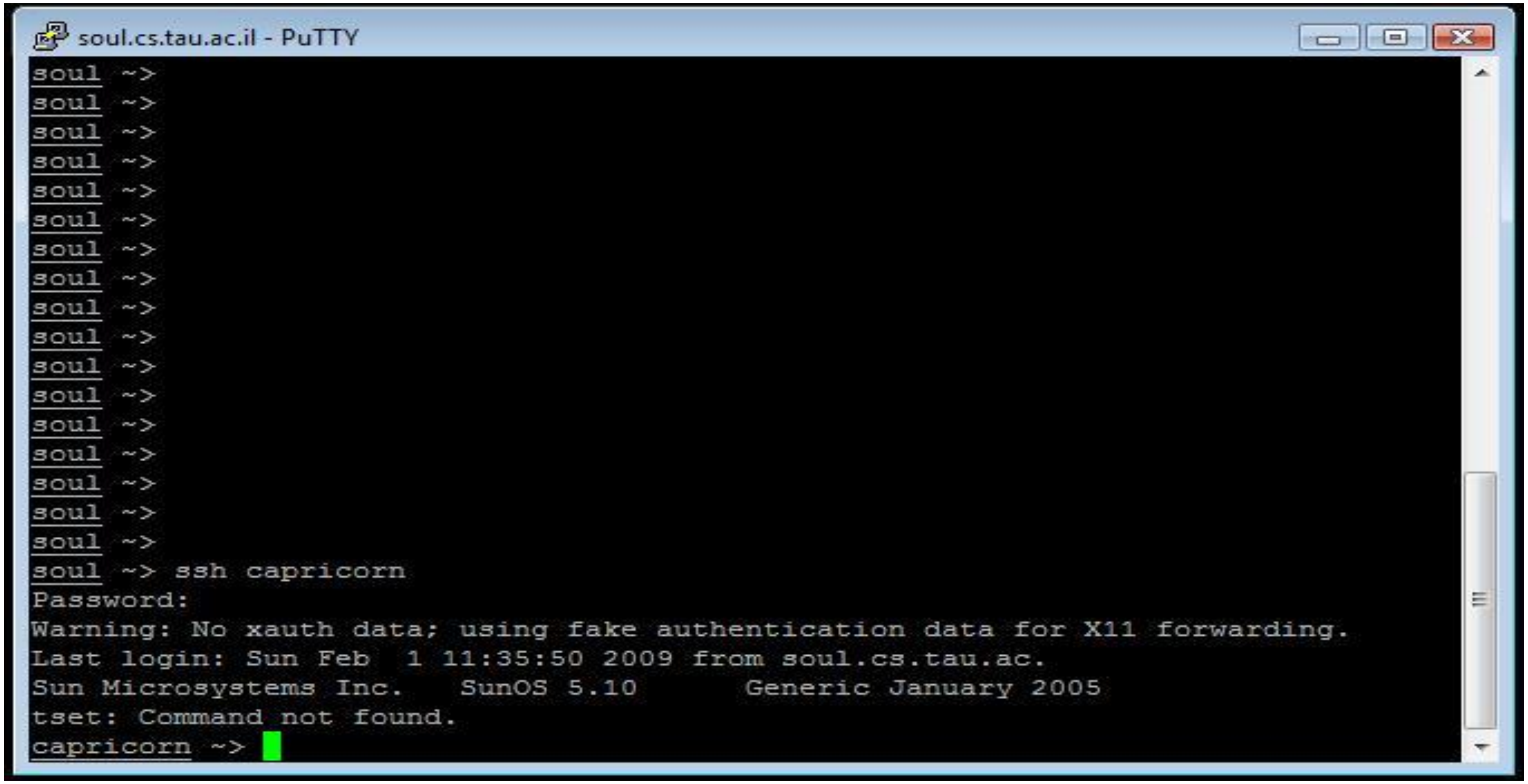
Load Save Delete

Close window on exit: Always Never [✓ Only on clean exit]

About Open Cancel

Hardware: remote access II

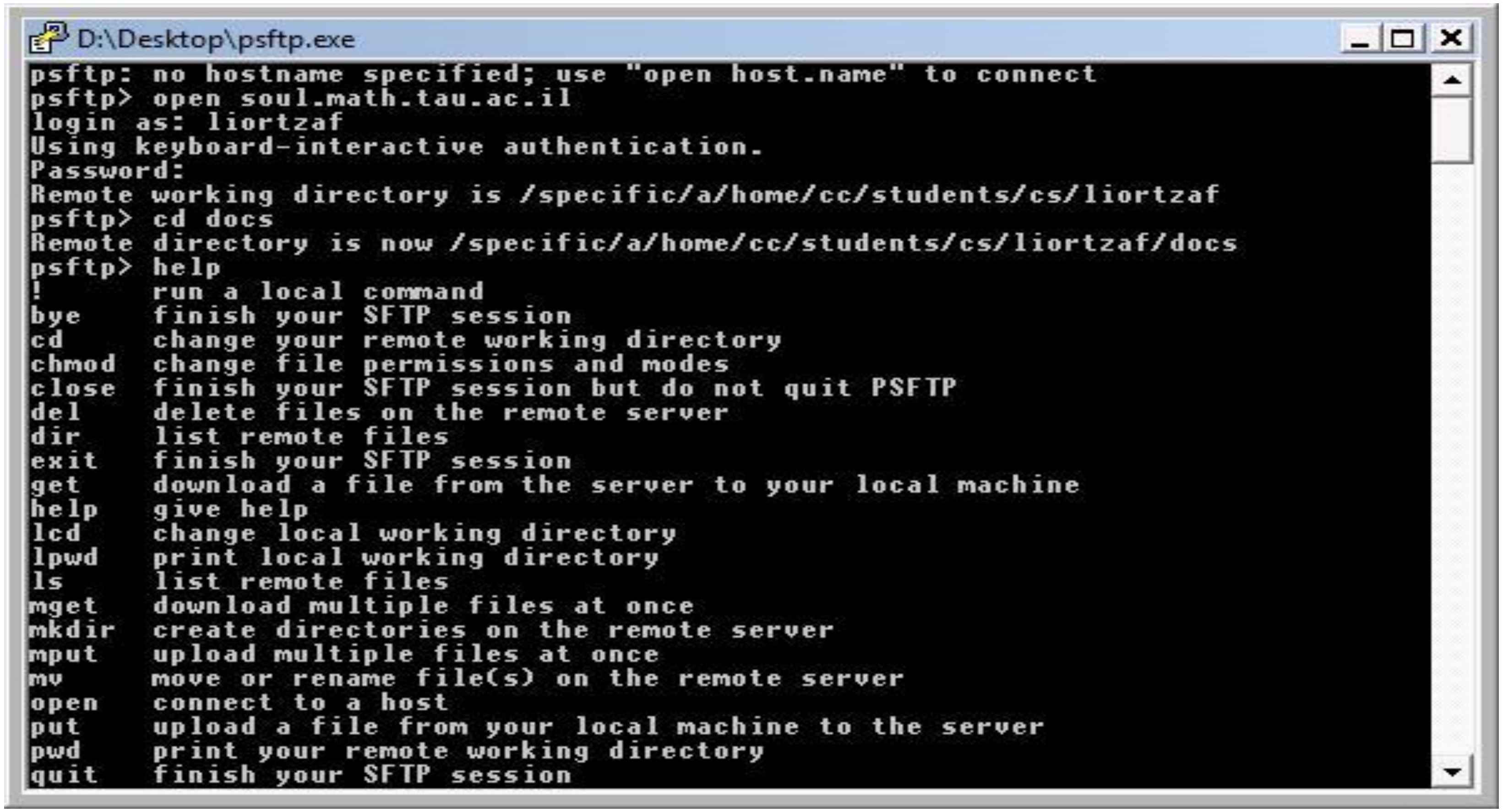
1. **ssh** to the machine
2. e.g. "ssh capricorn", "ssh aries", etc.



```
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~>
soul ~> ssh capricorn
Password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Sun Feb  1 11:35:50 2009 from soul.cs.tau.ac.
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
tset: Command not found.
capricorn ~>
```

Hardware: remote access III

1. Upload your project to the remote machine using secure FTP.



```
psftp: no hostname specified; use "open host.name" to connect
psftp> open soul.math.tau.ac.il
login as: liortzaf
Using keyboard-interactive authentication.
Password:
Remote working directory is /specific/a/home/cc/students/cs/liortzaf
psftp> cd docs
Remote directory is now /specific/a/home/cc/students/cs/liortzaf/docs
psftp> help
!      run a local command
bye    finish your SFTP session
cd     change your remote working directory
chmod  change file permissions and modes
close  finish your SFTP session but do not quit PSFTP
del    delete files on the remote server
dir    list remote files
exit   finish your SFTP session
get    download a file from the server to your local machine
help   give help
lcd    change local working directory
lpwd   print local working directory
ls     list remote files
mget   download multiple files at once
mkdir  create directories on the remote server
mput   upload multiple files at once
mv     move or rename file(s) on the remote server
open   connect to a host
put    upload a file from your local machine to the server
pwd    print your remote working directory
quit   finish your SFTP session
```

Hardware: *SPARC - Scalable Processor Architecture*

<u>UltraSPARC T1 - Sun Fire T2000 - 32 Threads in 2 Square Inches</u>

8 cores x 4 HT

16 KB primary instruction cache per core

8 KB primary data cache per core

On-chip level 2 cache

3 MB unified level 2 cache, 4 banks

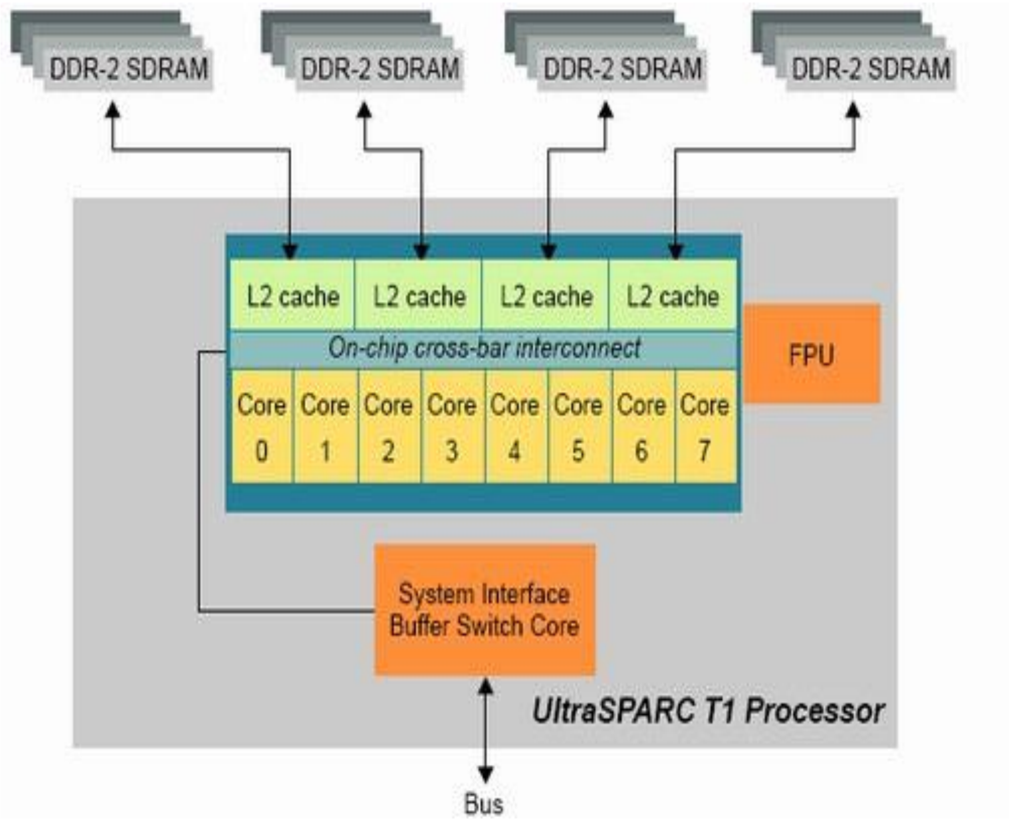
Single FPU

4 DIMMS per controller – 16 DIMMS total

3.1 GB/sec peak effective bandwidth

line size 64b

32GM Megabytes



Hardware: *SPARC - Scalable Processor Architecture*

UltraSPARC T2 Plus - Sun SPARC Enterprise T5140 Server

8 core 1.2-GHz x 8 HT

4 MB integrated L2

One floating point unit per core

two processors per system,

maximum 128 threads

Up to 32 FB-DIMM slots,

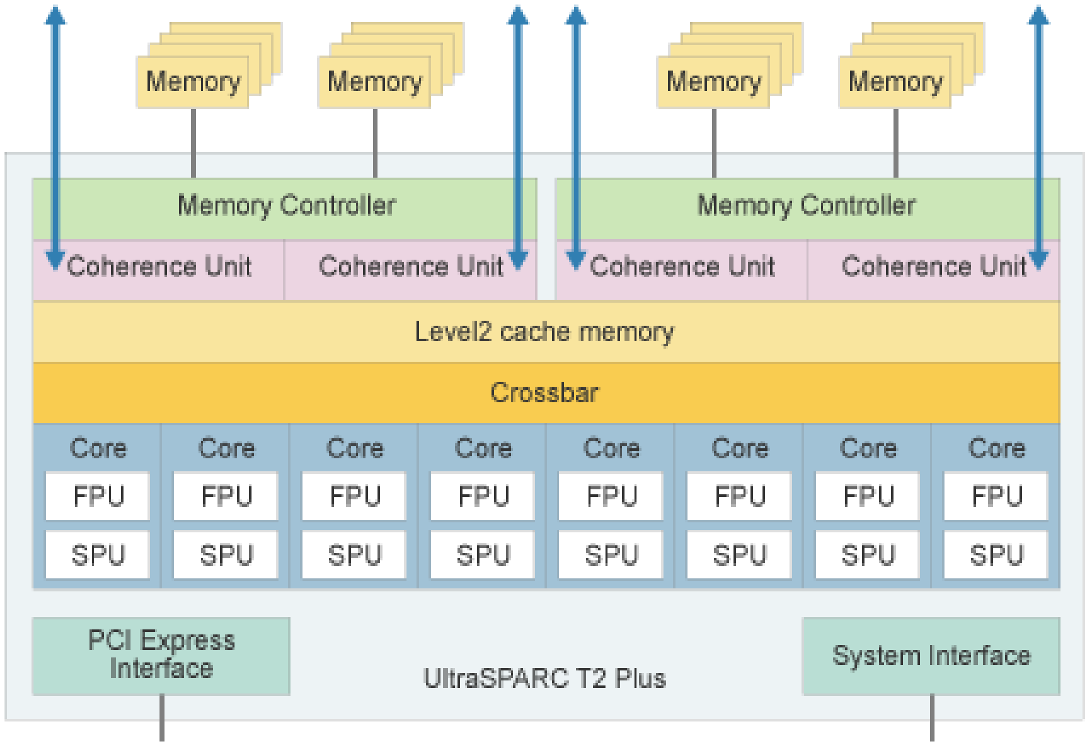
system maximum of 256 GB

60 GB/sec peak effective bandwidt

line size 64b

64GM Megabytes

UltraSPARC T2 Plus Processor Diagram



Hardware: *Nehalem micro architecture*

i7 920 Extreme Edition

4 core 1.2-GHz x 2 HT

256 KB L2 cache per core

One floating point unit per core

8 MB L3 (combined instruction and data)

32 KB L1 instruction cache per core

32 KB L1 data cache per core

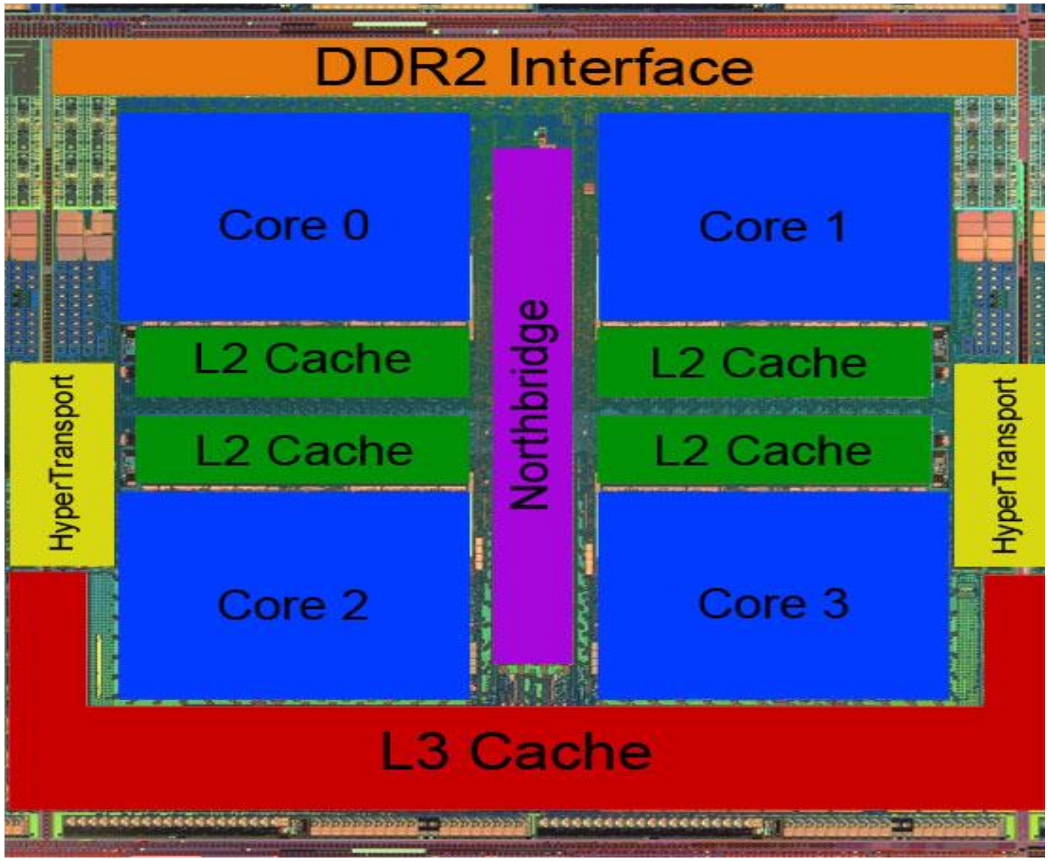
maximum 8 threads

3x DDR3-800/1066

25 GB/sec peak effective bandwidth

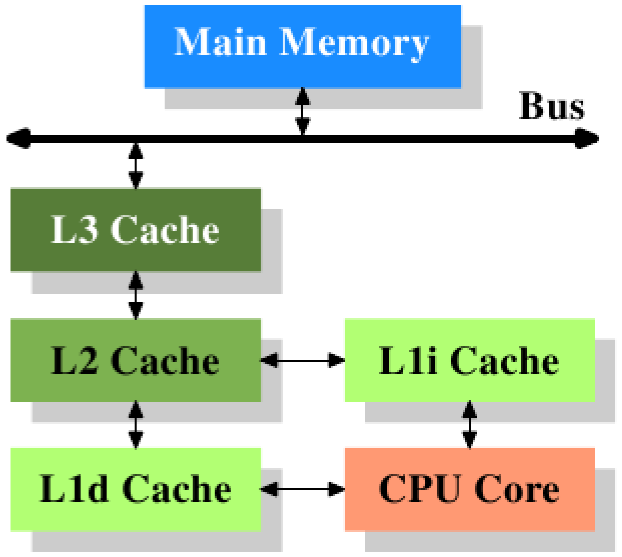
line size 64b

8GM Megabytes



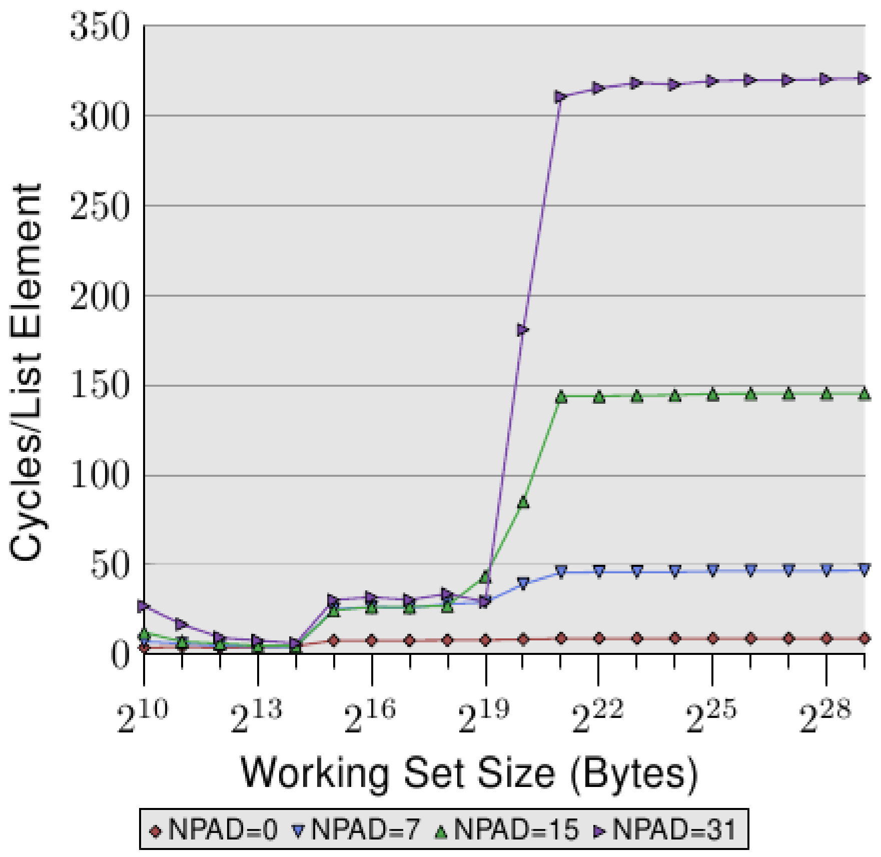
Hardware: *Hard Facts*

To Where	Cycles
Register	<= 1
L1d	~3
L2	~14
Main Memory	~240

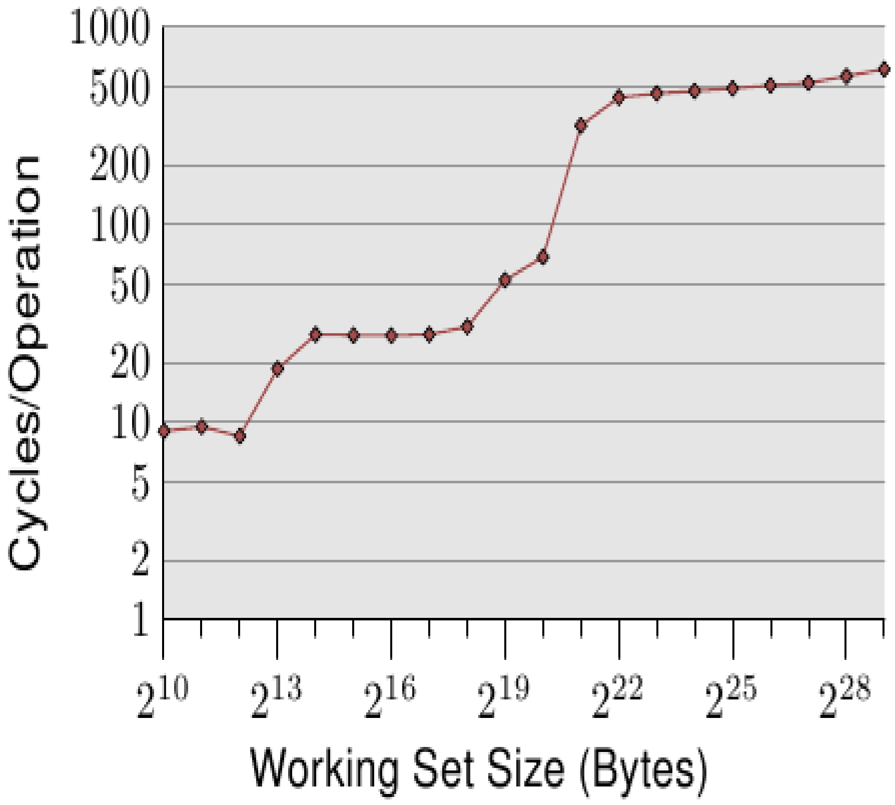


LOAD < STORE < STORE-Memory Barrier < CAS < CAS-Atomic-Markable-Reference

Hardware: *Hard Facts*



sequential Read for Several Sizes



Access Times for Random Writes

Software – our alternatives

JAVA:

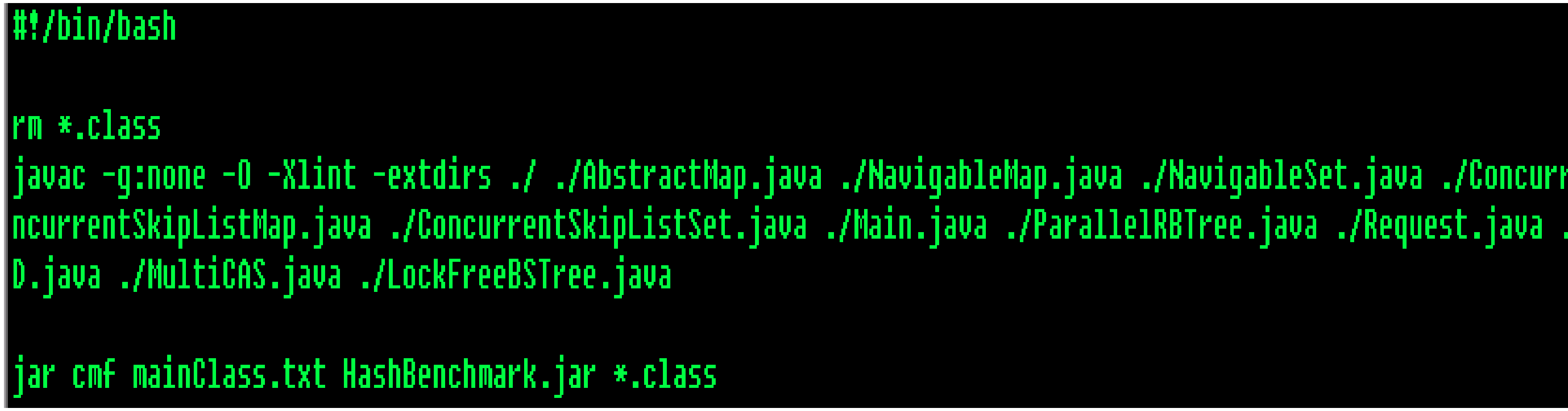
1. Java supports threads that shared memory
2. strong safety and security properties
3. ease of use for programmers
4. implementation flexibility for system designers
5. Weak on optimization,
6. **Hard to take advantage of the underlying hardware. (for example pre fetching, etc)**
7. **Very hard to create cache efficient algorithms. No control over memory alignment.**

C++:

1. defined as single threaded languages
2. no reference to thread
3. compilers are thread unaware
4. compilers reorder assignments to independent vars
5. does not preserve meaning of multi-threaded programs
6. **Can be optimized to take full advantage of the underlying hardware.**
7. **Full control over memory alignment.**

Software – Java - Compilation

- **Command line**

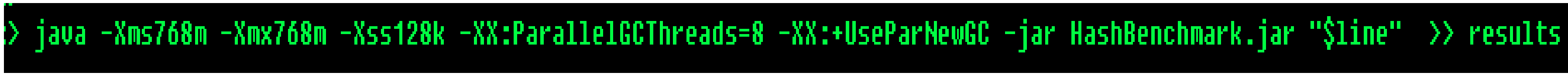


```
#!/bin/bash

rm *.class
javac -g:none -O -Xlint -extdirs ./ ./AbstractMap.java ./NavigableMap.java ./NavigableSet.java ./Concurr
ncurrentSkipListMap.java ./ConcurrentSkipListSet.java ./Main.java ./ParallelRBTree.java ./Request.java .
D.java ./MultiCAS.java ./LockFreeBSTree.java

jar cmf mainClass.txt HashBenchmark.jar *.class
```

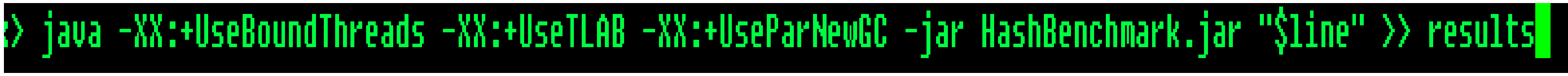
-g:none	- do not generate any debugging information
-Xlint: unclecked	- set warning level
-J*option*	- Pass *option* to the java launcher

Software – Java - java launcher

- **Command line**



```
> java -Xms768m -Xmx768m -Xss128k -XX:ParallelGCThreads=8 -XX:+UseParNewGC -jar HashBenchmark.jar "$line"  >> results
```



```
> java -XX:+UseBoundThreads -XX:+UseTLAB -XX:+UseParNewGC -jar HashBenchmark.jar "$line" >> results
```

- -Xms initial java heap size
- -Xmx maximum java heap size
- -Xmn the size of the heap for the *young generation*
- -Xss the stack size for each thread
- -XX:+UseParNewGCParallel copying [default]
- -XX:+UseParallelGCParallel scavenging
- -XincgcIncremental
- -XX:+UseConMarkSweepGCConcurrent mark and sweep

Software – Java – Code Optimization

```
double x = d * (lim / max) * sx;
double y = d * (lim / max) * sy;
```

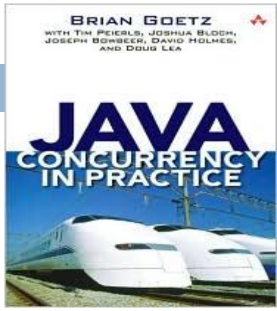
```
double depth = d * (lim / max); double x =
depth * sx;
double y = depth * sy;
```

```
for (int i = 0; i < x.length; i++)
    x[i] *= Math.PI * Math.cos(y);
```

```
double picosy = Math.PI * Math.cos(y);
for (int i = 0; i < x.length; i++)
  x[i] *= picosy;
```

- public static / private
- java -prof myClass
- long start = System.currentTimeMillis();
  // do operation to be timed here
  long time = System.currentTimeMillis() - start;

Software – Java – Memory Model

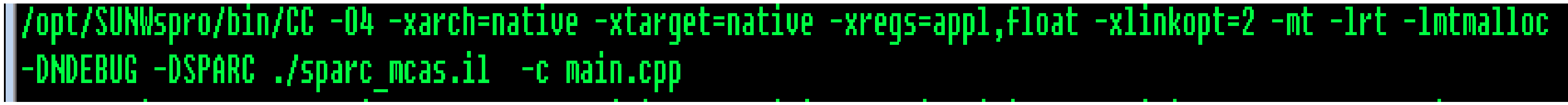

BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

Java Concurrency in Practice

1. Each action in a thread happens-before every action in that thread that comes later in the program order
2. An unlock on a monitor happens-before every subsequent lock on that same monitor
3. A write to a volatile field happens-before every subsequent read of that same volatile
4. A call to Thread.start() on a thread happens-before any actions in the started thread
5. All actions in a thread happen-before any other thread successfully returns from a Thread.join() on that thread

Software – C++ - Compilation

- **Command line**



```
/opt/SUNWspro/bin/CC -O4 -xarch=native -xtarget=native -xregs=appl,float -xlinkopt=2 -mt -lrt -lmtmalloc
-DNDEBUG -DSPARC ./sparc_mcas.il  -c main.cpp
```

/opt/SUNWspro/bin/CC	
-O4	
-xarch=native	Specify instruction set architecture.
-xregs=appl,float	Specifies the usage of registers for the generated code.
-xlinkopt=2	
-mt	Macro option that expands to -D_REENTRANT -lthread.
-lrt	
-lmtmalloc	
-DNDEBUG	
-DSPARC	
./sparc_mcas.il	

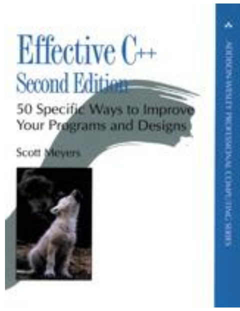
Software – C++ – Memory Model

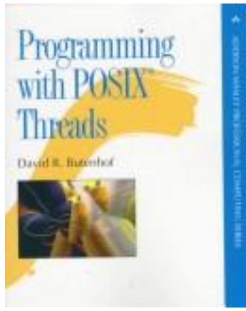
Avoiding Data Races

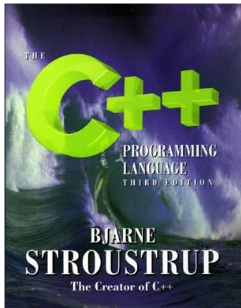
1. Atomic primitives that are safe for concurrent use
   - e.g. gcc _sync intrinsics (selected examples)
   - bool __sync_bool_compare_and_swap (*type* *ptr, *type* oldval *type* newval, ...)
   - *type* __sync_val_compar*type* __sync_add_and_fetch (*type* *ptr, *type* value, ...)
   - *type* __sync_sub_and_fetch (*type* *ptr, *type* value, ...)
   - *type* __sync_or_and_fetch (*type* *ptr, *type* value, ...)
   - *type* __sync_and_and_fetch (*type* *ptr, *type* value, ...)

2. **POSIX Threads**
   - pthread_mutex_lock
   - pthread_create
   - pthread_cond_init
   - Etc

3. **Compiler specific pragma**







Software – C++ - framework

```
class System {
public:
tatic tick_t read_cpu_ticks()
static tick_t currentTimeMillis()
    //Should add enumerate CPU, #CPU, bind, etc
};
```

Software – C++ - framework

```
class Memory {
public:
        static int get_cacheline_size()
        static void* byte_malloc(const size_t size)
        static void  byte_free(void* mem)
        static void* byte_aligned_malloc(const size_t size)
        static void* byte_aligned_malloc(const size_t size, const size_t alignment)
        static void  byte_aligned_free(void* mem)
        static void read_write_barrier()
        static void write_barrier()
        static void read_barrier()
        static _u32 compare_and_set(_u32 volatile* _a, _u32 _o, _u32 _n)
        static void* compare_and_set(void* volatile * _a, void* _o, void* _n)
        static _u64 compare_and_set(_u64 volatile* _a, _u64 _o, _u64 _n)
        static _u32 exchange_and_set(_u32 volatile* _a, _u32 _n)
        static void* exchange_and_set(void* volatile _a, void* _n)
};
```

Software – C++ - framework

1. class Integer
2. class AtomicInteger
3. class AtomicLong
4. class AtomicBoolean
5. template<typename V> class AtomicReference
6. template<typename V> class AtomicMarkableReference

Software **– C++ -** framework

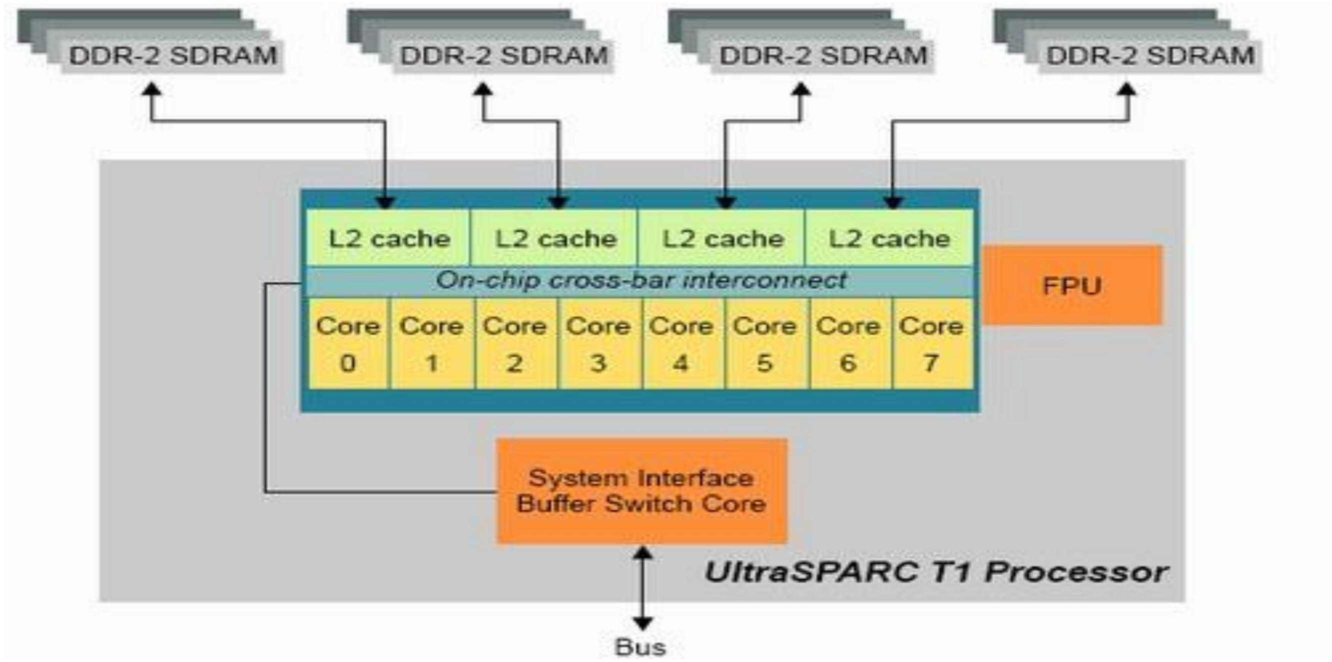
1. class Thread
   1. static void sleep(const long millis)
   2. static void sleep(const long millis, const int nanos)
   3. static void set_concurency_level(const int num_threads)
   4. int getPriority()
   5. void setPriority(const int newPriority)
   6. bool isAlive()
   7. void yield() {
   8. void setDaemon()
   9. void join()
   10. void start()
   11. virtual void run() = 0;

***Software* – C++ - framework**

1. class DummyLock
2. class TTASLock
3. class TASLock
4. class Random
   1. double nextUniform(void)
   2. double nextGaussian(double mean, double stddev)
   3. int nextInt(int lower,int upper)
   4. int nextInt(int upper)
   5. double nextDouble(double lower,double upper)
   6. double nextDouble(double upper)
   7. **static long long getSeed()**
   8. **static long long getRandom(long long seed, long long value)**
   9. **static long long getRandomScatter(long long seed, int value)**

Benchmarks - *What we want benchmark*

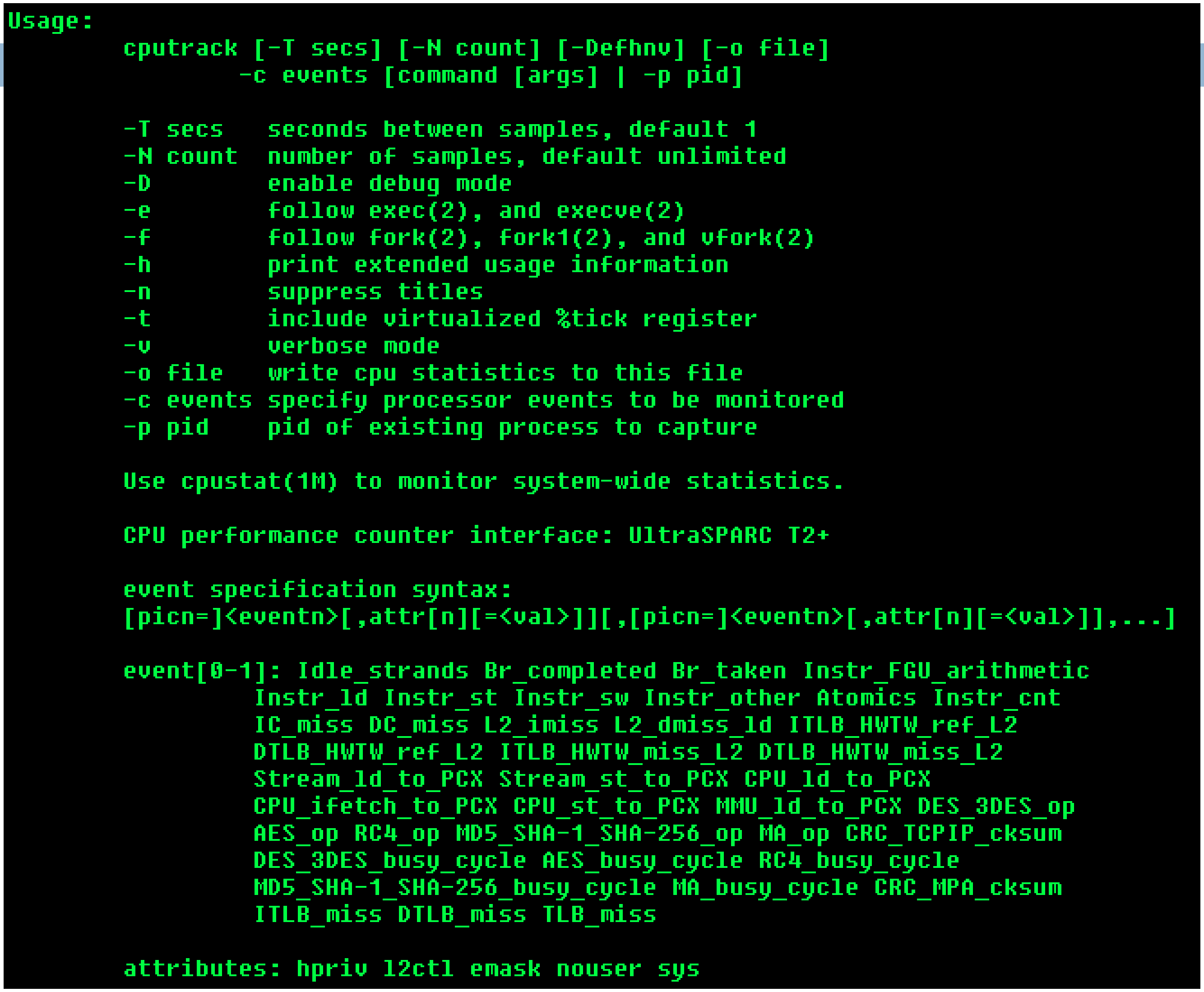
1. We want to quantify the contribution of our algorithm.
2. The issue is that we would like to know what is the contribution. e.g. CPU time, Memory Access time, contention, & total throughput.
3. Maybe we improved one thing and made other worse.



Benchmarks - What we can benchmark

1. Measuring the throughput of our algorithm can be tricky.
2. How to display the throughput, per thread/total.
3. Making sure all thread are working, the CPU ~99%.
4. Tools like **vmstat** and **mpstat** on Unix systems or **perfmon** on Windows systems can tell you just how "hot" the processors are running.
5. If the CPUs are asymmetrically utilized (some CPUs are running hot but others are not) your first goal should be to find increased parallelism in your program. Asymmetric utilization indicates that most of the computation is going on in a small set of threads, and your application will not be able to take advantage of additional processors.
6. If the CPUs are not fully utilized, you need to figure out why:
   1. **Insufficient load.**
   2. **I/O-bound.**
   3. **Externally bound.**
   4. **Lock contention.**

Benchmarks - CPUTRACK



```
Usage:
        cputrack [-T secs] [-N count] [-Defhnv] [-o file]
                -c events [command [args] | -p pid]

        -T secs   seconds between samples, default 1
        -N count  number of samples, default unlimited
        -D        enable debug mode
        -e        follow exec(2), and execve(2)
        -f        follow fork(2), fork1(2), and vfork(2)
        -h        print extended usage information
        -n        suppress titles
        -t        include virtualized %tick register
        -v        verbose mode
        -o file   write cpu statistics to this file
        -c events specify processor events to be monitored
        -p pid    pid of existing process to capture

        Use cpustat(1M) to monitor system-wide statistics.

        CPU performance counter interface: UltraSPARC T2+

        event specification syntax:
        [picn=]<eventn>[,attr[n][=<val>]][,[picn=]<eventn>[,attr[n][=<val>]],...]

        event[0-1]: Idle_strands Br_completed Br_taken Instr_FGU_arithmetic
                 Instr_ld Instr_st Instr_sw Instr_other Atomics Instr_cnt
                 IC_miss DC_miss L2_imiss L2_dmiss_ld ITLB_HWTW_ref_L2
                 DTLB_HWTW_ref_L2 ITLB_HWTW_miss_L2 DTLB_HWTW_miss_L2
                 Stream_ld_to_PCX Stream_st_to_PCX CPU_ld_to_PCX
                 CPU_ifetch_to_PCX CPU_st_to_PCX MMU_ld_to_PCX DES_3DES_op
                 AES_op RC4_op MD5_SHA-1_SHA-256_op MA_op CRC_TCPIP_cksum
                 DES_3DES_busy_cycle AES_busy_cycle RC4_busy_cycle
                 MD5_SHA-1_SHA-256_busy_cycle MA_busy_cycle CRC_MPA_cksum
                 ITLB_miss DTLB_miss TLB_miss

        attributes: hpriv l2ctl emask nouser sys
```

Benchmarks - cputrack – example I

- DC_miss might become a L2 cache miss too, so you should measure L2_dmiss_ld (cost 100 cycles).
- so just L1 cahce miss = DC_miss - L2_dmiss_ld

- Total cpu stall in cycles is :
- **(DC_miss - L2_dmiss_ld)* 20 + L2_dmiss_ld*100**

Benchmarks – cputrack – example II

- cputrack -T 9999 -e -f -c Instr_cnt main.o >> results
- cputrack -T 9999 -e -f -c Instr_cnt java -Xms768m -Xmx768m -Xss128k -XX:ParallelGCThreads=8 -XX:+UseParNewGC -jar HashBenchmark.jar "$line" >> results
- capricorn ~/docs/hopscotch> cputrack -T 9999 -c DTLB_miss a.out

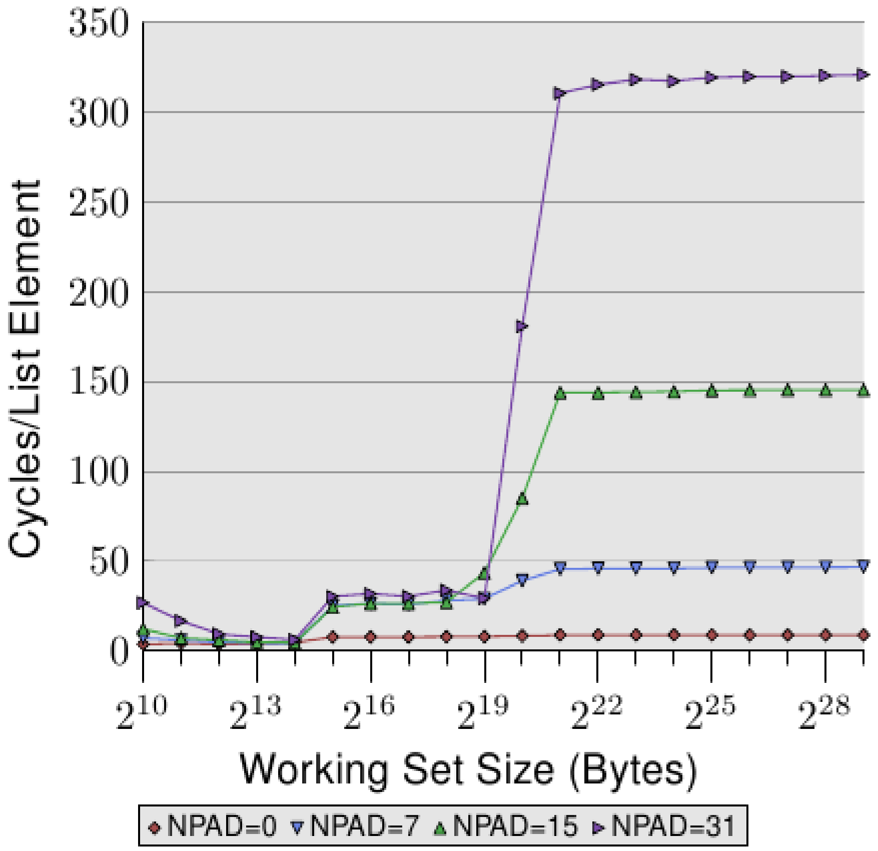
```
  time lwp      event      pic0
 0.034   1       exit      222
```

Benchmarks – cputrack – example III

```
 0.024  11391   1      exec    386003
 0.024  11391   1      exec                      # 'java -Xms768m
-Xmx768m -Xss128k -XX:ParallelGCThreads=8 -XX:+UseParNewGC -jar H'
 0.152  11391   2 lwp_create        0
 0.172  11391   3 lwp_create        0
 0.192  11391   4 lwp_create        0
 0.212  11391   5 lwp_create        0
 0.232  11391   6 lwp_create        0
……
 0.689  11391  16 lwp_create        0
 0.711  11391  17 lwp_create        0
 0.735  11391  18 lwp_create        0
 6.515  11391  19 lwp_create        0
 10.519  11391  19   lwp_exit 1364344825
622  15.578  11391  18   lwp_exit    201903
 15.588  11391  10   lwp_exit     86307
 15.594  11391   1       exit 3642863991
```

Benchmarks – *the data can be crucial*

- Work set size
- Sequential random
- Uniform Gaussian
- GC and the data
- Sequence of operations
- Operation per thread.



sequential Read for Several Sizes

Thanks !