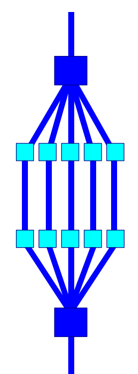


Application Performance Tuning
(An Overview)

Dr Simon See

High-Performance and Technical Computing

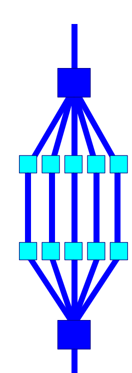
Asia Pacific Science and Technology Center
Sun Microsystems Inc

Java Technology Workshop
September 25th, 2003

Material is developed by Ruud Van Pas

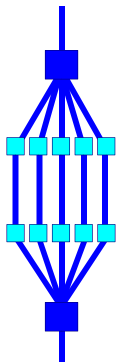
 Copyright©2003 Sun Microsystems

Outline



- ❑ *Introduction*
- ❑ *The Memory Hierarchy*
- ❑ *Single Processor Performance*
  - *The Sun Compilers*
  - *The Sun Performance Analyzer*
  - *Serial Optimization Techniques*
- ❑ *Parallelization*
  - *Introduction Parallelization*
  - *The SunFire Server Architecture*
  - *Automatic Parallelization by the Sun Compilers*
  - *Explicit Parallelization with OpenMP*

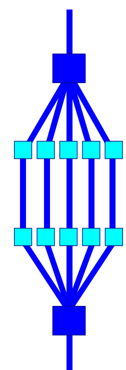
Copyright©2003 Sun Microsystems



Introduction

Copyright©2003 Sun Microsystems

Terminology/1



Mflop/s

- *Mflop/s = Million Floating Point operations/second*
- *Popular metric for performance*
- *Calculate by counting flops and divide by execution time*
- *Requires that one knows how many flops are performed*

Example:

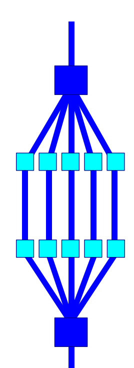
```
for (i=0; i < 1000000; i++)
   a[i] += 2 * b[i];
```

*Floating point operations : 2*1000000 = 2000000*
Execution time : 4 seconds

☞ *Performance = 1.0E-06*(2000000)/4 = 0.5 Mflop/s*

Copyright©2003 Sun Microsystems

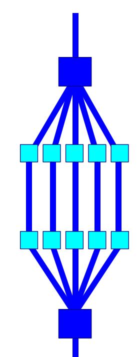
Terminology/2

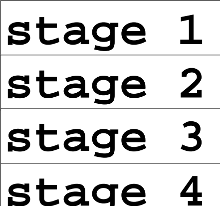


- ❑ *Cycles*
  - *Processor cycles (in nanoseconds)*
  - *Typically gives us a best-case scenario*
- ❑ *Pragma*
  - *#pragma "information to the C compiler"*
  - *!$directive "information to the Fortran compiler"*
- ❑ *Memory footprint*
  - *How much memory is used by the application ?*

Copyright©2003 Sun Microsystems

Pipelining





Let's assume we have an operation that takes 4 stages per iteration

inputs

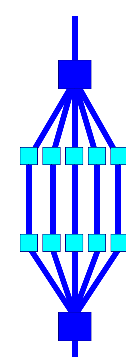
Time

stage 1
stage 2
stage 3
stage 4

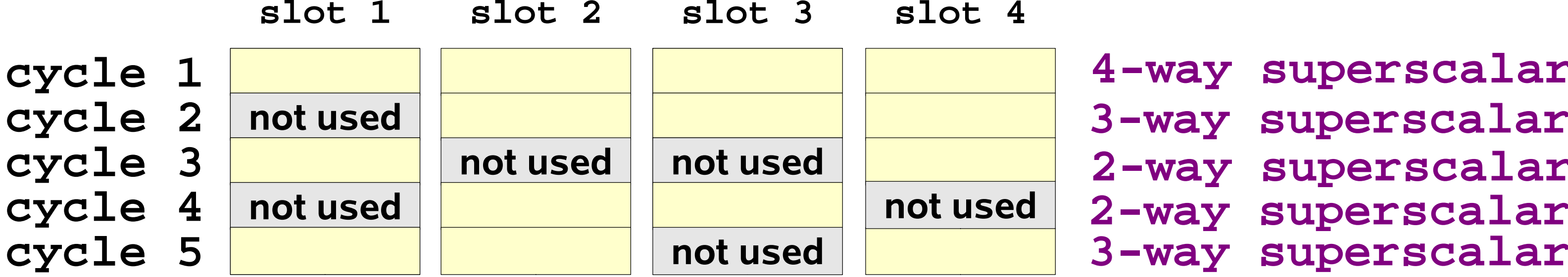
Latency of this operation ("start-up" time)

Copyright©2003 Sun Microsystems

Superscalar



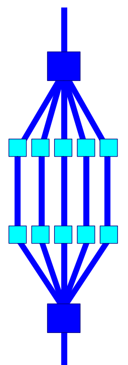
- ***N-way superscalar:***
  - ***Execute N instructions at the same time***
- ***This is also called Instruction Level Parallelism (ILP)***



	slot 1	slot 2	slot 3	slot 4	
cycle 1					4-way superscalar
cycle 2	not used				3-way superscalar
cycle 3		not used	not used		2-way superscalar
cycle 4	not used			not used	2-way superscalar
cycle 5			not used		3-way superscalar

- ***The hardware has to support this, but it is up to the software to take advantage of it***
- ***Often there are restrictions which instructions can be "bundled"***
- ***These are documented in the Architecture Reference Manual for the microprocessor***

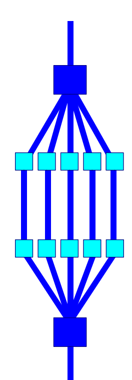
Copyright©2003 Sun Microsystems



The Memory Hierarchy

 Copyright©2003 Sun Microsystems

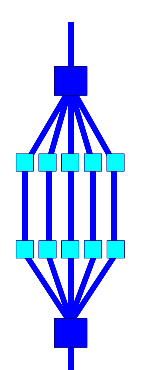
About Memory



- *Memory plays a crucial role in performance*
- *Not accessing memory in the right way will degrade performance on all computer systems*
- *The extent of the degradation depends on the system*
- *Knowing more about some of the relevant memory characteristics will help you to write code such that the problem will be non-existent, or at least minimal*

 Copyright©2003 Sun Microsystems

More about Memory



1x

200x

slow

CPU

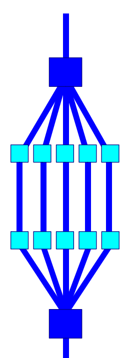
Memory

~1 ns

~200 ns

Copyright©2003 Sun Microsystems

About Caches and Memory



1x | 10- 20x | 200x

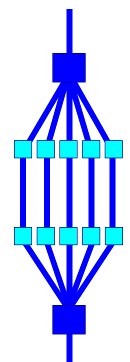
fastest | faster | slow

CPU | L1 cache | L2 cache | Memory

~1 ns | ~10-20 ns | ~200 ns

Copyright©2003 Sun Microsystems

The Memory Hierarchy



Size

Speed

< 1 ns

~10-20 ns

~200 ns

Microprocessor

L1-caches

I-cache

TLB

D-cache

CPU

Registers

L2 Unified Cache

Physical Memory (RAM)

page

element

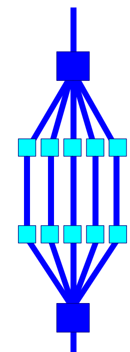
cache line

cache line

Memory Optimization:
Keep frequently used data close to the processor

Copyright©2003 Sun Microsystems

The TLB Cache ('Address Cache')



Instruction Pipeline

Load Instruction

1. TLB Lookup

TLB cache

page address

page address

page

Physical Memory

page

2. If address not in TLB:

Load Instruction Stalls

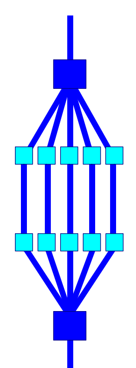
Set up new TLB entry

3. Load Instruction Resumes

Note that Solaris on SPARC uses a TSB in memory that acts as a buffer for the TLB

 Copyright©2003 Sun Microsystems

Performance Is Not Uniform



The length of a plateau is related to the size of that memory component

The amount of the drop is related to the latency (or bandwidth) of that memory component

Performance

Tuning "area"

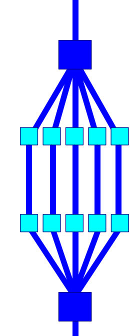
64 | 64 KB | 8 MB

Reg. | L1 | L2 | Main Memory | Virtual memory

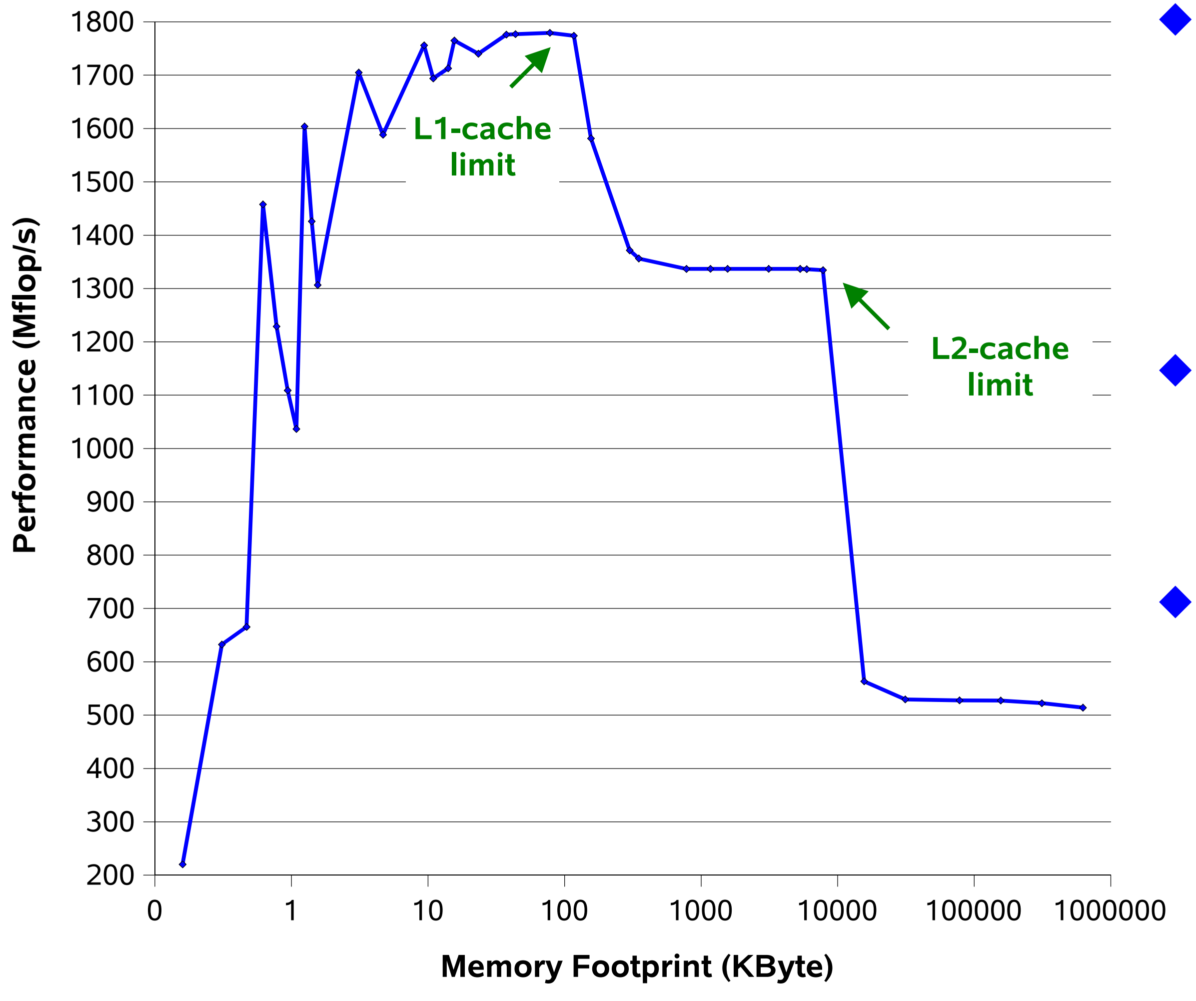
Memory Footprint

Copyright©2003 Sun Microsystems

Example - 13th deg. polynomial



```
for (i=0; i<vlen; i++)
   p[i] = c[0] + q[i]*(c[1] + q[i]*(c[2] + q[i]*(c[3] + ....
```



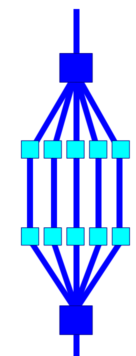
- ***This operation is <u>CPU bound</u> i.e. there are many more floating point operations than memory references***
- ***The system realizes over 98% of the absolute peak performance !***
- ***Note the start-up effect and the performance drop for larger problems***

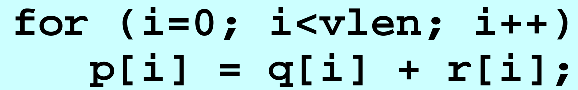
SF6800 - USIII Cu@900MHz

L1 cache	**:**	**64 KByte**
L2 cache	**:**	**8 MByte**
Peak speed	**:**	**1800 Mflop/s**

 Copyright©2003 Sun Microsystems

Example - Vector addition





Performance (Mflop/s)

300
275
250
225
200
175
150
125
100
75
50
25
0

L1-cache limit

L2-cache limit

0 1 10 100 1000 10000 100000 1000000

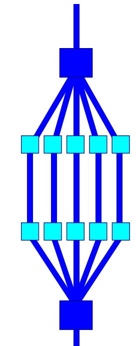
Memory Footprint (KByte)

- ***This operation is <u>memory bound</u> i.e. there are more memory references than floating point operations***
- ***The system realizes close to the theoretical peak performance for <u>this</u> operation (=16% of absolute peak)***
- ***Note the start-up effect and the performance drop for larger problems***

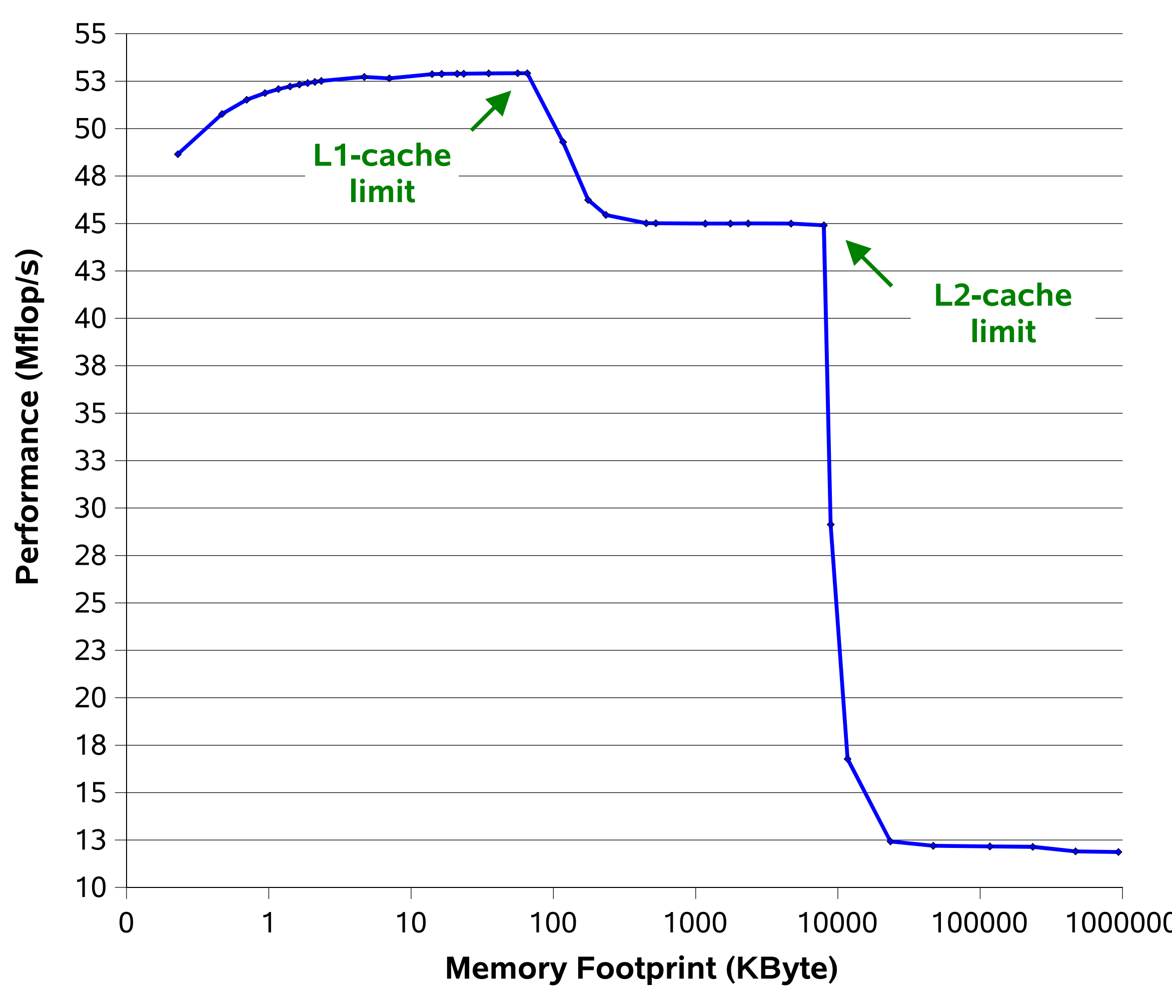
SF6800 - USIII Cu@900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

Copyright©2003 Sun Microsystems

Example - Vector divide



```
for (i=0; i<vlen; i++)
    p[i] = q[i] / r[i];
```

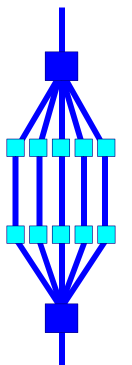


- ***This operation is <u>latency bound</u> i.e. the cost of the instruction outweighs the cost of fetching data (if the data is close enough to the processor)***
- ***The division is an example of a non-pipelined, long latency operation***
- ***Can be overlapped with other floating point operations***

SF6800 - USIII Cu@900MHz

L1 cache	**:**	**64 KByte**
L2 cache	**:**	**8 MByte**
Peak speed	**:**	**1800 Mflop/s**

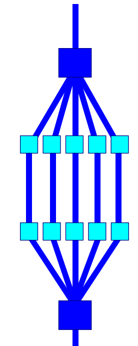
Copyright©2003 Sun Microsystems



Data Prefetch

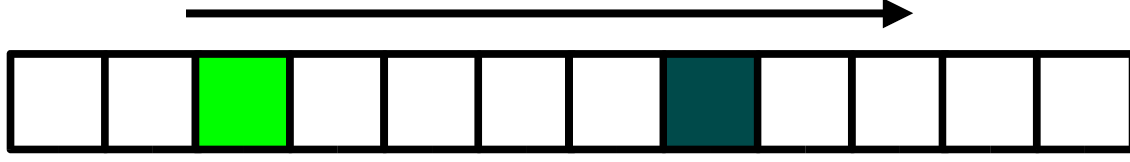
Copyright©2003 Sun Microsystems

Hiding Memory Latency

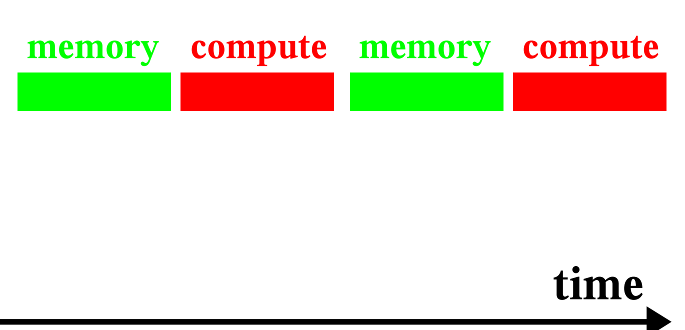


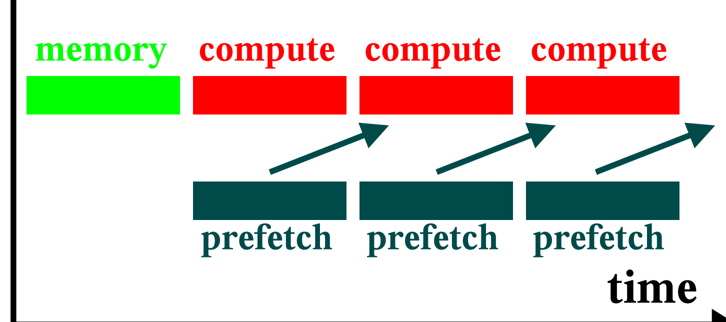
- *The memory access pattern may be predictable:*

  Example: summation of elements



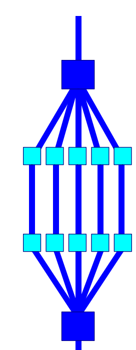
- *With prefetch, one fetches memory before it is needed*
- *This is called a "latency hiding technique"*





Copyright©2003 Sun Microsystems

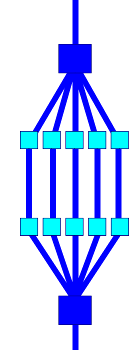
Prefetch on US-III Cu



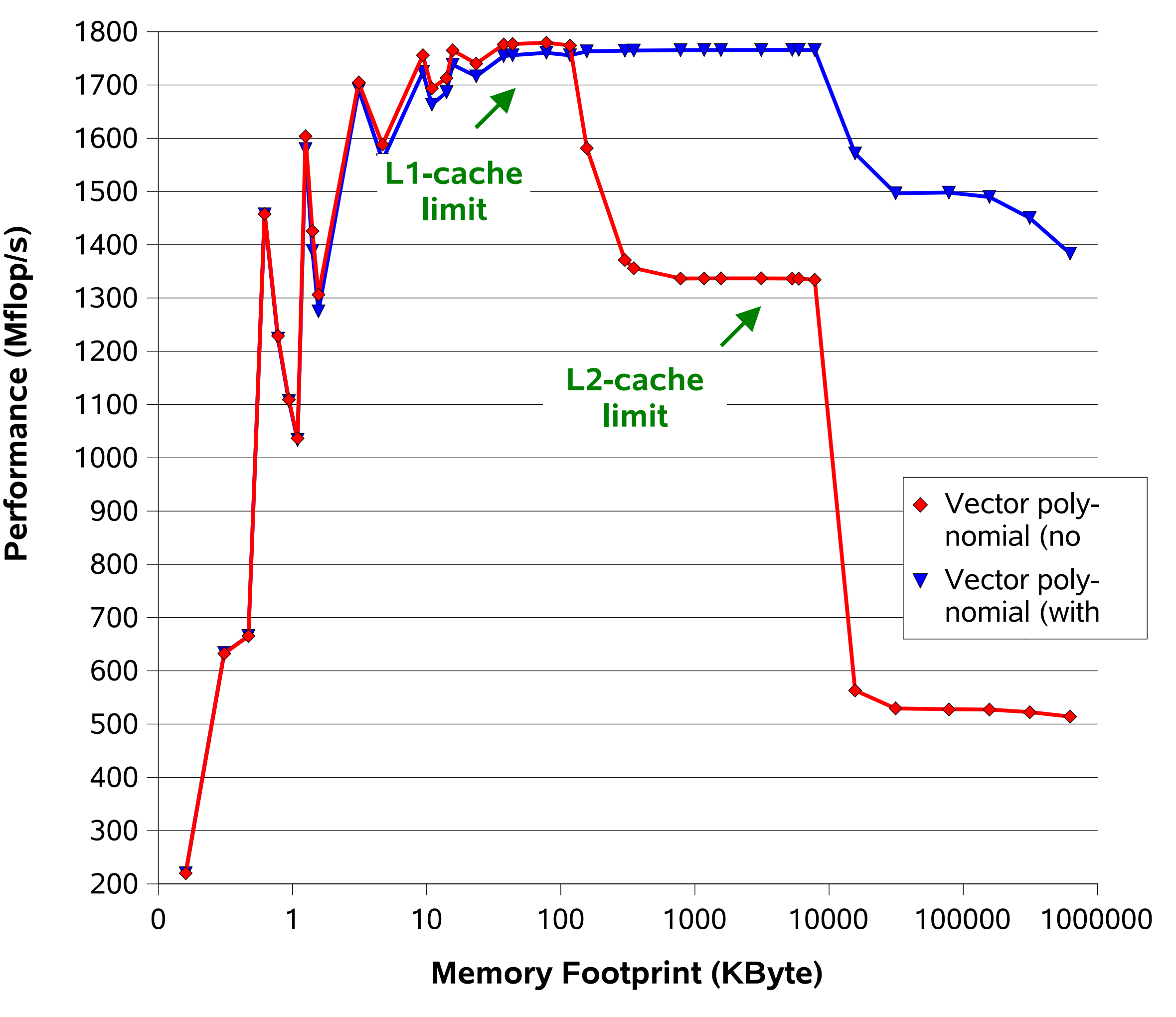
- ❑ *The Sun compilers support software prefetch:*
  - *Automatic - Compiler does a best effort*
    - ✔ *Use -xprefetch=yes and (optionally)*
    - ✔ *-xprefetch_level=n   (n=1, 2 or 3)*
  - *Explicit - User tells the compiler what to prefetch*
    - ✔ *Through function calls (C and C++) and directives (Fortran)*
  - *Combination of both*
- ❑ *Need UltraSPARC-III Cu for this*
- ❑ *If data is already 'close by', prefetch may slow down the application*

Copyright©2003 Sun Microsystems

Prefetch - 13th deg. polynomial



```
for (i=0; i<vlen; i++)
   p[i] = c[0] + q[i]*(c[1] + q[i]*(c[2] + q[i]*(c[3] + ....
```

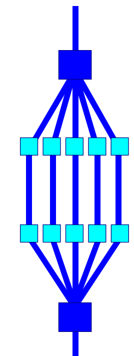


- ***Re-compiled with automatic prefetch enabled***
- ***Performance for L1 resident problem sizes is the same***
- ***For larger problem sizes, automatic prefetch is a big win !***

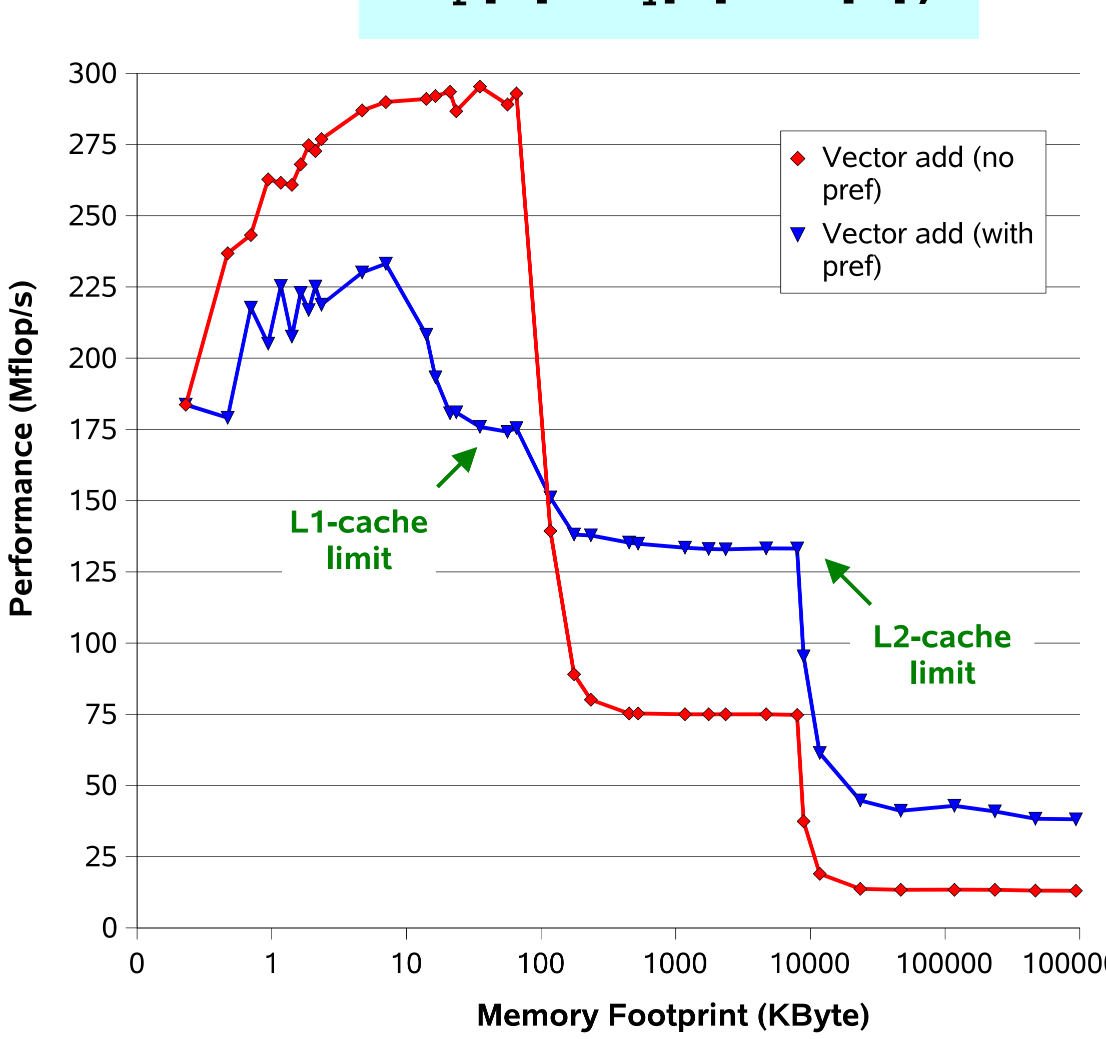
SF6800 - USIII Cu@900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

Copyright©2003 Sun Microsystems

Prefetch - Vector addition



```
for (i=0; i<vlen; i++)
   p[i] = q[i] + r[i];
```



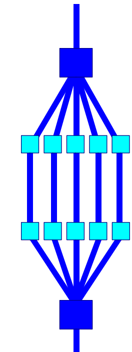
- ***Re-compiled with automatic prefetch enabled***
- ***Performance for L1 resident problem sizes is less if prefetch is used***
- ***For larger problem sizes, automatic prefetch gives a significant performance improvement***

SF6800 - USIII Cu@900MHz

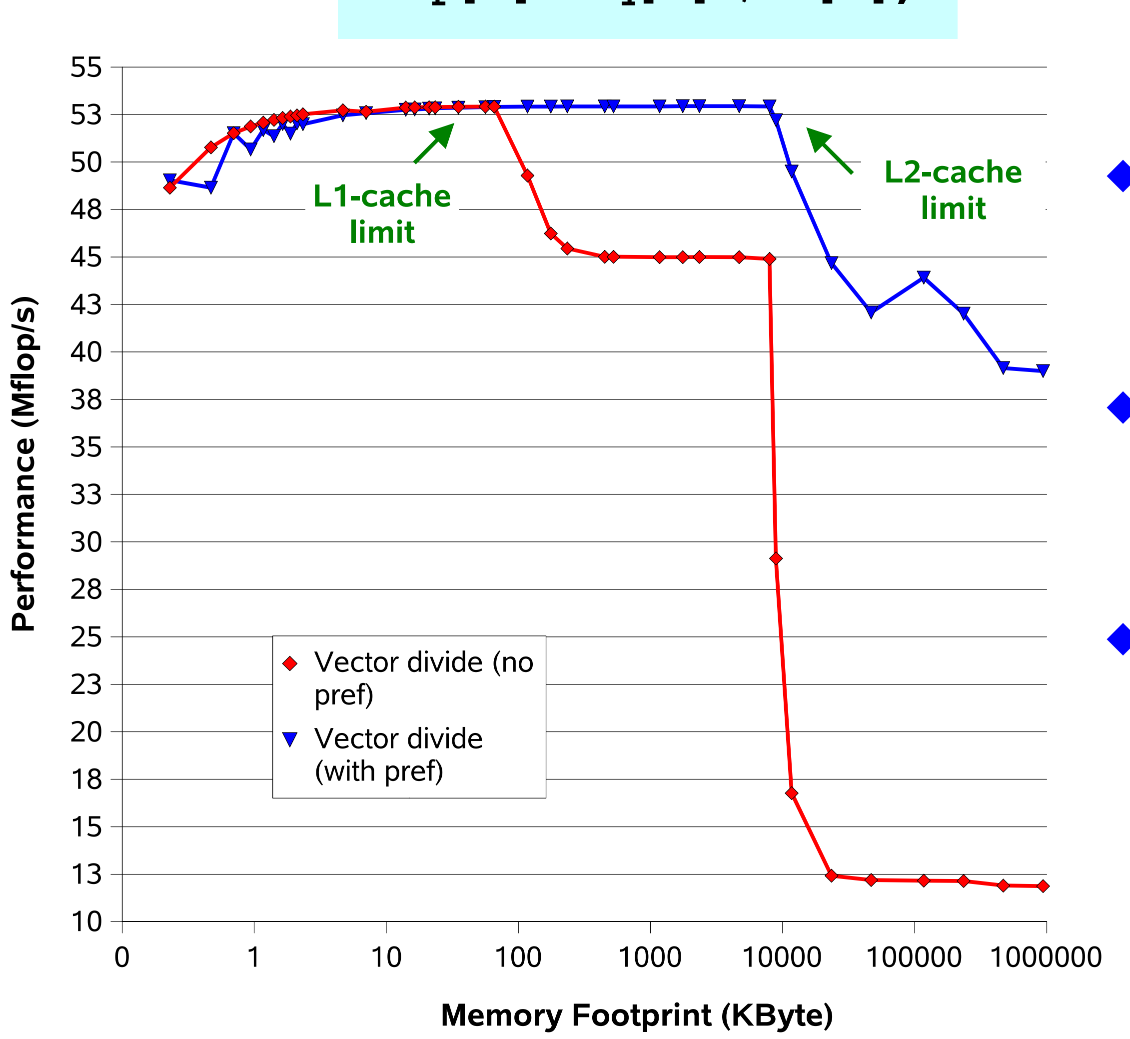
L1 cache	**:**	**64 KByte**
L2 cache	**:**	**8 MByte**
Peak speed	**:**	**1800 Mflop/s**

Copyright©2003 Sun Microsystems

Prefetch - Vector divide



```
for (i=0; i<vlen; i++)
   p[i] = q[i] / r[i];
```

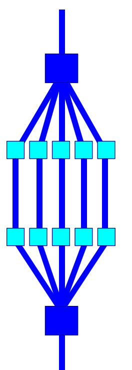


- *Re-compiled with automatic prefetch enabled*
- *Performance for L1 resident problem sizes is the same*
- *For L2 resident problem sizes, the improvement is noticeable*
- *For larger problem sizes, automatic prefetch gives a dramatic performance improvement !*

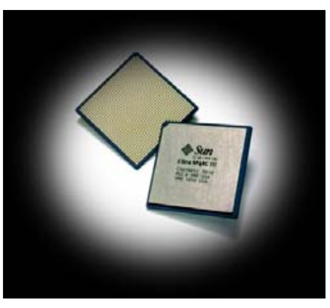
SF6800 - USIII Cu@900MHz

L1 cache	**:**	**64 KByte**
L2 cache	**:**	**8 MByte**
Peak speed	**:**	**1800 Mflop/s**

Copyright©2003 Sun Microsystems

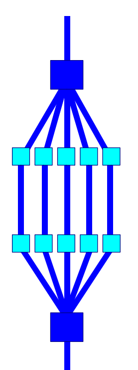


The UltraSPARC-III Cu Microprocessor



Copyright©2003 Sun Microsystems

US-III Cu Functional Units



The Chip

Instruction Issue Unit (IIU)

4 instructions

Floating Point Unit (FPU)

Integer Execution Unit (IEU)

Data Cache Unit (DCU)

L2 cache (SRAM)

Memory (SDRAM)

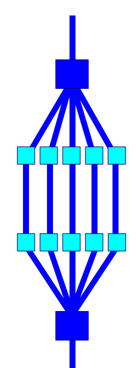
External Memory Unit (EMU)

System Interface Unit (SIU)

Interconnect

Copyright©2003 Sun Microsystems

US-III Cu Block Diagram



The Chip

Instruction Issue Unit (IIU)
I-TLB | I-Cache | B-Pred | I-Queue

4 instructions

Floating Point Unit (FPU)
FP Reg. File | FP mul/div/sqrt/VIS | FP add/subtract/VIS

Integer Execution Unit (IEU)
W&A RF | Dependency/trap logic | ALU 0 | ALU 1 | Ld/St/Special pipe

Data Cache Unit (DCU)
Data TLB | L1 Cache | Store Queue | Write Cache | Pref Cache

External Memory Unit (EMU)
Memory Controller | L2 Cache Controller | L2 Cache Tags

System Interface Unit (SIU)
Snoop pipe Controller | Data Switch Controller

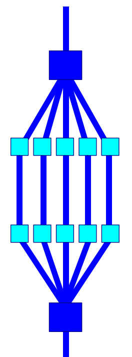
L2 cache (SRAM)

Memory (SDRAM)

Interconnect

Copyright©2003 Sun Microsystems

The Pipeline



14-stage pipeline

Integer

R	E	C	M	W	X

Instruction

A	P	F	B	I	J

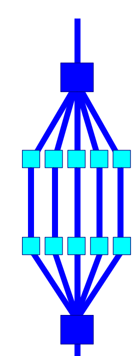
T	D

RF	F1	F2	F3	F4

Floating Point

Copyright©2003 Sun Microsystems

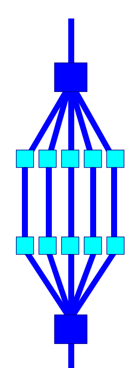
US-III/US-III Cu: 4-way superscalar



- Six execution pipelines:
  - A0&A1 - Two integer arithmetic and logical pipelines
  - BR - Branch pipeline
  - MS - Load/store pipeline (also handles special instructions)
  - FGM - FP multiply pipeline (also handles VIS instructions)
  - FGA - FP add pipeline (also handles VIS instructions)
- Execute up to 4 instructions in parallel (2 IEU + 2 FPU)
- Floating Point Peak Performance is 2*Speed in MHz:
  - 750 MHz: 1.5 Gflop/s
  - 900 MHz: 1.8 Gflop/s
  - 1050 MHz: 2.1 Gflop/s
  - 1200 MHz: 2.4 Gflop/s

Copyright©2003 Sun Microsystems

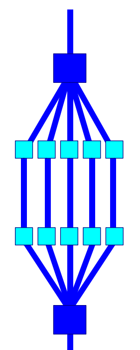
UltraSPARC Cache Evolution



Feature	US-II	US-III	US-III Cu
L1 cache (KB)	16 (1-way)	64 (4-way)	64 (4-way)
I-cache (KB)	16 (2-way)	32 (4-way)	32 (4-way)
E-cache (MB)	8 (1-way)	8 (1-way)	8 (2-way)
D-TLB entries	64 (fully ass.)	512 (2-way) 16 (fully ass.)	512 (2-way, t8_0) 16 (fully ass.) 512 (2-way, t8_1)
I-TLB entries	64 (fully ass.)	128 (2-way) 16 (fully ass.)	128 (2-way) 16 (fully ass.)
Prefetch cache (KB)	n.a.	n.a.	2 (4-way)
Write cache (KB)	n.a.	2 (4-way)	2 (4-way)

Copyright©2003 Sun Microsystems

US-III Cu Memory Hierarchy



Main Memory

L2
E-Cache

L1
D-Cache

L1
I-Cache

Prefetch
Cache

Write
Cache

D-TLBs

CPU

Registers

I-TLBs

UltraSPARC-III Cu

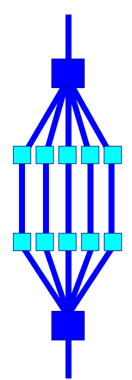
Load-Use Latencies (cycles)

From L1	2-3
From P$	3
L2	~16
Memory	System dep.

Prefetch Flavours:

read many	P$ and E$
read once	P$
write many	E$ (exclusive)
write once	E$ (shared)

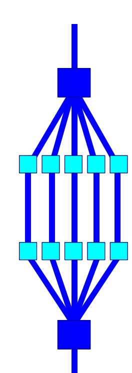
Copyright©2003 Sun Microsystems



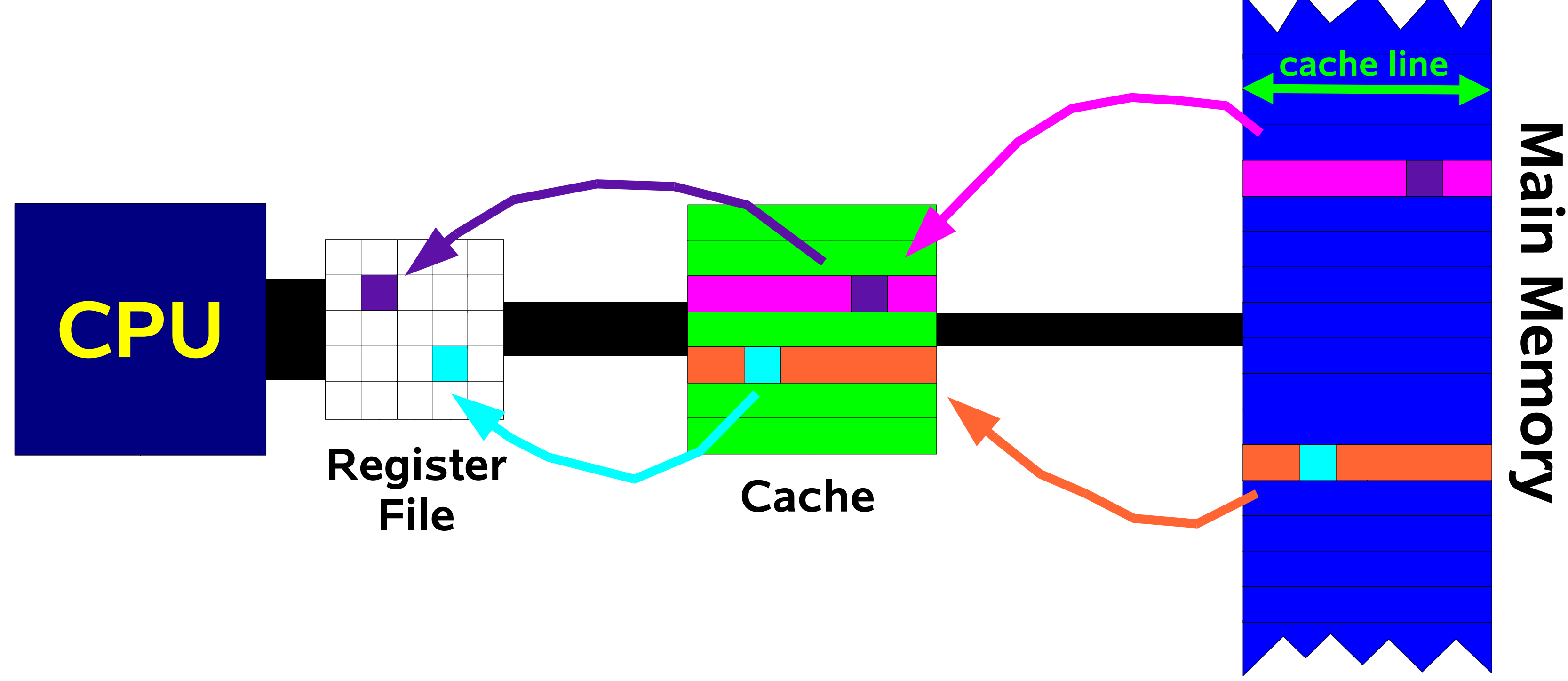
Memory Access (The Good, The Bad and the Ugly)

Copyright©2003 Sun Microsystems

Cache lines

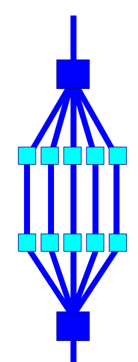


- *For good performance, it is crucial to use the cache(s) in the intended (=optimal) way*
- *Recall that the unit of transfer is a cache "line"*
- *A cache line is a linear structure i.e. it has a fixed length (in bytes) and a starting address in memory*

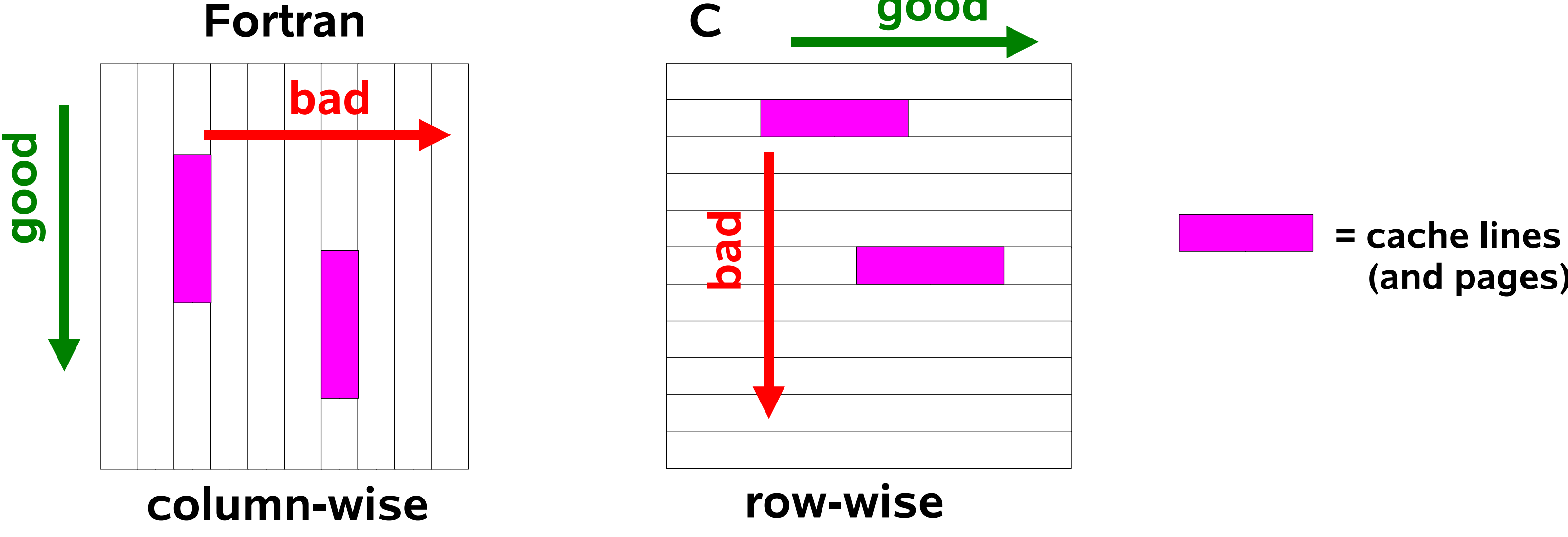


Copyright©2003 Sun Microsystems

Memory Access



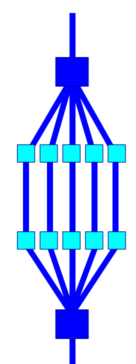
- Memory has a 1D, linear, structure
- Access to multi-dimensional arrays depends on the way data is stored
- This is language dependent:

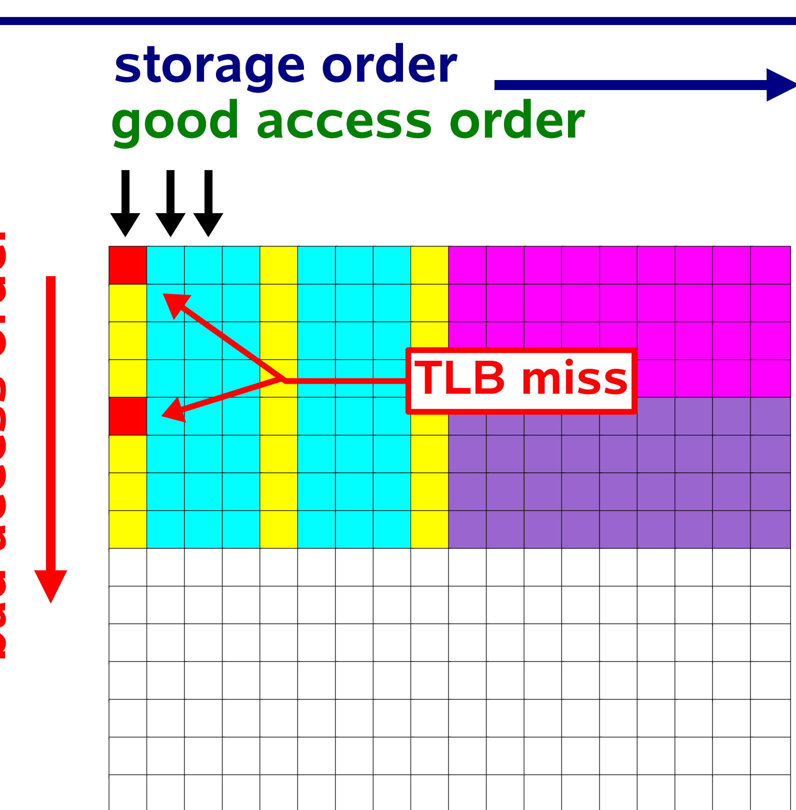


Bad Memory Access Has A Huge Impact On Performance

Copyright©2003 Sun Microsystems

Bad Memory Access (C)



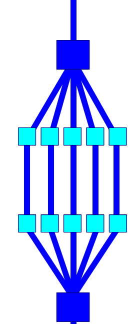


- = TLB miss
- = D-cache miss
- = Cached elements
- = Virtual memory page

- ✔ ***If the entire matrix fits in the cache, the access pattern hardly matters***
- ✔ ***For out-of-cache matrices however, the access pattern does matter***
- ✔ ***With a bad memory access pattern, we will get many more D-cache and TLB misses***

 Copyright©2003 Sun Microsystems

Performance - Matrix Summation



Row version

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        sum += x[i][j];
```

Column version

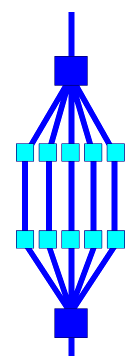
```
for (j=0; j<n; j++)
    for (i=0; i<n; i++)
        sum += x[i][j];
```


SB 1000 – USIII @ 750MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1500 Mflop/s

Note: Had to prevent the compiler from transforming the bad version into the good version !

Copyright©2003 Sun Microsystems

Comparison US-III and US-III Cu

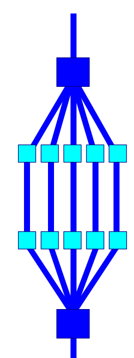


Performance (Mflop/s)
1000
100
10
1
0
1
10
100
1000
10000
100000
1000000
Memory Footprint (KByte)
Clock speed scaling
Where do these differences come from ?
Row version faster on SB1000 !
Column version faster on SF6800 !
SB1000 ROW
SB1000 COL
SF6800 ROW
SF6800 COL

SB1000 - US III @ 750 MHz
SF6800 - USIII Cu@ 900MHz

Copyright©2003 Sun Microsystems

The effect of -xprefetch=yes



Performance (Mflop/s)

1000
100
10
1

0 1 10 100 1000 10000 100000 1000000

Memory Footprint (KByte)

Prefetch slows down

Prefetch hides latency to L2-cache

Column version does not really benefit from prefetch
WHY ?

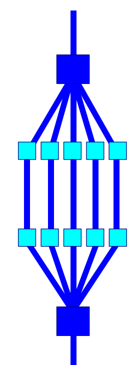
5 times faster with prefetch !

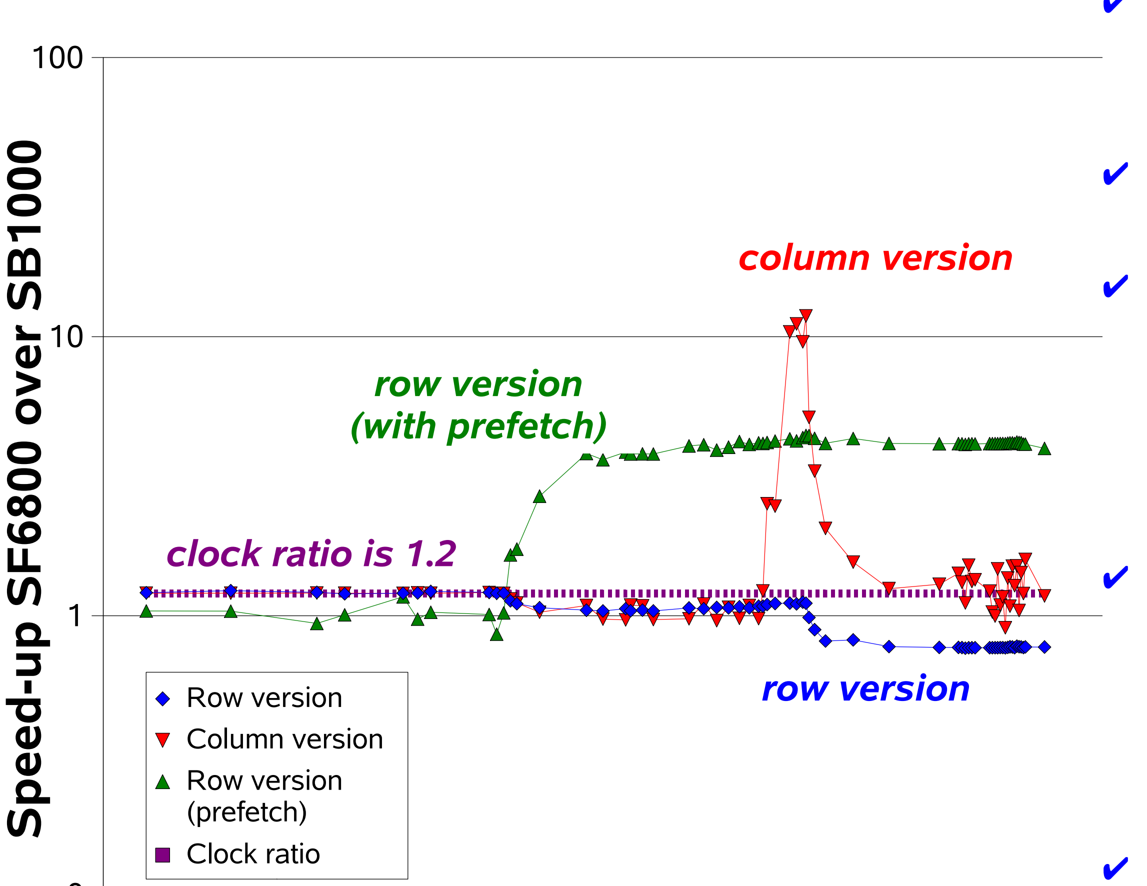
- SF6800 ROW
- SF6800 ROW pref
- SF6800 COL
- SF6800 COL pref

SF6800 – USIII Cu @ 900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

Copyright©2003 Sun Microsystems

Nothing is uniform ....



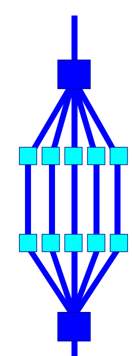


- *Very often we do not see the clock ratio ....*
- *It is either higher or lower*
- *The row version without prefetch is slower on the SF6800 for large problems (higher memory latency)*
- *Using prefetch on the row version, the SF6800 is much faster on large problems*
- *The column version takes advantage of the larger TLB capacity in the US-III Cu processor*

SB1000 - US III @ 750 MHz
SF6800 - USIII Cu@ 900MHz

Copyright©2003 Sun Microsystems

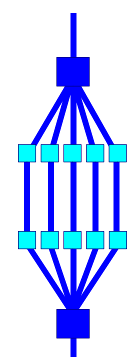
More about the TLB cache



- ❑ *Total mapping capacity = #TLB entries * Page Size*
  - ✔ *For example: 512 entries @ 8KB => 4 MB*
- ❑ *If an application suffers from excessive TLB misses:*
  - *Use UltraSPARC-III Cu*
  - *Use Solaris 9 with large page support:*
    - ✔ *The ppgsz command is your friend*
      - ◆ *Total capacity using large pages is 2 GB !*
    - ✔ *The pagesize command (with the -a option) will show you which page size(s) your system supports*
    - ✔ *The pmap command can be used to check which page size(s) the application is using*

Copyright©2003 Sun Microsystems

Using large pages with ppgsz *



Performance (Mflop/s)

1000
100
10
1

0 1 10 100 1000 10000 100000 1000000

Memory Footprint (KByte)

- SF6800 ROW pref
- SF6800 COL pref
- SF6800 ROW pref - 4MB
- SF6800 COL pref - 4MB

1.5 times faster with 4 MB pages

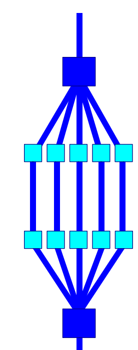
9 times faster with 4 MB pages !

SF6800 – USIII Cu @ 900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

) This command is available as of Solaris 9

Copyright©2003 Sun Microsystems

Doing The Right Thing Helps



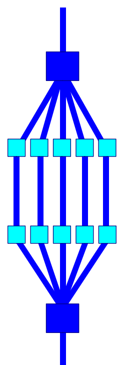
- Correct memory access
- Latest processor
- Latest software
- Prefetch
- Large pages

SB1000 COL
SF6800 COL
SF6800 ROW pref - 4MB

Performance (Mflop/s)
600
550
500
450
400
350
300
250
200
150
100
50
0

0 1 10 100 1000 10000 100000 1000000
Memory Footprint (KByte)

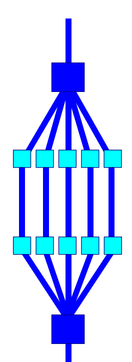
Copyright©2003 Sun Microsystems



Intro Performance Tuning

 Copyright©2003 Sun Microsystems

Performance Factors



Performance

Micro processor
Features
Availability
Speed

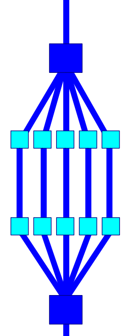
User
Algorithm Expertise
Experience
Time

Software
Compilers
Libraries
Operating System

System
I/O
Memory
Config
#Procs
Inter connect
Addr. B/W
Data B/W
Latencies

Copyright©2003 Sun Microsystems

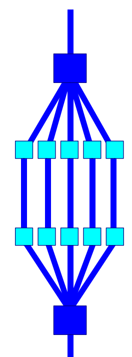
Assembly Listing Example



```
                          .............
/* 0x0820    79 (139 143) */     faddd   %f38,%f62,%f28
/* 0x0824   143 (139 159) */     fdivd   %f18,%f26,%f62
/* 0x0828     0 (139 140) */     add %l7,%o5,%o5
/* 0x082c     0 (140 141) */     prefetch    [%o5+128],0
/* 0x0830    62 (141 143) */     ld  [%fp-1080],%i0
/* 0x0834     0 (142 144) */     ld  [%fp-1104],%o4
/* 0x0838   127 (142 146) */     fmuld   %f22,%f32,%f32
/* 0x083c   124 (143 147) */     ldd [%fp-368],%f22
/* 0x0840     0 (144 146) */     ld  [%fp-1024],%o5
/* 0x0844     0 (144 145) */     add %g3,%o4,%o4
/* 0x0848     0 (145 146) */     prefetch    [%o4+128],2
/* 0x084c    62 (146 150) */     ldd [%g4+%i0],%f38
/* 0x0850     0 (146 147) */     add %l7,%o5,%o5
/* 0x0854     0 (147 148) */     prefetch    [%o5+128],0
/* 0x0858    39 (148 150) */     ld  [%fp-1116],%i0
/* 0x085c     0 (149 151) */     ld  [%fp-1032],%o4
/* 0x0860     0 (150 152) */     ld  [%fp-1120],%o5
/* 0x0864    39 (151 155) */     ldd [%g4+%i0],%f40
                          .............
```


Copyright©2003 Sun Microsystems

What to tune ?



Execution time T = Ti + Td

Ti = Time to execute the instructions
Td = Time to move data in and out

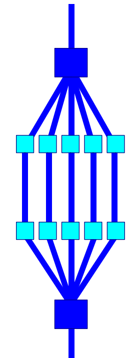
*Ti = Σinstructions * (# cycles/instruction)*

*Td = Σmemops * (# cycles/memop)*

All these 4 components may be influenced through optimization techniques

 Copyright©2003 Sun Microsystems

Instruction Execution Time

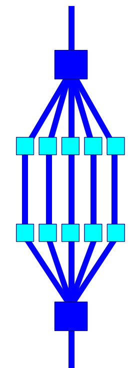


$$Ti = \Sigma instructions * (\# cycles/instruction)$$

Feature	Benefit	Requires hw	Requires sw
superscalar	*less cycles/inst*	+	+
modulo scheduling	*more superscalar*	-	+
code tuning	*less instructions*	-	+

Copyright©2003 Sun Microsystems

Moving Data

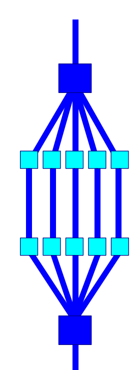


$$Td = \sum memops * (\# cycles/memop)$$

Feature	Benefit	Requires hw	Requires sw
caches	*less cycles/memop*	+	+
prefetch	*less cycles/memop*	+	+
code tuning	*less cycles/memop* *less memops*	-	+

Copyright©2003 Sun Microsystems

Four Different Ways To Optimize

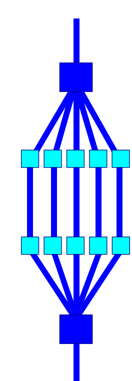


- ***Operating System features***
  - ***Effort: nothing, just use them***
- ***Faster libraries***
  - ***Effort: relink your application***
- ***The compiler***
  - ***Effort: read***
- ***Source code changes***
  - ***Effort: "unlimited"***

In practice one tends to use a combination of all of these four

 Copyright©2003 Sun Microsystems

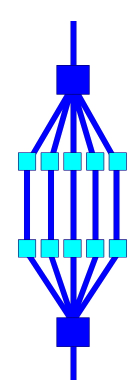
The Solaris Operating System



- ***Large Pages***
- ***Single Thread Library***
- ***Multi-threaded malloc***
- ***Memory Placement Optimization***
- **.......**

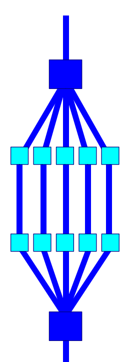
 Copyright©2003 Sun Microsystems

Faster Libraries



- *Faster intrinsics*
  - *Examples: libmopt and libmvec*
  - *Additional options to support this are available*
- *The Sun Performance Library*
  - *Available in Fortran and C*
  - *Highly tuned versions of BLAS 1-3, LAPACK*
  - *Optimized Fast Fourier Transforms*
  - *Many routines have been parallelized for shared memory*

Copyright©2003 Sun Microsystems

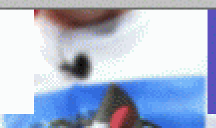


The Sun Compilers

(Sun ONE Studio Compiler Collection)

 Copyright©2003 Sun Microsystems

Developer Collection Portal



File Help

Back Forward Reload Home Search Netscape Print Security Stop

Bookmarks Netsite: http://developers.sun.com/

Members WebMail Connections BizJournal SmartUpdate Mktplace

The Source for Developers
» Products & Technologies » Technical Topics

developers.sun.com

Register?

The Source for Developers

Products & Technologies

Java Technology
- J2EE
- J2SE
- J2ME
- Java Card
- XML
- Other Technologies

Operating Systems
- Solaris
- Linux

Development Tools
- Java Tools
- Compiler Collection
- Web Services Platform, Developer Edition

Web & Directory Servers
- Identity Server
- Web Server
- Directory Server
- Portal Server

» See all

Application & Integration Servers
- Application Server

Technical Topics

The following topics organize advice, best practices, and other resources to enhance your productivity, provide information about our products and technologies, and help solve your critical business and computing problems.

- Identity
- Security
- Mobility
- Desktop
- Web Services
- Globalization
- Emerging Technologies

Compiler Collection

What's New

June, 2003
... Sun Developer Network. Take advantage of the tools, technologies, and expertise necessary to support your entire development lifecycle -- from initial planning to product release.
» Read more

April, 2003
JAX-RPC on the Sun ONE Web Services Platform Developer Edition, Part 1. This paper describes how to use Java API for XML-based remote procedure calls (JAX-RPC) with Sun ONE Application Server 7 and Sun ONE Studio 4, Update 1, Enterprise Edition.
» Read more

Community

Developer Resources
- Downloads
- APIs
- Documentation
- FAQs
- Code Samples & Apps
- BluePrints
- Technical Articles & Tips
- White Papers
- Case Studies
- Learning Resources

Related Links

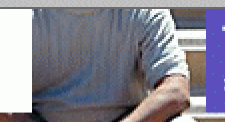
Sun Resources
- Sun Open Source
- Download Center
- Tech Support
- Product Documentation
- Training
- System Administration

JavaOne Conference
The JavaOne Pavilion at the JavaOne Conference, features the latest products and technologies from leading companies.

100%

Copyright©2003 Sun Microsystems

Compiler Collection Portal



File Help

Back Forward Reload Home Search Netscape Print Security Stop

Bookmarks Netsite: http://developers.sun.com/prodtech/cc/

Members WebMail Connections BizJournal SmartUpdate Mktplace

The Source for Developers
» Products & Technologies » Technical Topics

The Source > Products & Technologies > Development Tools >

Profile and Registration | Why Register?

Technical Info

Development Tools
Sun ONE Studio Compiler Collection

Reference
- Documentation
- Code Samples & Apps
- Technical Articles & Tips
- Knowledge Base

Community
- Forums
- User Groups

Learning
- Training

Srikanth Raju, Staff Engineer, Technology Evangelist, Java and Wireless Technology

Develop scalable, reliable, high-performance C, C++, and Fortran applications for Solaris SPARC environments with the comprehensive set of command line-based (CLI) tools in Sun ONE Studio 8, Compiler Collection. You can debug mixed language C, C++, and Java software applications, and support multiple platforms with gcc, Visual C++, C99, OpenMP, and Fortran 2000 compatibility. Or do GUI-based development with Sun ONE Studio 7, Enterprise Edition for Solaris, which bundles the Compiler Collection tools with the Enterprise Edition Java tools.

What's New

Download Sun ONE Studio 8. The latest release, Sun ONE Studio 8, Compiler Collection, is now available on SPARC and x86 Platforms.
» Read more

April 2003
Compiling for the UltraSPARC IIICu Processor. Suggests how to get the best performance from the UltraSPARC-IIICu processor by compiling with the right set of optimization options.
» Read more

Community

Events
SC2003, Supercomputing Conference - Igniting Innovation - Phoenix Civic Plaza Convention Center - Phoenix, Arizona. November 15-21, 2003
» Read more

Subscribe to Newsletters

- Solaris Developer Connection (SolDC) Newsletter

Related Links

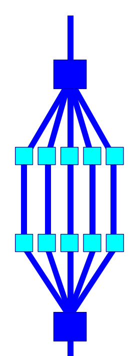
Top Downloads
- Sun ONE Studio 8, Compiler Collection
- Sun ONE Studio 7, Enterprise Edition for Solaris
- Solaris

Products & Technologies
- Sun ONE Studio 7, Enterprise Edition for Solaris
- Solaris

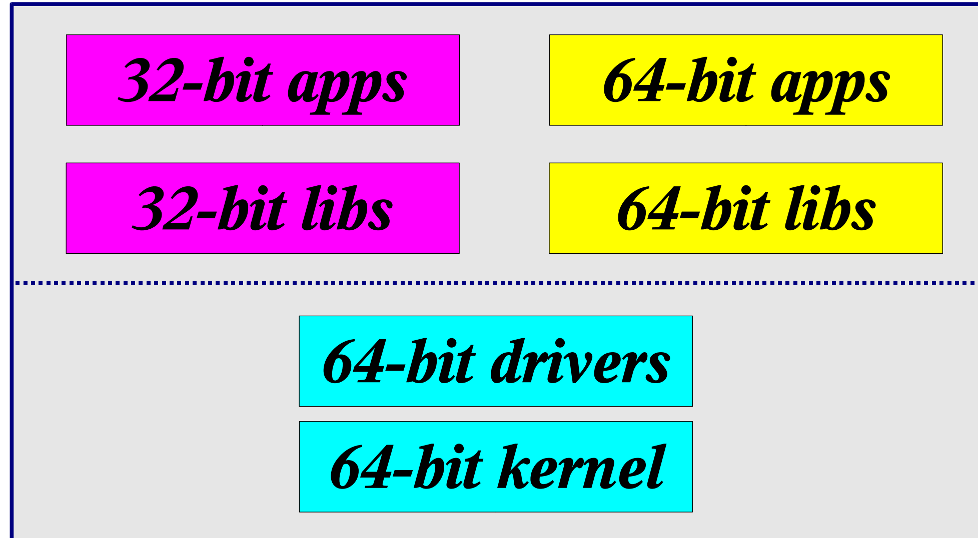
Sun Resources
- Product Information
- Awards & Endorsements

Copyright©2003 Sun Microsystems

Solaris is a 64-bit OS

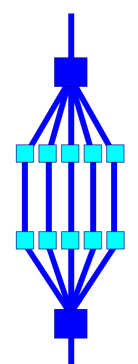


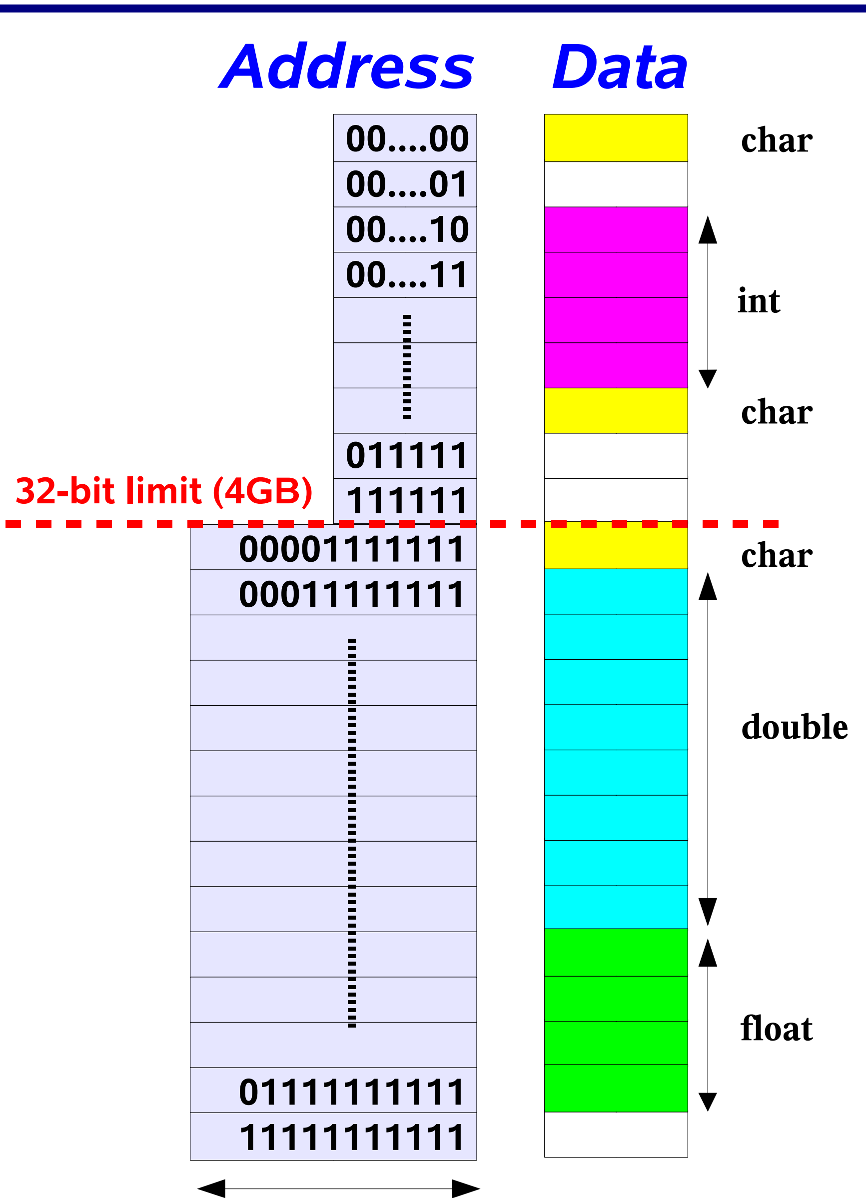
- *Solaris 7 (and above) is a full 64-bit operating system*
- *Implication: the address space of a single application can be larger than 4 GB*



 Copyright©2003 Sun Microsystems

Address =/= Data !

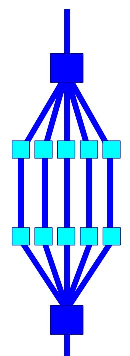




- Sun systems are 'byte addressable'
- This means that memory can be accessed at the byte level
- The size of the data type can range from 1 byte to 16 bytes
- This means that for an 'n' sized data type, the next element is 'n' bytes further
- This increment has nothing to do with the size of the address (32-bit or 64-bit)

Copyright©2003 Sun Microsystems

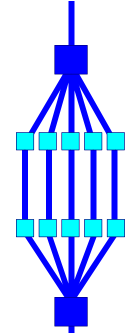
ILP32 and LP64



C data type	ILP32 (bits)	LP64 (bits)
char	8	same
short	16	same
int	32	same
long	32	64
long long	64	same
pointer	32	64
enum	32	same
float	32	same
double	64	same
long double	128	same

Copyright©2003 Sun Microsystems

About aliasing



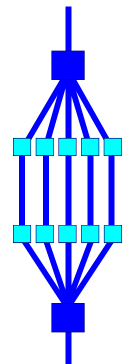
- *This function very much looks like a vector update:*

```
void vadd(int n, float *p, float *q, float *r)
{
   int i;

   for (i=0; i<n; i++)
       *p++ = *q++ + *r++;
}
```

- *However, the C compiler has to assume p, q and r overlap*
- *This is referred to as "the aliasing problem"*
- *Only the programmer will know whether this overlap is true or not*

Copyright©2003 Sun Microsystems

About overlap/1



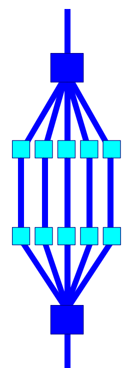
```
void vadd(int n, float *p, float *q, float *r)
{
   for (i=0; i<n; i++)
        *p++ = *q++ + *r++;
}
```

```
(void) vadd(n, &a[1], &a[0], &r[0])
```

```
void vadd(n, &a[1], &a[0], &r[0])
{
   for (i=0; i<n; i++)
        a[i+1] = a[i] + r[i];
}
```


Copyright©2003 Sun Microsystems

About overlap/2



```
void vadd(n, &a[1], &a[0], &r[0])
{
   for (i=0; i<n; i++)
       a[i+1] = a[i] + r[i];
}
```

```
a[i+1] = a[i]   + r[i];

a[i+2] = a[i+1] + r[i+1];

a[i+3] = a[i+2] + r[i+2];
```

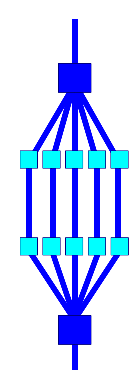
Data Dependency

Use the* -xrestrict *option if pointers do <u>not</u> overlap*

) One can also use a pragma for this

Copyright©2003 Sun Microsystems

Aliasing

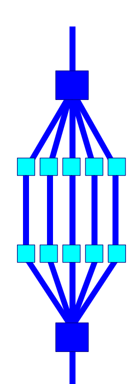


- *The example just shown is a classical* ***aliasing*** *problem*
- *The C compiler* <u>**has to assume**</u> *that different pointers may overlap **
  - *Correct, but non-optimal, code will be generated*
  - *The programmer may know that there is no overlap*
- *How to inform the compiler there is no overlap:*
  - *Use the* ***-xrestrict*** *option*
  - *Put a* **pipeloop** *pragma/directive in the source*
- *However, remember that* <u>*you are then responsible*</u> *that the underlying assumption is not violated !*

**) Note that in Fortran this would be illegal if the names of the arrays are different*

 Copyright©2003 Sun Microsystems

The -fast macro expansion



Common options (Fortran, C, C++)

- *-xtarget=native*
- *-xO5*
- *-fns*
- *-fsimple=2*
- *-dalign*
- *-xlibmil*

Additional options (Fortran, C)

- *-xdepend*
- *-xprefetch=yes*

Fortran

- *-xprefetch_level=2*
- *-xvector=yes*
- *-pad=local*
- *-xlibmopt*
- *-ftrap=common*

C

- *-xprefetch_level=1*
- *-fsingle*
- *-xbuiltin=%all*
- *-xalias_level=basic*
- *-ftrap=%none*

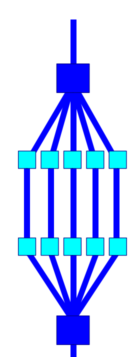
C++

- *-xbuiltin=%all*
- *-xlibmopt*
- *-ftrap=%none*

Note: Valid for the 7.0 release

Copyright©2003 Sun Microsystems

Instruction set and chip

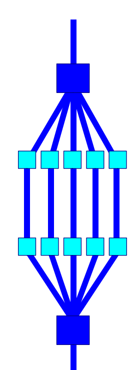


- **For best performance, one should:**
  - ***Use the most powerful SPARC Architecture Instruction Set available today (-xarch option)***
    - ✔ ***Impacts performance and backward compatibility***
  - ***Ask the compiler to tune for the UltraSPARC-II (or III) processor (-xchip option)***
    - ✔ ***Impacts performance only***
- ***The compiler takes defaults for this *, but we recommend to specify this explicitly***

****) The compiler defaults depend on the system that you compile on***

Copyright©2003 Sun Microsystems

Minimal Effort



In general, one obtains very good performance out of the Sun compilers by just using 3 options on the compile and link line:

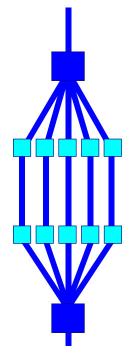
For the UltraSPARC-III Cu processor:

```
-fast -xchip=ultra3cu -xarch=v8plusb    (32-bit addressing)
-fast -xchip=ultra3cu -xarch=v9b        (64-bit addressing)
```

- ***The -fast option is a macro that expands to a series of options***
- ***Purpose of -fast is to give you very good performance with just one single option***
- ***Works fine for many applications, but does make some assumptions. When in doubt whether this is acceptable, one is advised to check the documentation about the details.***

Copyright©2003 Sun Microsystems

Recommendation



Fragment from make file:

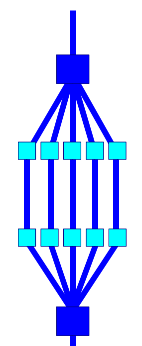
```
ISA    = -xarch=v8plusb
CHIP   = -xchip=ultra3cu
CACHE  = -xcache=64/32/4:8192/512/2
FFLAGS = -fast .... $(ISA) $(CHIP) $(CACHE)
```

This will ensure that the settings desired are not implicitly overruled through the -fast option

Copyright©2003 Sun Microsystems

The effect of compiler optimizations

Sun microsystems
take it to the nth



Speed-up on SF6800

22
20
18
16
14
12
10
8
6
4
2
0

0 1 2 3 4 5 6 7 8 9 10 11

Job ID

average

stdev

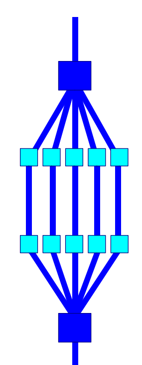
stdev

- ♦ *Ten real user codes*
  - ✔ *Chemistry*
  - ✔ *Physics*
  - ✔ *Mathematics*
- ♦ *Identified by Job ID*
  - ✔ *Different versions*
  - ✔ *Different jobs*
- ♦ *Compiled and ran:*
  - ✔ *No optimization*
  - ✔ *Full optimization*
- ♦ *Plot ratio of times*

SF6800 - USIII Cu@ 900MHz
S1SCC 7.0
Solaris 9

Copyright©2003 Sun Microsystems

Compiler Components



source	source	source
f95	cc	CC

driver

Intermediate Representation (IR)

IRopt

CG

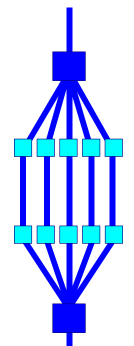
important for performance

object file

CG : Code Generator and Instruction Level Optimizer

Copyright©2003 Sun Microsystems

Who Does What ?



Instruction generation and scheduling

CG

High-level transformations

IRopt

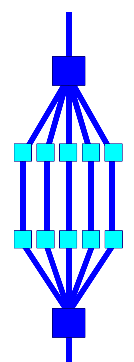
Performance

Reg.
L1
L2
Main Memory
Virtual memory

Problem size

Copyright©2003 Sun Microsystems

Example: Matrix * vector product



j

i

=

*

Good Memory Access +
2 Loads +

```
for (i=0; i<m; i++)
{
    sum = 0.0;
    for (j=0; j<n; j++)
      sum += b[i*n+j]*c[j];
    a[i] = sum;
}
```

=

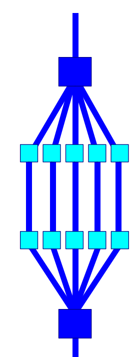
*

Bad Memory Access –
2 Loads and 1 Store –

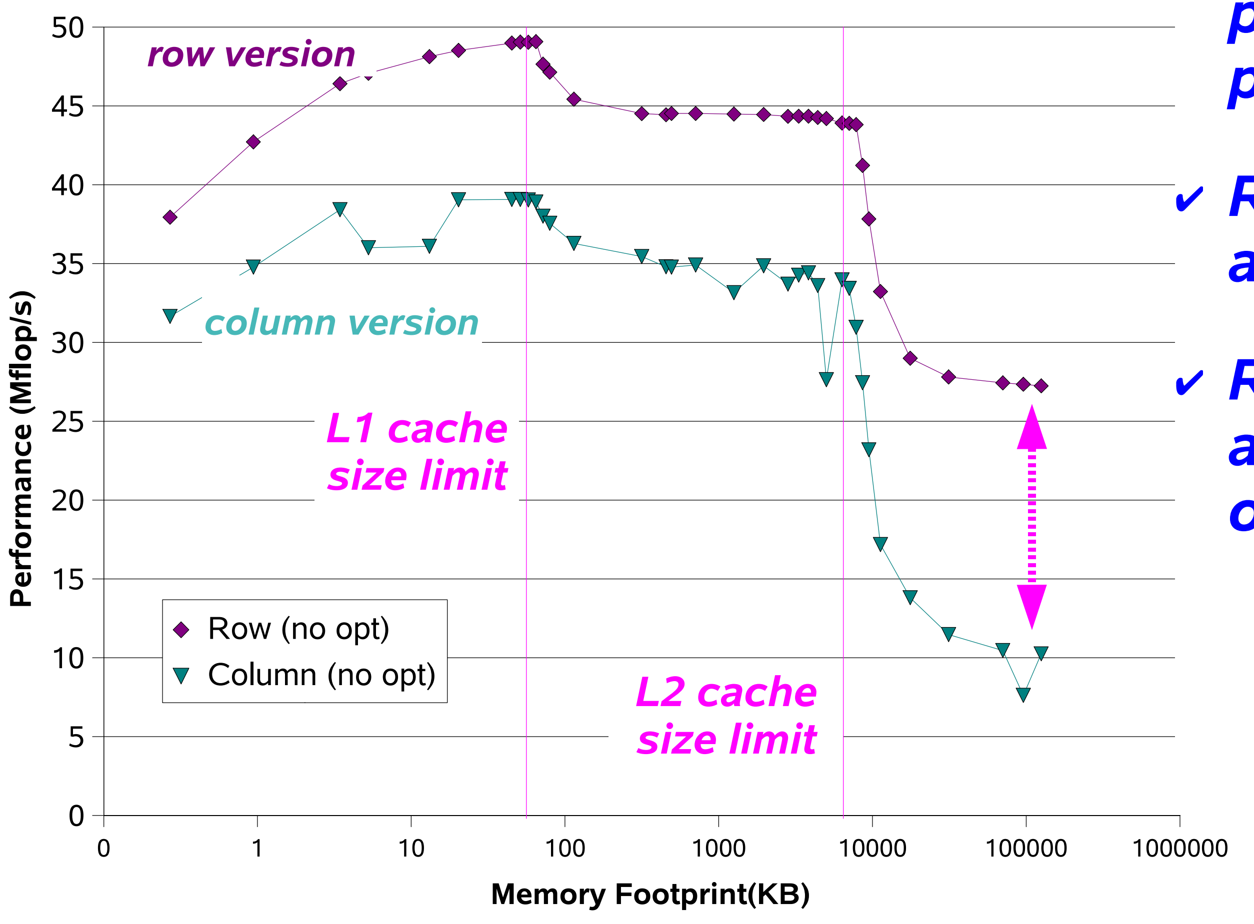
```
for (i=0; i<m; i++)
    a[i] = b[i*n]*c[0]

for (j=1; j<n; j++)
   for (i=0; i<m; i++)
      a[i] += b[i*n+j]*c[j];
```


Copyright©2003 Sun Microsystems

Matrix * vector - Unoptimized



- ✔ *Absolute performance is poor*
- ✔ *Row version is always faster*
- ✔ *Row version is about 2.5x faster on large problems*

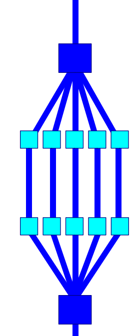


SF6800 - USIII Cu@ 900MHz
S1SCC 8.0
Solaris 9

Copyright©2003 Sun Microsystems

Matrix * vector product - Optimized

Sun microsystems
take it to the nth



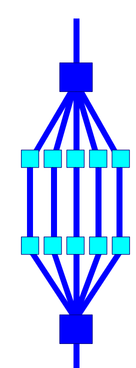
Both versions perform more or less the same now !

SF6800 - USIII Cu@ 900MHz
S1SCC 8.0
Solaris 9

Options:
-fast -xrestrict -xprefetch_level=3

Copyright©2003 Sun Microsystems

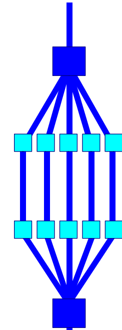
Compiler Commentary



- Get information about the optimizations performed:
  - Loop transformations (iropt phase)
  - Instruction scheduling (cg phase)
- How to get these messages:
  - Add -g to the other compiler options you use
  - Example: % cc -c -fast -xarch=v8plusb -g funcA.c
- Two ways to display the compiler messages:
  - Use the er_src command to display the messages on the screen
    - ✔ Example: % er_src funcA.o
  - Automatically shown in analyzer source window

Copyright©2003 Sun Microsystems

Example Compiler Optimizations



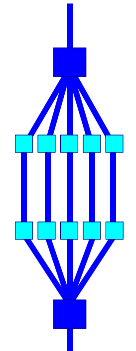
```
  1. void mxv_col(int m, int n, double *a, double *b, double *c)
  2. {
  3.     int i, j;
  4.
Loop below fused with loop on line 10
  5.     for (i=0; i<m; i++)
  6.         a[i] = b[i*n]*c[0];
  7.
Loop below interchanged with loop on line 10
  8.     for (j=1; j<n; j++)
  9.     {
Loop below interchanged with loop on line 8
Loop below fused with loop on line 5
 10.       for (i=0; i<m; i++)

Loop below pipelined with steady-state cycle count = 2
before unrolling
Loop below unrolled 8 times
Loop below has 2 loads, 0 stores, 4 prefetches, 1 FPadds,
               1 FPmuls, and 0 FPdivs per iteration
 11.         a[i] += b[i*n+j]*c[j];
 12.     }
 13. }
```

IRopt messages

CG messages

Copyright©2003 Sun Microsystems

Matrix * vector: Sun Perf. Library



Performance (Mflop/s)

1100
1000
900
800
700
600
500
400
300
200
100
0

0.1 1 10 100 1000 10000 100000 1000000

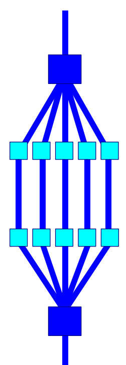
Memory Footprint(KB)

Row (no opt)
Row (opt)
Column (no opt)
Column (opt)
Library-version

Library version has been tuned very well

SF6800 - USIII Cu@ 900MHz
S1SCC 8.0
Solaris 9

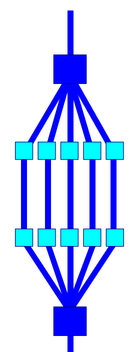
Copyright©2003 Sun Microsystems



The Sun Performance Analyzer

 Copyright©2003 Sun Microsystems

Statistical Callstack Sampling



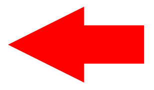
The program

main
11101110
11111110
10111100
00111010

sub1
11011011
01011100

sub2
11100110
10101010
10011011
10001111

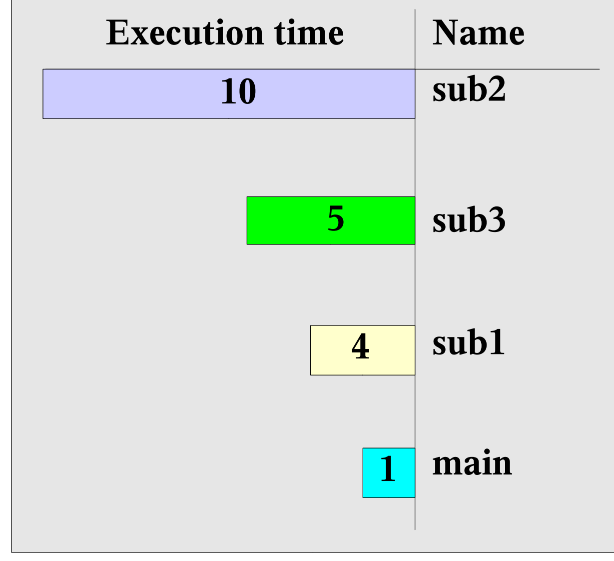
sub3
00110111
00001100
00000001
11010110



❶ The program is stopped at regular intervals

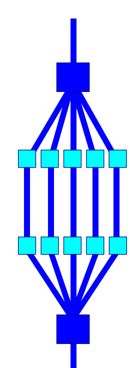
❷ The Program Counter (PC) and other information is recorded when the program stops

❸ A histogram with the execution time distribution is produced



Copyright©2003 Sun Microsystems

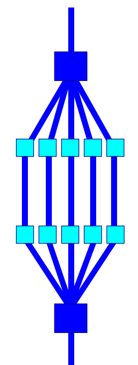
The Performance Analyzer



- ❑ *Worldclass Product ! Very Easy To Use !*
- ❑ *Supports multi-threaded programs*
- ❑ *Uses statistical callstack sampling*
  - *Clock-based*
  - *Hardware counter-based: memory and cache counters*
  - *Synchronization wait tracing*
- ❑ *Offers top to bottom performance data:*
  - *Routine level*
  - *Statement level*
  - *Instruction level*
  - *Caller and callee level*
- ❑ *All this information can be obtained in a single run !*

Copyright©2003 Sun Microsystems

Caller-callee info



10

5

15

20

50

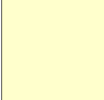
40

10

***Exclusive time** - Time spent in routine, excluding time spent calling other routines*

***Inclusive time** -Time spent in routine, including time spent in other routines*

Example:

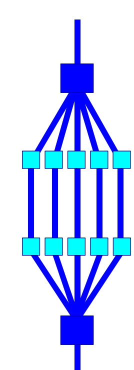


Exclusive time : 5 seconds
Inclusive time : 5+20+10=35 seconds

The Performance Analyzer accurately attributes the time the multiple callers of this leaf routine contribute to the total

Copyright©2003 Sun Microsystems

How To Use The Analyzer

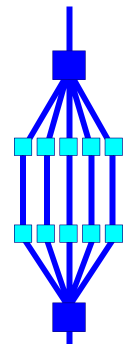


- ***Works with unmodified binaries***
- ***For the most complete information:*** ***recompile with -g***
- ***Three ways of using the Performance Analyzer* :***
  - ***Through the collect/analyzer commands:***
    - % ***collect - gather the performance data***
    - % ***analyzer - GUI to analyze the data***
  - ***Through the Integrated Development Environment:***
    - % ***runide.sh***
  - ***Through dbx (not covered here)***
- ***Optional: use the er_print command to analyze the data and get the information in ASCII format***

**) The default path is /opt/SUNWspro/bin*

Copyright©2003 Sun Microsystems

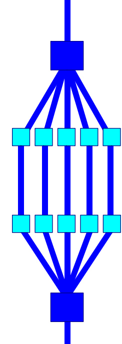
The collect command/1



```
% collect
NOTE: SunOS 5.8 system "hpc" is correctly patched and set up for use with the
      Performance tools.

 usage:  collect <args> target <target-args>

 Sun Performance Tools 7.1 Dev 2003/01/08
  -p <interval>     specify clock-profiling
     Clock profiling interval range on this system = 0.500 - 1000.000 millisec.;
     resolution = 0.001 millisec.
  -s <threshold>    specify synchronization wait tracing
  -H {on|off}  specify heap tracing
  -m {on|off}  specify MPI tracing
  -h <counter>[,<interval>[<counter2>[,<interval2>]]]  specify HW counter profiling
If <counter2> is specified, <counter> and <counter2> must be on different registers
For counters that count load or store instructions, if the counter name is preceeded by +,
the collector attempts to determine the PC and virtual address of the triggering load or
store; the + is ignored for counters not counting loads or stores
   HW counters available for profiling:

   CPU Cycles (cycles = Cycle_cnt/*) 9999991  hi=1000003, lo=100000007
   Instructions Executed (insts = Instr_cnt/*) 9999991  hi=1000003, lo=100000007
   I$ Misses (icm = IC_miss/1) 100003  hi=10007, lo=1000003
   D$ Read Misses (dcrm = DC_rd_miss/1) 100003  hi=10007, lo=1000003 ld
   D$ Write Misses (dcwm = DC_wr_miss/1) 100003  hi=10007, lo=1000003 st
   D$ Read Refs (dcr = DC_rd/0) 1000003  hi=100003, lo=9999991 ld
   D$ Write Refs (dcw = DC_wr/0) 1000003  hi=100003, lo=9999991 st
   E$ Refs (ecref = EC_ref/0) 1000003  hi=100003, lo=9999991 ld-st

               .......... etc .................
```


Copyright©2003 Sun Microsystems

The collect command/2

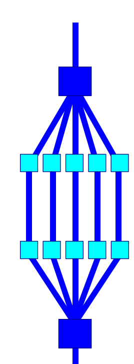


```
                   .......... etc ..................
   -j {on|off}  specify Java profiling
   -x specify leaving the target waiting for a debugger attach
   -n dry run -- don't run target or collect performance data
   -y <signal>[,r]   specify delayed initialization and pause/resume signal
      When set, the target starts in paused mode;
   if the optional r is provided, it starts in resumed mode
   -F {on|off}  specify following descendant processes
   -A {on|off|copy}  specify archiving of load-objects; default is on
   -S <interval>     specify periodic sampling interval (secs.)
   -L <size>    specify experiment size limit (MB.)
   -l <signal>  specify signal for samples
   -o <expt>    specify experiment name
   -d <directory>    specify experiment directory
   -g <groupname>    specify experiment group
   -v print expanded log of processing
   -R show the README file and exit
   -V print version number and exit

   Default experiment:
 expt_name = test.1.er
 clock profiling enabled, 10.007 millisec.
 descendant processes will not be followed
 periodic sampling, 1 secs.
 experiment size limit 2000 MB.
 experiment archiving: on
 data descriptor: "p:10007;S:1;L:2000;A:1;"
 host: `hpc', cpuver = 1002, ncpus = 2, clock frequency 750 MHz.
```

see the collect.1 man page for more information

Copyright©2003 Sun Microsystems

Start the Analyzer



Assume you have used "collect" to run one or more performance experiments

For example, like this: ***% collect a.out***

Now, start the analyzer and load the experiment(s) you're interested in:

% analyzer

Look in: pinwheel

bin
src
pinwheel.par1.er
pinwheel.par2.er
pinwheel.ser.er

There are three experiments here

File name: pinwheel.ser.er
Files of type: Performance Analyzer Experiment (*...
Open
Cancel

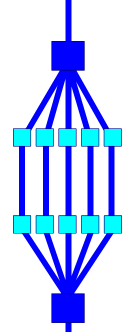
You can also load the experiment(s) directly

% analyzer "exp_name(s)"

Within the analyzer, experiments can also be dropped and reloaded

Copyright©2003 Sun Microsystems

Main Analyzer Window



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Disassembly | Timeline | Statistics | Experiments

Functions | Callers-Callees | Source

User CPU (sec.)	User CPU (%)	User CPU (sec.)	Wall (sec.)	Name
117.540	100.0	117.540	123.670	<Total>
113.870	96.9	113.870	118.070	calc_r_loop_on_neighbours
0.860	0.7	115.420	0.880	calc_r
0.700	0.6	1.410	0.710	_doprnt
0.450	0.4	0.990	0.500	init_visual_input_on_V1
0.290	0.2	0.290	1.890	_write
0.170	0.1	0.170	0.190	round_coord_cyclic
0.150	0.1	0.450	0.150	__k_double_to_decimal
0.120	0.1	0.120	0.120	__arint_set_n
0.080	0.1	0.080	0.080	_realbufend
0.080	0.1	0.630	0.080	fconvert
0.080	0.1	0.360	0.080	printf
0.070	0.1	0.540	0.070	double_to_decimal
0.060	0.1	1.190	0.060	fprintf
0.050	0.0	0.200	0.050	init_variables_loop_on_neighbo
0.040	0.0	0.040	0.040	<static>@0xb8484
0.040	0.0	0.040	0.040	__four_digits_quick
0.040	0.0	0.040	0.050	memcpy
0.030	0.0	0.130	0.030	__double_to_digits
0.030	0.0	0.030	0.030	__libmopt__rem_pio2
0.030	0.0	0.030	0.030	calc_distance1

Summary | Event | Legend

Data for Selected Function/Load-Object:

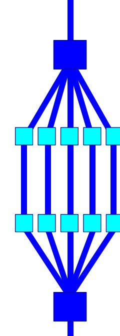
Name: calc_r_loop_on_neighbours
C Address: 2:0x00003560
Size: 144
Source File: lyzer_70/pinwheel/src/pinwheel_base.
Object File: er_70/pinwheel/src/pinwheel_base_16.
oad Object: <pinwheel_ser.exe>
gled Name:
Aliases:

Process Times (sec.) / Counts

	Exclusive	Inclusive
User CPU:	113.870 (96.9%)	113.870 (96.9%
Wall:	118.070 (95.5%)	118.070 (95.5%
Total LWP:	118.070 (95.5%)	118.070 (95.5%
System CPU:	3.970 (75.6%)	3.970 (75.6%
Wait CPU:	0.160 (76.2%)	0.160 (76.2%
t Page Fault:	0. (0. %)	0. (0. %
a Page Fault:	0.070 (100.0%)	0.070 (100.0%
Other Wait:	0. (0. %)	0. (0. %

Copyright©2003 Sun Microsystems

Information On Experiment(s)



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Disassembly | Timeline | Statistics | Experiments

Functions | Callers-Callees | Source

<collaudit.so> (/d2/ruudp/Archive2/kraktoa/current/bin/../YNH/bin/../lib/dbxru
<ld.so.1> (/usr/lib/ld.so.1)
<libcollector.so> (/d2/ruudp/Archive2/kraktoa/current/bin/../YNH/bin/../lib/dbx
<libc.so.1> (/usr/lib/libc.so.1)
<libdl.so.1> (/usr/lib/libdl.so.1)
No stabs information in /mnt/home/ruudp/SunTune/Examples_New/analy
<libc_psr.so.1> (/usr/platform/SUNW,Ultra-2/lib/libc_psr.so.1)
No stabs information in /mnt/home/ruudp/SunTune/Examples_New/analy

/mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/pinwheel.se

Target command: './bin/pinwheel_ser.exe init0'
Process pid 5296, ppid 5286, pgrp 5286, sid 28601
Collector version: 'Forte Developer 7 Performance Analyzer 7.0 Dev 2001/09/
Host 'hpc', OS 'SunOS 5.8 ', page size 8192
Experiment started Sat Jun 22 11:30:02 2002

Data collection parameters:
Clock-profiling, interval = 10 millisecs.
Periodic sampling, 1 secs.

Error/Warning Logs:

Summary | Event | Legend

Data for Selected Function/Load-Object:

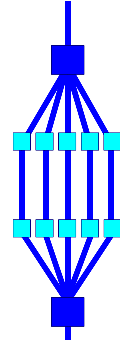
Name:	calc_r_loop_on_neighbours
C Address:	2:0x00003560
Size:	144
Source File:	lyzer_70/pinwheel/src/pinwheel_base.
Object File:	er_70/pinwheel/src/pinwheel_base_16.
oad Object:	<pinwheel_ser.exe>
gled Name:	
Aliases:	

Process Times (sec.) / Counts

	Exclusive	Inclusive
User CPU:	113.870 (96.9%)	113.870 (96.9%
Wall:	118.070 (95.5%)	118.070 (95.5%
Total LWP:	118.070 (95.5%)	118.070 (95.5%
System CPU:	3.970 (75.6%)	3.970 (75.6%
Wait CPU:	0.160 (76.2%)	0.160 (76.2%
t Page Fault:	0. (0. %)	0. (0. %
a Page Fault:	0.070 (100.0%)	0.070 (100.0%
Other Wait:	0. (0. %)	0. (0. %

Copyright©2003 Sun Microsystems

Statistics



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

Experiments

<Sum across selected experiments>

Execution statistics for entire program:

Start Time:	N/A
End Time:	N/A
Duration (sec):	123.668
Process Times (sec):	
User CPU:	119.235 (96.4%)
System CPU:	3.178 (2.6%)
Wait CPU:	0.581 (0.5%)
Text Page Fault:	0. (0. %)
Data Page Fault:	0.078 (0.1%)
Other Wait:	0.596 (0.5%)

Minor Page Faults:	0
Major Page Faults:	0
Process swaps:	0
Input blocks:	0
Output blocks:	47
Messages sent:	0
Messages received:	0
Signals handled:	12313
Voluntary context switches:	23
Involuntary context switches:	3825
System calls:	68956
Characters of I/O:	1657069

/mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/pinwheel.ser.er

Summary | Event | Legend

Data for Selected Funct

Name:	calc_r_loop_on
C Address:	2:0x00003560
Size:	144
Source File:	lyzer_70/pinwhe
Object File:	er_70/pinwheel/
oad Object:	<pinwheel_ser.e
gled Name:	
Aliases:	

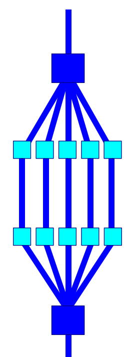
Process Times (se

Exclusiv

User CPU:	113.870 (96
Wall:	118.070 (95
Total LWP:	118.070 (95
System CPU:	3.970 (75
Wait CPU:	0.160 (76
t Page Fault:	0. (0
a Page Fault:	0.070 (100
Other Wait:	0. (0

Copyright©2003 Sun Microsystems

Filters and Metrics Selection



Experiments:

lyzer_70/pinwheel/pinwheel.ser.er

Sele... Clea... Rev...

Samples: 1-124 All

(100 % of total range: 1-124)

Threads: 1 All

(total range: 1-1)

LWPs: 1 All

(total range: 1-1)

Enable All | Disable All

Enable Selected | Disable

Filter data

OK Apply Default

Metrics Visible | Sort | Source and Disassembly

	Exclusive			Inclusive		
	Time	Value	%	Time	Value	%
User CPU	☑		☑	☑		☐
Wall	☑		☐	☐		☐
Total LWP	☐		☐	☐		☐
System CPU	☐		☐	☐		☐
Wait CPU	☐		☐	☐		☐
Text Page Fault	☐		☐	☐		☐
Data Page Fault	☐		☐	☐		☐
Other Wait	☐		☐	☐		☐
Size		☐				
PC Address		☐				

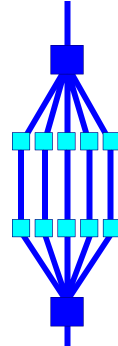
Select data

OK Apply Save Close

Note:
Favourite metrics can be saved

Copyright©2003 Sun Microsystems

Callers-Callees Information



File View Timeline   Selected Function/Load-Object: calc_r   Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

User CPU (sec.)	User CPU (%)	User CPU (sec.)	User CPU (sec.)	Wall (sec.)	Wall (sec.)	Name
115.420	100.0	0.010	117.330	120.950	0.010	do_moving_grating
0.860	0.7	0.860	115.420	0.880	0.880	calc_r
113.870	98.7	113.870	113.870	118.070	118.070	calc_r_loop_on_neighbo
0.360	0.3	0.080	0.360	0.370	0.080	printf
0.330	0.3	0.010	0.340	1.630	0.010	fflush

Summary | Event | Legend

Data for Selected Function/Load-Object:

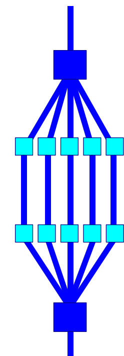
lame:	calc_r
lress:	2:0x00003254
Size:	780
File:	lyzer_70/pinwheel/src/pinwheel
t File:	er_70/pinwheel/src/pinwheel_ba
bject:	<pinwheel_ser.exe>
lame:	
ases:	

Process Times (sec.) / Counts

	Exclusive	Inclus
r CPU:	0.860 (0.7%)	115.420 (
Wall:	0.880 (0.7%)	120.950 (
l LWP:	0.880 (0.7%)	120.950 (
n CPU:	0.010 (0.2%)	5.090 (
it CPU:	0.010 (4.8%)	0.170 (
Fault:	0. (0. %)	0. (
Fault:	0. (0. %)	0.070 (
r Wait:	0. (0. %)	0.200 (

Copyright©2003 Sun Microsystems

From Source Line ....



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

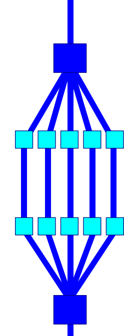
```
User CPU          User CPU   Wall      Source File: /mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/src/pi
(sec.)   (%)      (sec.)     (sec.)    Object File: /mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/src/pi
                                       Load Object: <pinwheel_ser.exe>

                                       Loop below collapsed with loop on line 956
                                        957.      for (x1 = 0; x1 < NPOSITIONS_X; x1++) {
0.140    0.1      0.140      0.160      958.        h[y1][x1] = 0.0;
                                        959.      }
                                        960.    }
                                        961.
                                        962.    loop_size = NPOSITIONS_Y;
                                        963.
                                        964.    /* #pragma parallel local(y1,x1) shared(loop_size,x1,h,V1,g,T,beta_i
                                        965.       #pragma pfor iterate(y1=0;loop_size;1) */
                                        966.  /*---- Start of taskloop RvdP/Sun --*/
                                        967.
                                        968.  /*
                                        969.    ----------------------------------------------------------------
                                        970.    Replac
                                        971.    ------                                                ------------
                                        972.  #pragma
                                        973.  #pragma
                                        974.  */
                                        975.
                                        976.  #pragma omp parallel for default(none)   \
                                        977.          private(y1,x1) shared(h,V1,g,T,beta_inv,beta)
0.       0.       0.         0.         978.    for (y1 = 0; y1 < NPOSITIONS_Y; y1++) {
0.010    0.0      0.010      0.010      979.      for (x1 = 0; x1 < NPOSITIONS_X; x1++) {
0.060    0.1      113.930    0.060      980.        calc_r_loop_on_neighbours(y1,x1);
0.180    0.2      0.180      0.180      981.        h[y1][x1] += V1[y1][x1].I;
```

Most expensive statement

Note the compiler message (more on this later)

Copyright©2003 Sun Microsystems

Down To The Instruction Level !



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

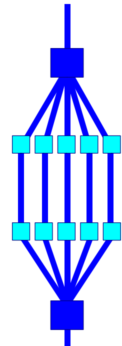
Source File: /mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/src/pi
Object File: /mnt/home/ruudp/SunTune/Examples_New/analyzer_70/pinwheel/src/pi
Load Object: <pinwheel_ser.exe>

1047. next_p = next_p->next) {
1048. h[yl][xl] += next_p->strength * Vl[next_p->y][next_p->x].r;

User CPU (sec.)	User CPU (%)	User CPU (sec.)	Wall (sec.)	Line	Address	Instruction	Operands
0.010	0.0	0.010	0.010	[1048]	13590:	ld	[%g3 + 8], %g1
0.	0.	0.	0.	[1042]	13594:	sethi	%hi(0x40c00), %g4
0.	0.	0.	0.	[1046]	13598:	sll	%o0, 7, %o0
0.	0.	0.	0.	[1042]	1359c:	add	%g4, 664, %o4
0.020	0.0	0.020	0.020	[1042]	135a0:	sethi	%hi(0x42c00), %g5
0.	0.	0.	0.	[1046]	135a4:	add	%o5, %o4, %g4
0.	0.	0.	0.010	[1046]	135a8:	ldd	[%g4 + %o0], %f6
0.	0.	0.	0.	[1042]	135ac:	add	%g5, 800, %g5
4.620	3.9	4.620	4.750	[1048]	135b0:	ld	[%g3 + 4], %g2
33.950	28.9	33.950	35.300	[1048]	135b4:	sll	%g1, 4, %g1
0.420	0.4	0.420	0.420	[1048]	135b8:	ldd	[%g3 + 16], %f2
6.740	5.7	6.740	6.920	[1048]	135bc:	add	%g2, %g1, %o1
4.080	3.5	4.080	4.290	[1048]	135c0:	sll	%o1, 5, %o2
3.840	3.3	3.840	3.960	[1048]	135c4:	add	%g5, %o2, %o3
3.620	3.1	3.620	3.810	[1048]	135c8:	ldd	[%o3 + 16], %f0
30.970	26.3	30.970	31.950	[1048]	135cc:	fmuld	%f2, %f0, %f4
12.380	10.5	12.380	12.910	[1048]	135d0:	faddd	%f6, %f4, %f6
0.	0.	0.	0.	[1048]	135d4:	std	%f6, [%g4 + %o0]
4.230	3.6	4.230	4.430	[1048]	135d8:	ld	[%g3], %g3
7.580	6.4	7.580	7.830	[1048]	135dc:	cmp	%g3, 0
0.	0.	0.	0.	[1048]	135e0:	bne,a,pt	%icc,0x135b0
0.560	0.5	0.560	0.580	[1048]	135e4:	ld	[%g3 + 8], %g1
0.080	0.1	0.080	0.090	[1048]	135e8:	jmp	%o7 + 8
0.	0.	0.	0.	[1048]	135ec:	nop	

Copyright©2003 Sun Microsystems

Timeline Overview



File View Timeline Selected Function/Load-Object: calc_r_loop_on_neighbours Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

sec 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120

Ex 1

Zoom in and out

Selects an event

Swatches | HSB | RGB

Recent:

Color Chooser

OK Apply Close

Summary | Event | Legend

Data for Current Timeline Selection

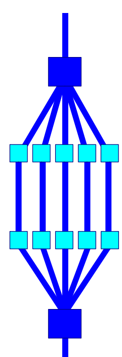
Event Type:	Profiling Data
Leaf Function:	calc_r_loop_on_r
Timestamp (sec.):	11.344519
LWP:	1
Thread:	1
CPU:	(unknown)
Duration (msec.):	10.000
Micro State:	User CPU

Call Stack for Selected Event

- calc_r_loop_on_neighbours
- calc_r
- do_moving_grating
- main
- _start

Copyright©2003 Sun Microsystems

Zoom In On Timeline



File View Timeline Selected Function/Load-Object: _smalloc Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

Summary | Event | Legend

sec .1 .2 .3 .4 .5 .6 .7 .8 .9

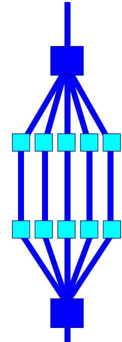
Ex 1

_morecore
_open
_realbufend
_rt_boot
_sbrk_unlocked
_setup
_smalloc
_start
_write
_xflsbuf
calc_distance
calc_distance1
calc_r
calc_r_loop_on_neighbours
call_init
collector_init
collector_open_experiment
decimal_to_double
do_moving_grating
double_to_decimal
fconvert
fflush
fopen

Can change colors

Copyright©2003 Sun Microsystems

Timeline For A Parallel Program



File View Timeline Selected Function/Load-Object: do_moving_grating Help

Find Text:

Functions | Callers-Callees | Source | Disassembly | Timeline | Statistics | Experiments

Summary | Event | Legend

sec .1 .2 .3 .4 .5

Ex 1

1

4

Thread is idle

Two threads have been used

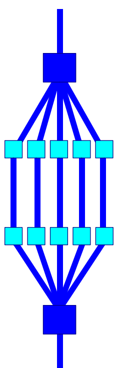
Data for Current Timeline Selection

Event Type:	Profiling Data
Leaf Function:	calc_r_loop_on_:
Timestamp (sec.):	0.331382
LWP:	1
Thread:	1
CPU:	(unknown)
Duration (msec.):	10.000
Micro State:	User CPU

Call Stack for Selected Event

- calc_r_loop_on_neighbours
- calc_r -- MP construct from line 976 [_1m
- __mt_runLoop_int_
- __mt_run_my_job_
- __mt_MasterFunction_
- calc_r
- do_moving_grating
- main
- _start

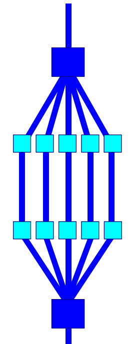
Copyright©2003 Sun Microsystems



Serial Optimization Techniques

Copyright©2003 Sun Microsystems

Who Does What ?



Instruction generation and scheduling

CG

High-level transformations

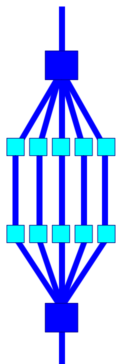
IRopt

Performance

Reg.
L1
L2
Main Memory
Virtual memory

Problem size

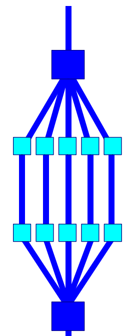
Copyright©2003 Sun Microsystems



Modulo Scheduling

 Copyright©2003 Sun Microsystems

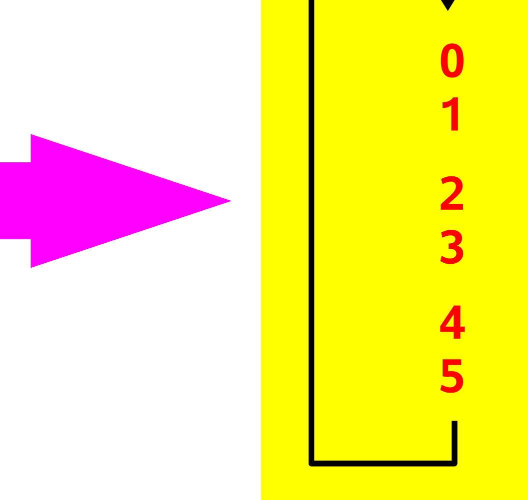
Example



Program

```
for (i=0; i<n; i++)
{
   ...statements ...
}
```

Instructions*

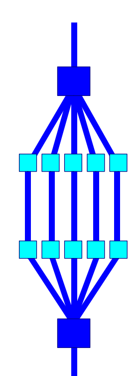


Executed serially, this loop would need 6*n cycles to be run

****) Assume each instruction takes 1 cycle to execute***

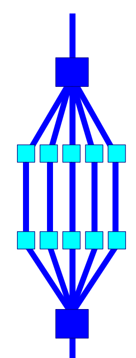
Copyright©2003 Sun Microsystems

Terminology



Copyright©2003 Sun Microsystems

The Goal of Modulo Scheduling

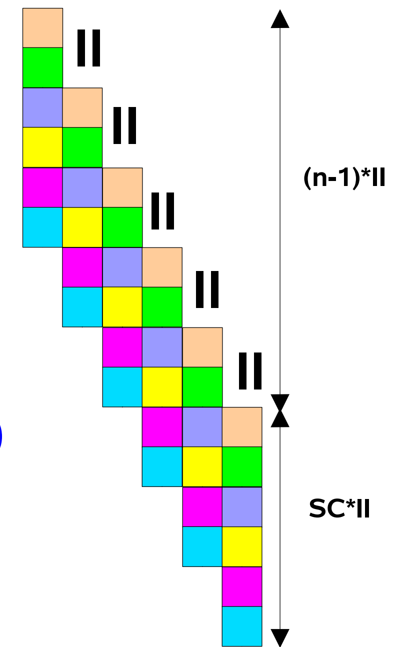


The time T(n) it takes to execute n loop iterations:

$$T(n) = (n-1) * II + SC * II = n * II + (SC-1) * II$$

The time per loop iteration:

$$T(n) / n = II + (SC-1)*II / n \rightarrow II \text{ (for n large)}$$

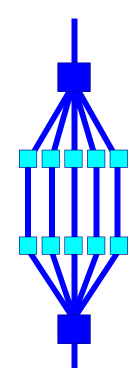


The Goal

Find a minimal value for II, such that the kernel part of the loop delivers an asymptotic speed of II cycles per loop iteration

Copyright©2003 Sun Microsystems

About the II value*

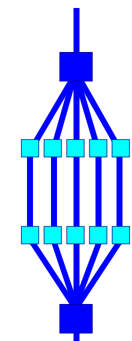


- *Use the II value to judge the quality of the instruction scheduling*
- *Care should be taken when considering the II value:*
  - *It is a theoretical (static) estimate by the compiler*
  - *Memory is assumed to be "close by"*
  - *Other factors, like a TLB miss, are not taken into account*
- *One should not expect to measure a performance based on the II value across all memory footprints*

***) The II value is also called "Steady-State Cycle Count"**

 Copyright©2003 Sun Microsystems

Modulo Scheduling at work



- ***The Modulo Scheduler tries to:***
  - ***Exploit the superscalar architecture***
  - ***Hide the instruction latencies:***

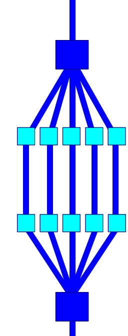
```
for (i=0; i<n; i++)
  sum += x[i]*y[i];
```

```
for (i=0; i<n; i+=4){
  sum0 += x[i  ]*y[i  ];
  sum1 += x[i+1]*y[i+1];
  sum2 += x[i+2]*y[i+2];
  sum3 += x[i+3]*y[i+3];
}
sum = sum0+sum1+sum2+sum3;
```

Note that this works best with pipelined instructions

Copyright©2003 Sun Microsystems

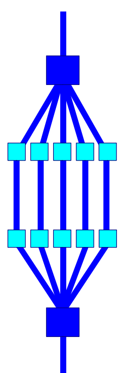
Example Latency Hiding



Fragment from an assembly listing (US-II):

Initiate	Finish	Instruction
18	26	ld [%o2],%f24
18	19	add %o0,5,%o0
18	19	add %o2,20,%o2
18	21	fmuls %f8,%f16,%f22
19	19	cmp %o0,%o4
19	27	ld [%o1],%f8
19	20	add %o1,20,%o1
19	22	fadds %f21,%f18,%f18
20	23	fmuls %f6,%f14,%f21
20	28	ld [%o2-16],%f16
21	29	ld [%o1-16],%f6
21	24	fadds %f20,%f22,%f22

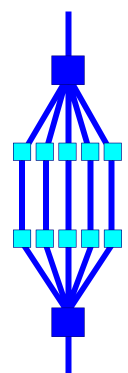
Copyright©2003 Sun Microsystems



Loop Based Optimizations

 Copyright©2003 Sun Microsystems

Cache Line Utilization

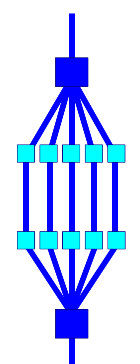


Two Key Rules - Maximize

- ***Spatial Locality - Use all data in one cache line***
  - ✔ ***This strongly depends on the storage of your data and the access pattern(s)***
- ***Temporal Locality - Re-use data in a cache line***
  - ✔ ***This mainly depends on the algorithm used***

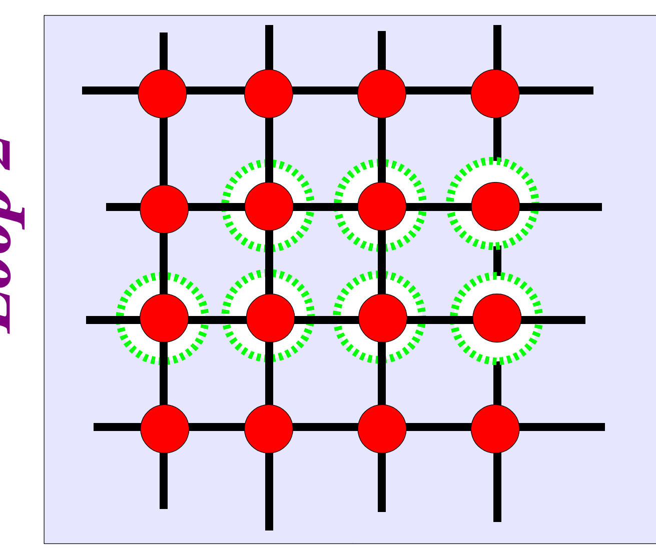
Copyright©2003 Sun Microsystems

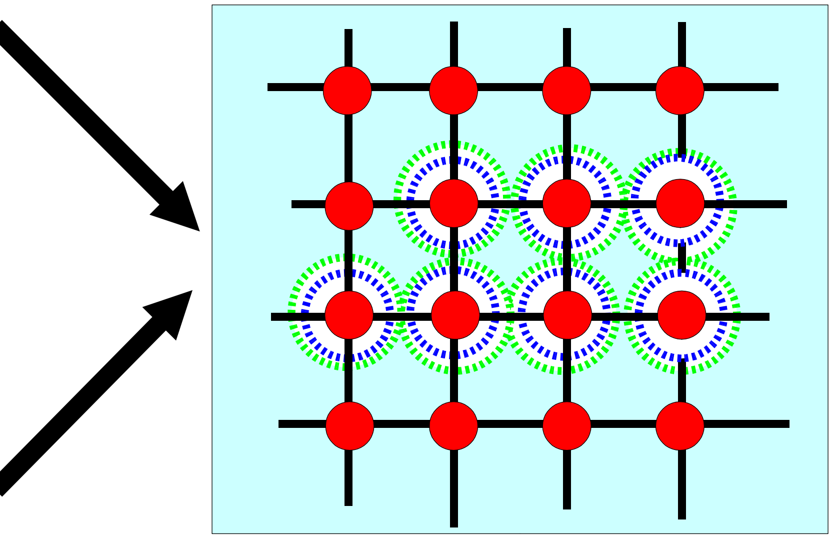
Cache line re-use



Loop 1

- *On the left we show a typical 'vector' style of coding*
- *It is not a good approach for cache based systems: all grid elements have to be reloaded for each loop*

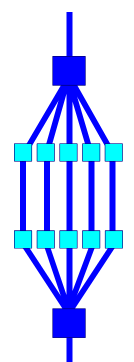




- *It is more beneficial to (pre-) calculate expressions on the already loaded grid points*

Copyright©2003 Sun Microsystems

Loop Interchange



```
for (j=0; j<n; j++)
    for (i=0; i<m; i++)
        a[i][j]=b[i][j]+c[i][j];
```

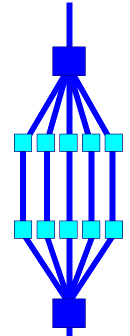
Interchange loops

```
for (i=0; i<m; i++)
    for (j=0; j<n; j++)
        a[i][j]=b[i][j]+c[i][j];
```

- ***All 3 matrices are accessed over the columns first***
- ***In C, this is the wrong access order***
- ***Interchanging the loops will solve the problem***
- ***In Fortran, the situation is reversed:***
  - *column access is okay*
  - *row access is bad*

Copyright©2003 Sun Microsystems

Compiler Output

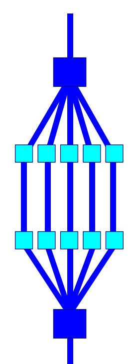


Options: -fast -xdepend -xrestrict

```
Loop below interchanged with loop on line 6
Loop below pipelined with steady-state cycle
count = 2 before unrolling
Loop below unrolled 4 times
Loop below has 2 loads, 1 stores, 3 prefetches,
1 FPadds, 0 FPmuls, and 0 FPdivs per iteration
 5.      for (j=0; j<n; j++)

Loop below interchanged with loop on line 5
 6.         for (i=0; i<m; i++)
 7.            a[i][j] = b[i][j] + c[i][j];
```

 Copyright©2003 Sun Microsystems

Loop Fission - Example



```
for (j=0; j<n; j++)
{
   c[j] = exp(j/n);
   for (i=0; i<m; i++)
      a[i][j]=b[i][j]+d[i]*e[j];
}
```

- *Access on arrays 'a' and 'b' is bad*
- *We can not simply interchange the loops*
- *Fission/splitting is the solution*

Fission

```
for (j=0; j<n; j++)
   c[j] = exp(j/n);
```

New loop created

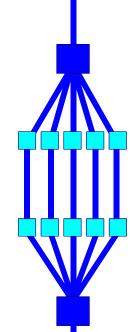
This loop can now also be vectorized

```
for (j=0; j<n; j++)
   for (i=0; i<m; i++)
      a[i][j]=b[i][j]+d[i]*e[j];
```

Interchange loops for better performance

Copyright©2003 Sun Microsystems

Compiler Output

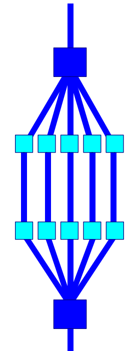


Options: -fast -xdepend -xrestrict -xvector

```
Loop below fissioned into 2 loops
Loop below interchanged with loop on line 11
Loop below strip-mined
Loop below transformed to use calls to vector intrinsic __vexp_
Loop below pipelined with steady-state cycle count = 3 before
unrolling
Loop below unrolled 4 times
Loop below has 2 loads, 1 stores, 0 prefetches, 1 FPadds, 1 FPmuls,
and 0 FPdivs per iteration
     8.     for (j=0; j<n; j++)
     9.     {
    10.        c[j] = exp(j/n);

Loop below interchanged with loop on line 8
    11.        for (i=0; i<m; i++)
    12.           a[i][j] = b[i][j] + d[i]*e[j];
    13.     }
```

 Copyright©2003 Sun Microsystems

Loop Fusion - Example



```
for (i=0; i<n; i++)
    a[i] = 2 * b[i];
```

```
for (i=0; i<n; i++)
    c[i] = a[i] + d[i];
```

- *Assume that 'n' is large*
- *In the second loop, a[i] will no longer be in the cache*
- *Fusing the loops will ensure a[i] is still in the cache when needed*

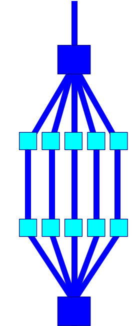
```
for (i=0; i<n; i++)
{
    a[i] = 2 * b[i];
    c[i] = a[i] + d[i];
}
```

Note that it is possible to apply fusion to loops with (slightly) different boundaries

In such a case, some iterations will have to be 'peeled' off

Copyright©2003 Sun Microsystems

Compiler Output

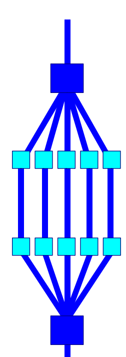


Options: -fast -xdepend -xrestrict

```
Loop below fused with loop on line 8
Loop below pipelined with steady-state cycle count = 2
before unrolling
Loop below unrolled 8 times
Loop below has 2 loads, 2 stores, 8 prefetches, 1 FPadds,
1 FPmuls, and 0 FPdivs per iteration
  6.      for (i=0; i<n; i++)
  7.         a[i] = 2 * b[i];

Loop below fused with loop on line 6
  8.      for (i=0; i<n; i++)
  9.         c[i] = a[i] + d[i];
```


Copyright©2003 Sun Microsystems

Loop Fission and Fusion



```
for(i=0;i<n;i++)
{
   ... [i] ...
}
```

Fission

```
for(i1=0;i1<n;i1++)
{
   ... [i1] ...
}
```

```
for(i2=0;i2<n;i2++)
{
   ... [i2] ...
}
```

Fusion

```
for(i=0;i<n;i++)
{
   ... [i] ...
}
```

Fission

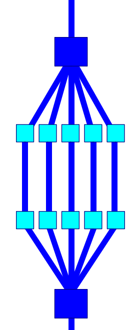
- ✔ *Reduce register pressure*
- ✔ *Enable loop interchange*
- ✔ *Isolate dependencies*
- ✔ *Increase opportunities for optimization (e.g. vectorization of intrinsics)*

Fusion

- ✔ *Reduce cache reloads*
- ✔ *Increase Instruction Level Parallelism (ILP)*
- ✔ *Reduce loop overhead*

Copyright©2003 Sun Microsystems

Inner Loop Unrolling - Example



Through unrolling, the loop overhead ('book keeping') is reduced

```
for (i=0; i<n; i++)
   a[i] = b[i] + c[i];
```

```
Addresses  : 3
Loads      : 2
Stores     : 1
FP Adds    : 1
I=I+1
Test I < N ?
Branch
Addr. incr: 3
```

Work: 4
Overhead: 9

Loop is unrolled with a factor of 4

```
for (i=0; i<n; i+=4)
{
   a[i  ] = b[i  ] + c[i  ];
   a[i+1] = b[i+1] + c[i+1];
   a[i+2] = b[i+2] + c[i+2];
   a[i+3] = b[i+3] + c[i+3];
}
<clean-up loop>
```

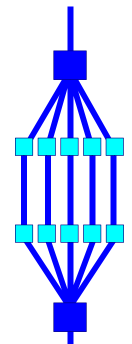
```
Addresses  : 3
Loads      : 8
Stores     : 4
FP Adds    : 4
I=I+4
Test I < N ?
Branch
Addr. incr: 3
```

Work: 16
Overhead: 9

Note: the amount of addressing needed in reality is less

Copyright©2003 Sun Microsystems

Compiler Output

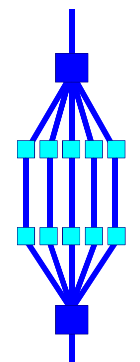


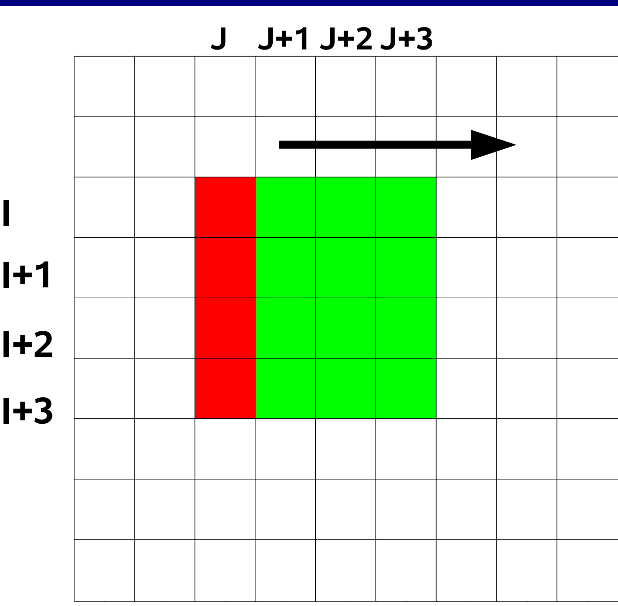
Options: -fast -xdepend -xrestrict

```
Loop below pipelined with steady-state cycle count = 2
before unrolling
Loop below unrolled 4 times
Loop below has 2 loads, 1 stores, 3 prefetches, 1 FPadds,
0 FPmuls, and 0 FPdivs per iteration
  6.      for (i=0; i<n; i++)
  7.          a[i] =  b[i] + c[i];
```


Copyright©2003 Sun Microsystems

Outer Loop Unrolling - Example



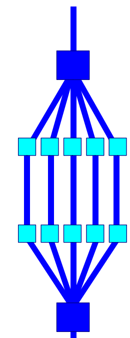



```
for (i=0; i<m; i+=4)
  for(j=0; j<n; j++)
  {
    a[i  ] += b[i  ][j] * c[j];
    a[i+1] += b[i+1][j] * c[j];
    a[i+2] += b[i+2][j] * c[j];
    a[i+3] += b[i+3][j] * c[j];
  }
<clean-up loop>
```

- ***Advantage:***
  - *c[j] is re-used 3 more times* ***(temporal locality)***
- ***Deeper unrolling, say 8, requires more fp registers (17 instead of 9), but improves re-use of c[j]***

Copyright©2003 Sun Microsystems

Compiler Output



Options: -fast -xdepend -xrestrict

```
Loop below unrolled and jammed
 5.     for (i=0; i<m; i++)

Loop below unrolled and jammed
Loop below pipelined with steady-state cycle count = 2
before unrolling
Loop below unrolled 8 times
Loop below has 2 loads, 0 stores, 4 prefetches,
1 FPadds, 1 FPmuls, and 0 FPdivs per iteration
 6.        for (j=0; j<n; j++)

Loop below pipelined with steady-state cycle count = 9
before unrolling
Loop below unrolled 4 times
Loop below has 9 loads, 0 stores, 8 prefetches,
8 FPadds, 8 FPmuls, and 0 FPdivs per iteration
 7.            a[i] += b[i][j]*c[j];
```

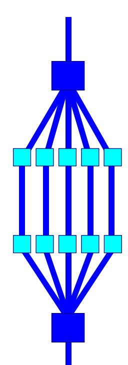
outer loop unrolling (points to line 5)

inner loop unrolling (points to "Loop below unrolled 8 times")

inner loop unrolling (points to "Loop below unrolled 4 times")

Copyright©2003 Sun Microsystems

Unroll and Jam



```
for (i=0; i<m; i++)
  for(j=0; j<n; j++)
    a[i] += b[i][j] * c[j];
```

Outer loop unrolling

```
for (i=0; i<m-m%4; i+=4)
{
  for(j=0; j<n; j++)
    a[i  ] += b[i  ][j] * c[j];
  for(j=0; j<n; j++)
    a[i+1] += b[i+1][j] * c[j];
  for(j=0; j<n; j++)
    a[i+2] += b[i+2][j] * c[j];
  for(j=0; j<n; j++)
    a[i+3] += b[i+3][j] * c[j];
}
for (i=m-m%4; i<m; i++)
  for(j=0; j<n; j++)
    a[i] += b[i][j] * c[j];
```

clean-up loop

Unroll and Jam

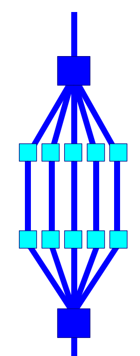
Jam the loops together again

```
for (i=0; i<m-m%4; i+=4)
  for(j=0; j<n; j++)
  {
    a[i  ] += b[i  ][j] * c[j];
    a[i+1] += b[i+1][j] * c[j];
    a[i+2] += b[i+2][j] * c[j];
    a[i+3] += b[i+3][j] * c[j];
  }
for (i=m-m%4; i<m; i++)
  for(j=0; j<n; j++)
    a[i] += b[i][j] * c[j];
```

clean-up loop

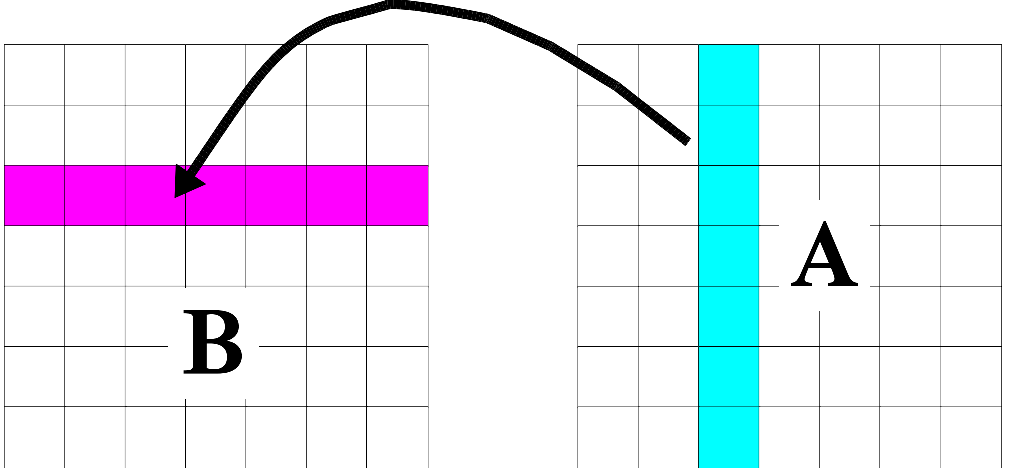
Copyright©2003 Sun Microsystems

Loop Blocking - Example



Transposing a matrix

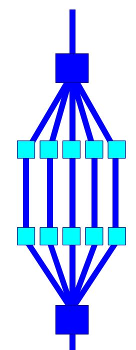
```
for (j=0; j<n; j++)
   for (i=0; i<n; i++)
      b[j][i] = a[i][j];
```



- *Loop interchange will not help here:*
  - *Role of 'a' and 'b' will only be interchanged*
- *Change of programming language won't help either*
- *Unrolling the i-loop can be beneficial, but requires more registers and doesn't address TLB-misses*
- *Loop blocking achieves good memory performance, without the need for additional registers*

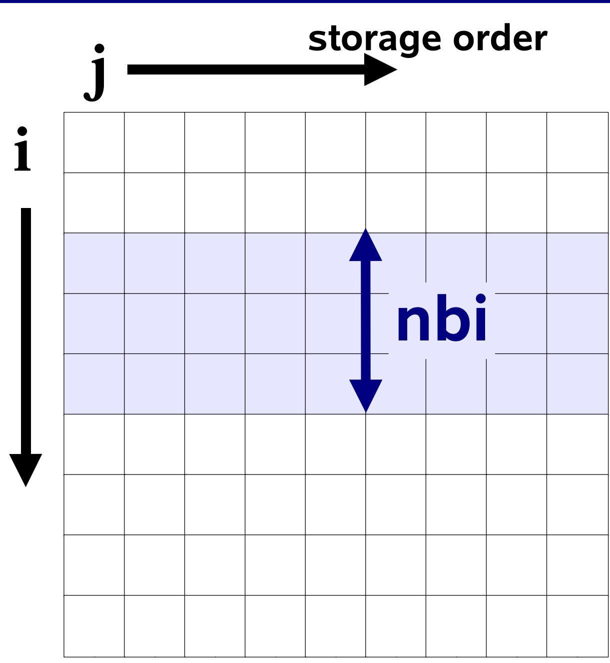
Copyright©2003 Sun Microsystems

Loop Blocking - Example

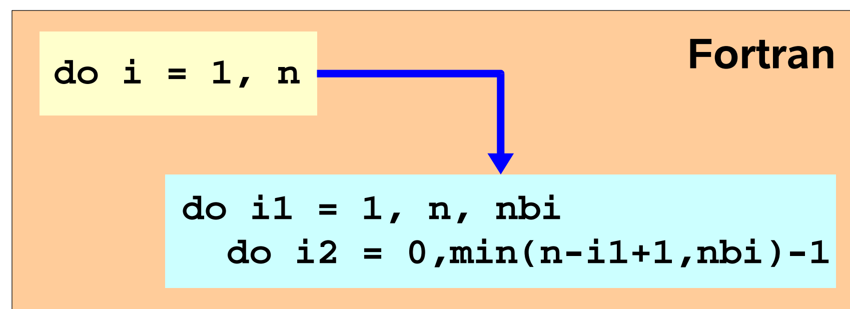


Blocking and interchanging the I-loop

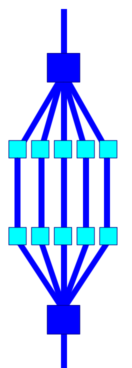
```
for(i1=0; i1<n; i1+=nbi)
  for (j=0; j<n; j++)
     for (i2=0;i2<MIN(n-i1,nbi);i2++)
        b[j][i1+i2] = a[i1+i2][j];
```



- *Parameter 'nbi' is the blocking size*
- *Should be chosen as large as possible*
- *Actual value depends on the cache to block for:*
  - ✓ *L1-cache*
  - ✓ *L2-cache*
  - ✓ *TLB*
  - ✓ ....



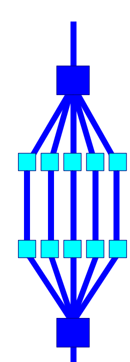
Copyright©2003 Sun Microsystems



Introduction Into Parallelization

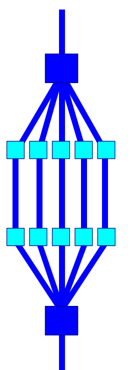
 Copyright©2003 Sun Microsystems

What is parallelization ?



- ***Parallelization is simply another optimization technique to get your results sooner***
- ***To this end, more than one processor is used to solve the problem***
- ***The "elapsed time" (also called wallclock time) will come down, but the total CPU time will probably go up***
- ***The latter is a difference with serial optimization, where one makes better use of existing resources i.e. the cost will come down***

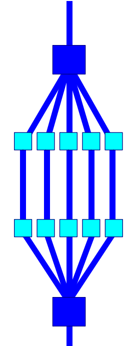
Copyright©2003 Sun Microsystems



The Name Of The Game

Copyright©2003 Sun Microsystems

What is parallelization ?

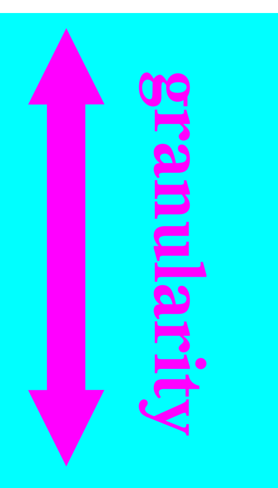


An attempt to give you a sort of definition:

> *"Something" is parallel if there is a certain level of independence in the order of operations*

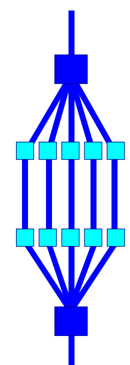
"Something" can be:

- *A collection of program statements*
- *An algorithm*
- *A part of your program*
- *The problem you're trying to solve*

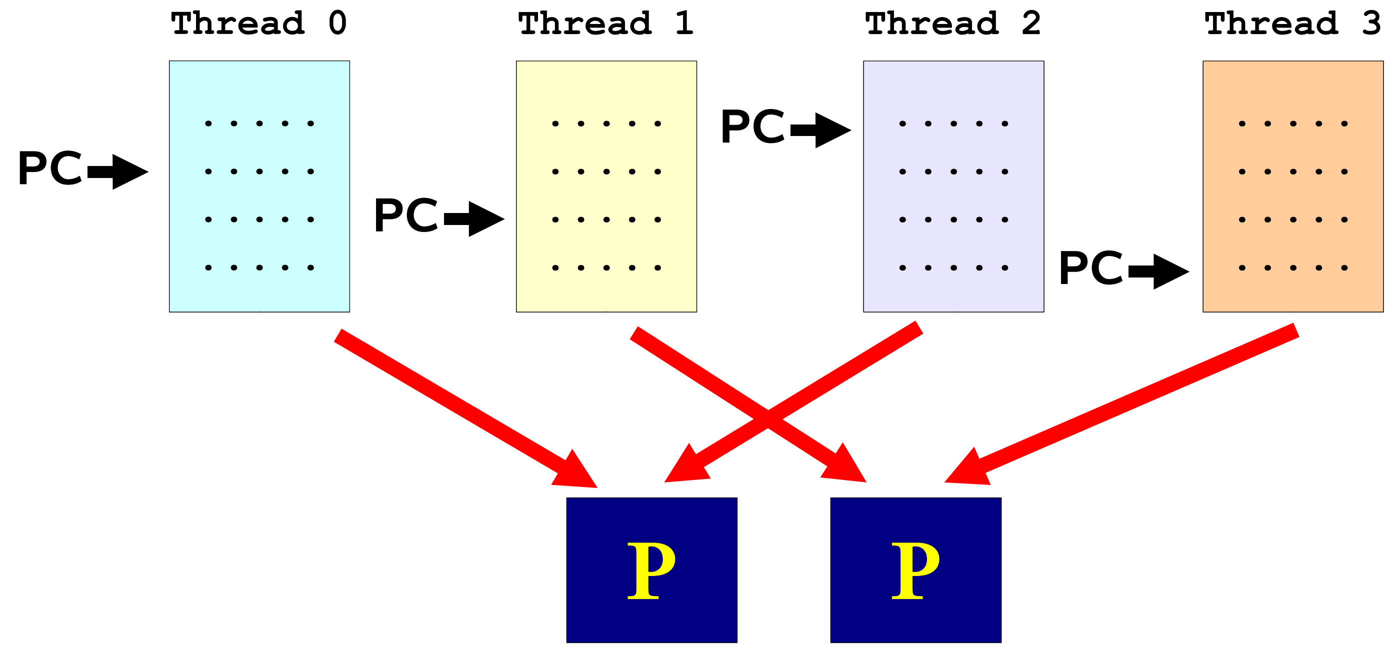


Copyright©2003 Sun Microsystems

What is a thread ?

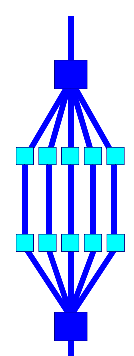


- *Loosely said, a thread consists of a series of instructions with it's own program counter (PC) and state*
- *A parallel program will execute threads in parallel*
- *These threads are then scheduled onto processors*



Copyright©2003 Sun Microsystems

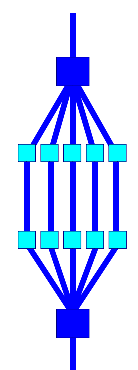
Parallel Overhead



- *The total CPU time may exceed the serial CPU time:*
  - *The newly introduced parallel portions in your program need to be executed*
  - *Processors need time sending data to each other and synchronizing ("communication")*
    - ✔ *Often the key contributor, spoiling all the fun*
- *Typically, things also get worse when increasing the number of processors*
- *Efficient parallelization is about minimizing the communication overhead*

Copyright©2003 Sun Microsystems

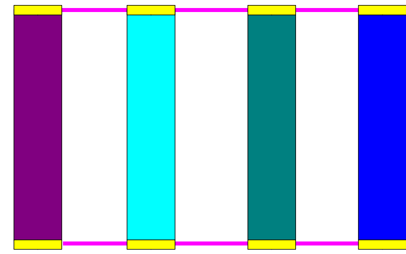
Communication



Wallclock time

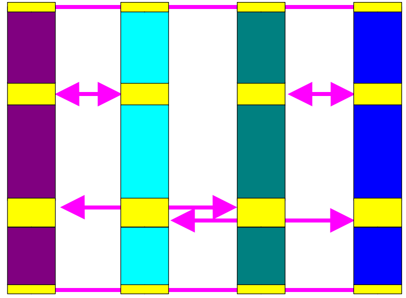
Serial Execution

Parallel - Without communication



- Embarrassingly parallel: 4x faster
- Wallclock time is ¼ of serial wallclock time

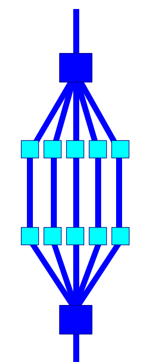
Parallel - With communication

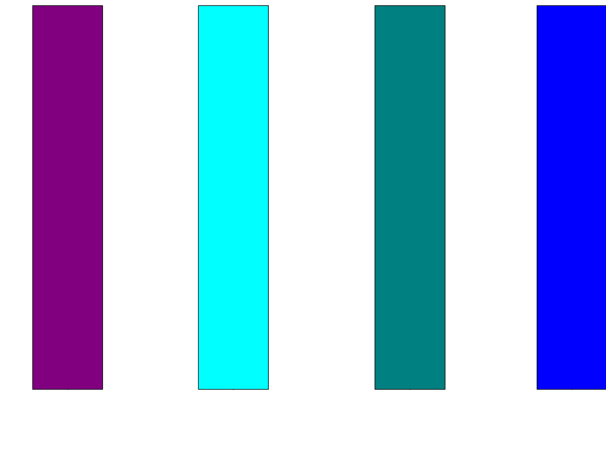


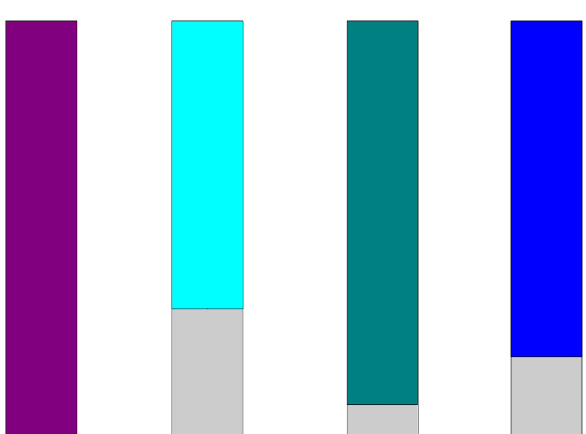
- Additional communication
- Less than 4x faster
- Consumes additional resources
- Wallclock time is <u>more than</u> ¼ of serial wallclock time
- Total CPU time increases

Copyright©2003 Sun Microsystems

Load balancing







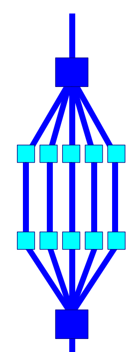
- All CPUs finish in the same time
- No CPU is idle

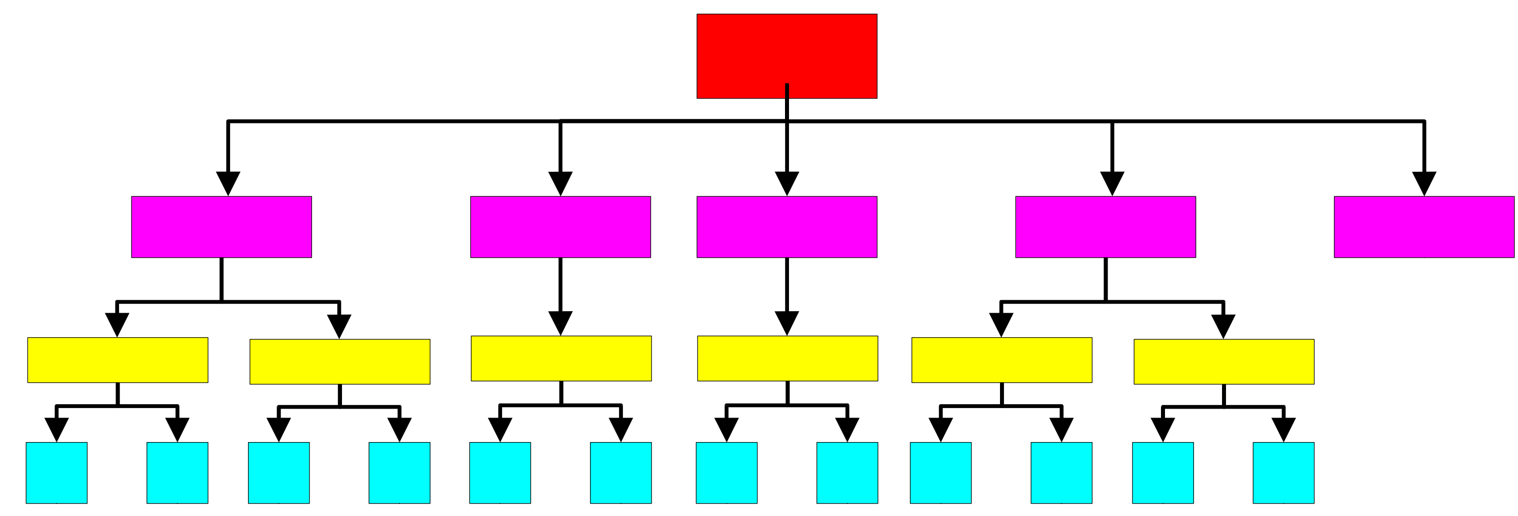
- Different CPUs need a different amount of time to finish their task
- Total wall clock time increases
- Program will not scale well

CPU is idle

Copyright©2003 Sun Microsystems

Dilemma

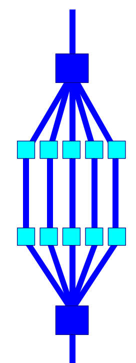


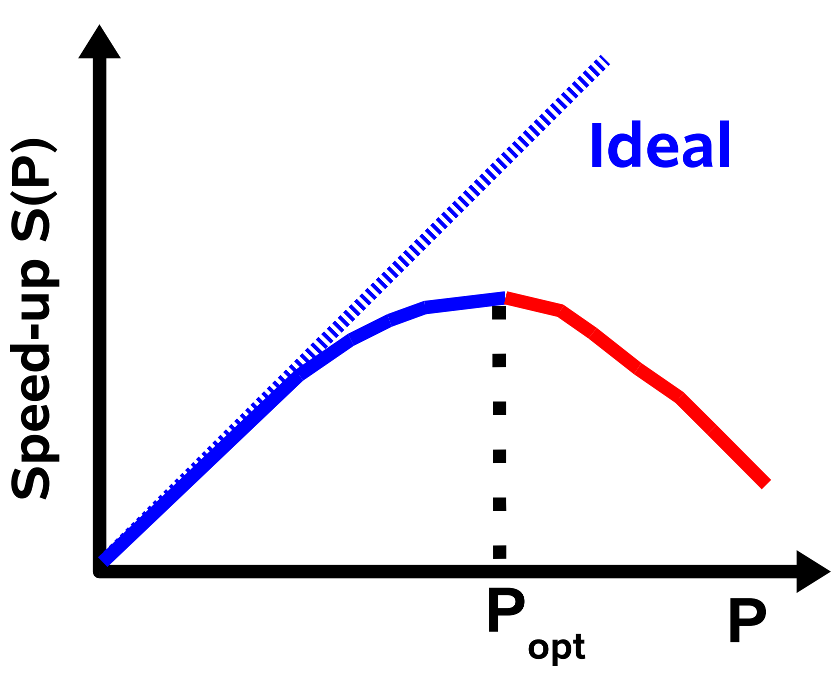


- *Parallelization at the highest ( ) level:*
  - *Low communication cost*
  - *Limited to 5 processors only*
  - *Potential load balancing issue*
- *Parallelization at the lowest ( ) level:*
  - *Higher communication cost*
  - *Not limited to a certain number of processors*
  - *Load balancing probably less of an issue*

 Copyright©2003 Sun Microsystems

About scalability





In some cases, S(P) will exceed P

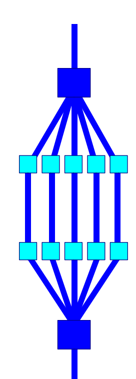
This is called "superlinear" behaviour

Don't count on this to happen though

- *Define the speed-up S(P) as $S(P) := T(1)/T(P)$*
- *The efficiency E(P) is defined as $E(P) := S(P)/P$*
- *In the ideal case, $S(P)=P$ and $E(P)=100\%$*
- *Unless the application is embarrassingly parallel, S(P) will start to deviate from the ideal curve*
- *Past this point P_{opt}, the application will get less and less benefit from adding processors*
- *Note that both metrics give no information on the actual run-time*
- *As such, they can be dangerous to use*

 Copyright©2003 Sun Microsystems

Amdahl's Law



Assume our program has a parallel fraction "f"

This implies the execution time $T := f*T + (1-f)*T$

On P processors: $T(P) = (f/P)*T + (1-f)*T$

Amdahl's law:

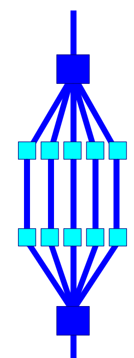
$$S(P) := T / T(P) = 1 / (f/P + 1-f)$$

Comments:

- *This "law' describes the effect that the non-parallelizable part of a program has on scalability*
- *Note that the additional overhead caused by parallelization and speed-up because of cache effects are not taken into account*

Copyright©2003 Sun Microsystems

Amdahl's law



Speed-up

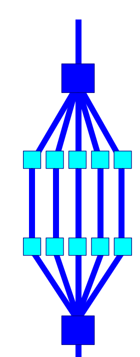
Processors

Value for f:
1.00
0.99
0.75
0.50
0.25
0.00

- ***It is easy to scale on a small number of processors***
- ***Scalable performance however requires a high degree of parallelization i.e. f is very close to 1***
- ***This implies that you need to parallelize that part of the code where the majority of the time is spent***
- ***Use the performance analyzer to find these parts***

Copyright©2003 Sun Microsystems

Amdahl's Law In Practice



We can estimate the parallel fraction "f"

Recall: $T(P) = (f/P)*T + (1-f)*T$

It is trivial to solve this equation for "f":

$$f = (1 - T(P)/T)/(1 - (1/P))$$

Example:

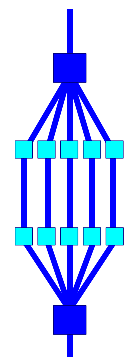
```
T = 100 and T(4)=37 => S(4) = T/T(4) = 2.70
f = (1-37/100)/(1-(1/4)) = 0.63/0.75 = 0.84 = 84%
```

Estimated performance on 8 processors is then:

```
T(8) = (0.84/8)*100 + (1-0.84)*100 = 26.5
S(8) = T/T(8) = 3.78
```


Copyright©2003 Sun Microsystems

Numerical Results



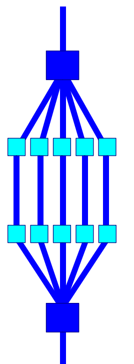
Consider:

A = B + C + D + E

Serial Processing	*Parallel Processing* CPU 1	CPU 2
`A = B + C`	`T1 = B + C`	`T2 = D + E`
`A = A + D`	`T1 = T1 + T2`	
`A = A + E`		

- *The roundoff behaviour is different and so the numerical results may be different too*
- *This is natural for parallel programs, but it may be hard to differentiate it from an ordinary bug ....*

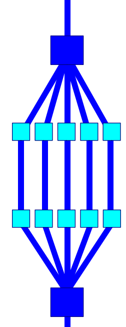
Copyright©2003 Sun Microsystems



Cache Coherency

Copyright©2003 Sun Microsystems

Cache line modifications



How to handle the problem of multiple copies of a line ?

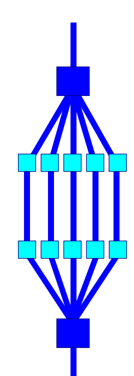
modifications

Caches

Main Memory

 Copyright©2003 Sun Microsystems

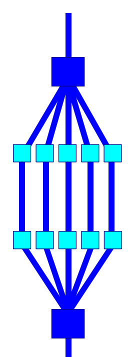
Cache Organization



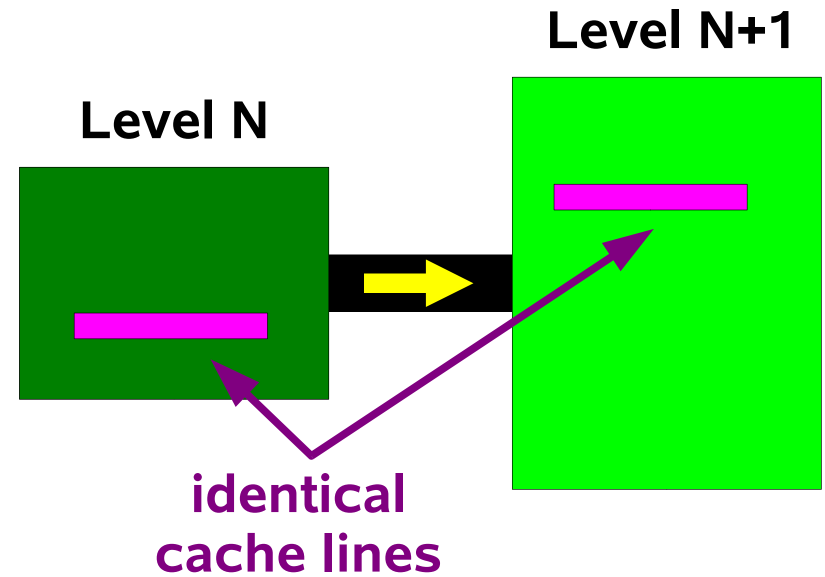
- ❑ *A cache contains a partial image of memory*
- ❑ *If data gets modified, the state of that data changes*
- ❑ *This has to be made known to the system*
- ❑ *Two popular approaches are:*
  - *Write-through*
  - *Write-back*

Copyright©2003 Sun Microsystems

Write-Through



- *Always flush a modified cache line back to a higher level in the memory hierarchy*
  - *For example, write a modified line back from the L1 cache to main memory*
- *In this way, the system will always know where to get the correct cache line from*

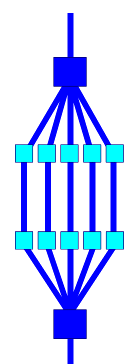


Comments:

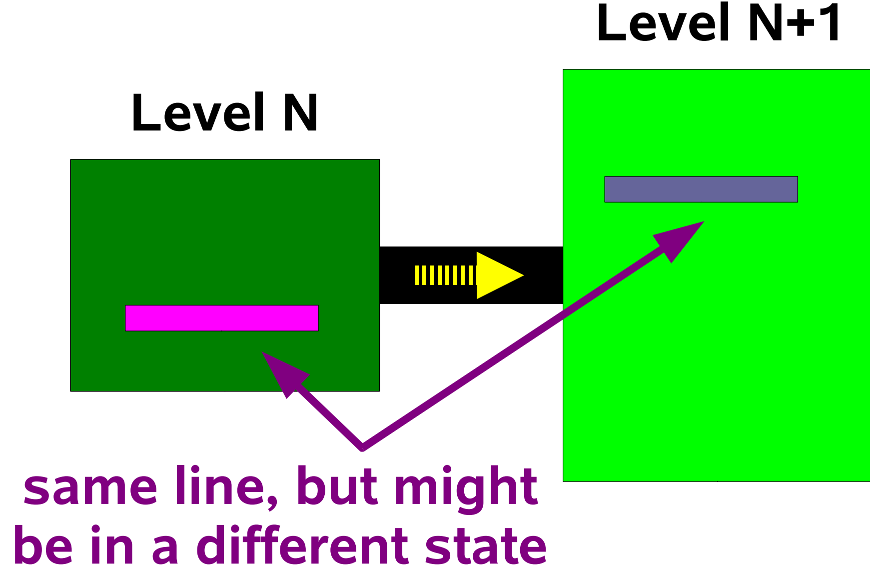
- *Relatively simple to implement*
- *Easy to find the right copy*
- *May waste bandwidth though*

Copyright©2003 Sun Microsystems

Write-Back



- ❑ *Only write a modified cache line back if needed*
  - *Capacity issue*
  - *Other cache line maps onto existing line*
  - *Other CPU needs this cache line*

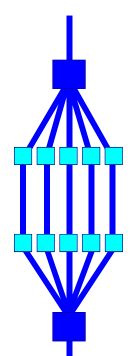


Comments:

- *Minimizes cache traffic*
- *Need to keep track of status though*
- *The mechanism to do this is called cache coherency*

Copyright©2003 Sun Microsystems

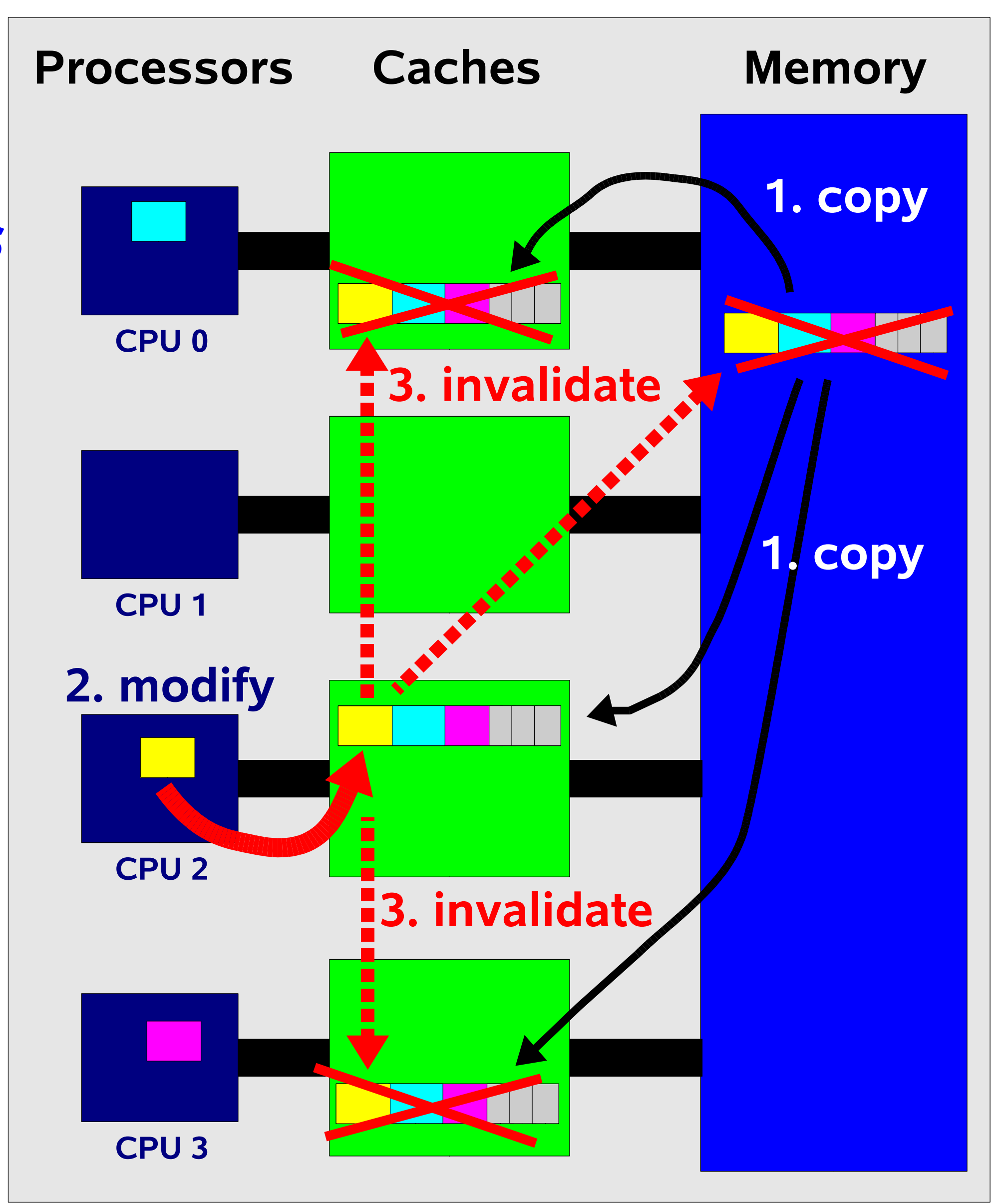
Caches in an MP system



- A cache line starts in memory
- Over time multiple copies of this line may exist

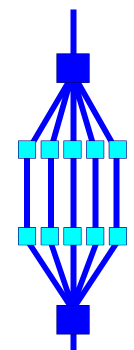
Cache Coherency ("cc"):

- ✔ Tracks changes in copies
- ✔ Makes sure correct cache line is used
- ✔ Different implementations possible
- ✔ Need hardware to make it efficient

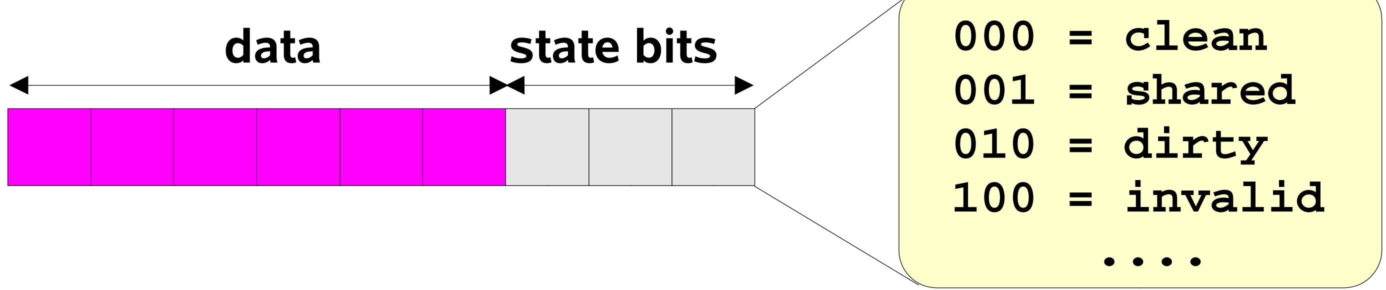


Copyright©2003 Sun Microsystems

Cache Coherency ("cc")



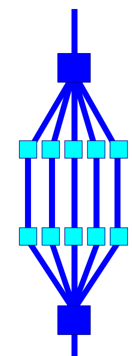
- Needed in a write-back cache organization
- Keeps track of the status of cache lines
- This is called the "state" information



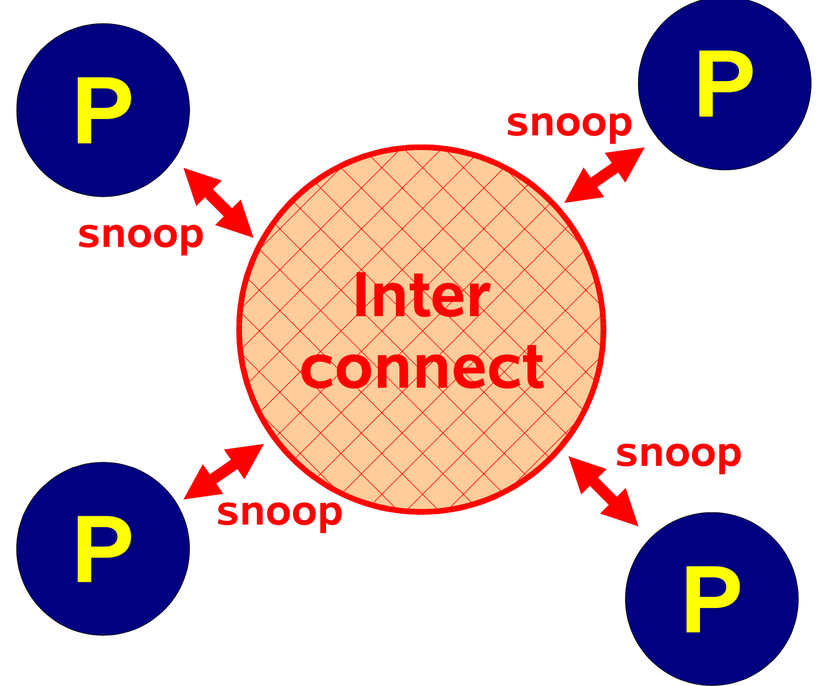
- The system uses signals ("coherency traffic") to update the status bits of cache lines
- Cache Coherency is a very convenient feature to have
- It makes it possible to build efficient shared memory parallel systems

 Copyright©2003 Sun Microsystems

Snoopy based cache coherence

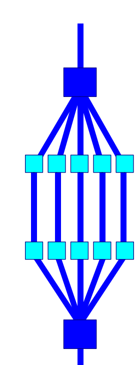


- Also called "Broadcast" cache coherence
  - All addresses sent to all devices
  - Result of the snoop is computed in a few cycles
- Advantages:
  - Low latency in general
  - Fast cache-to-cache transfers
- Disadvantage
  - Data bandwidth limited by snoop bandwidth
  - Difficult to scale to a large number of processors



Copyright©2003 Sun Microsystems

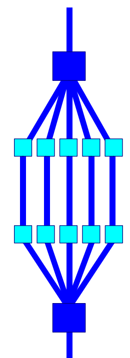
Directory based cache coherence



- *Also called SSM (Scalable Shared Memory)*
- *This is a point-to-point protocol*
- *Through a directory, the system keeps track which processor(s) are involved in a particular line*
- *Address requests sent to specific caches only*
- *Advantages:*
  - *Bandwidth can be much greater*
  - *Scalable to large processor count*
- *Disadvantages*
  - *Latency is usually longer and no longer uniform*
  - *Slower cache-to-cache transfers*
  - *Need to store the additional directory entries*

Copyright©2003 Sun Microsystems

Example SSM



data | state bits | bit vector

0 1 1 0 0 1 0 1

modified

P0 P1 P2 P3 P4 P5 P6 P7

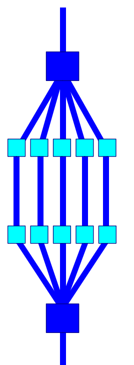
Inter connect

invalidate

invalidate

invalidate

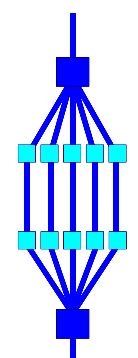
Copyright©2003 Sun Microsystems

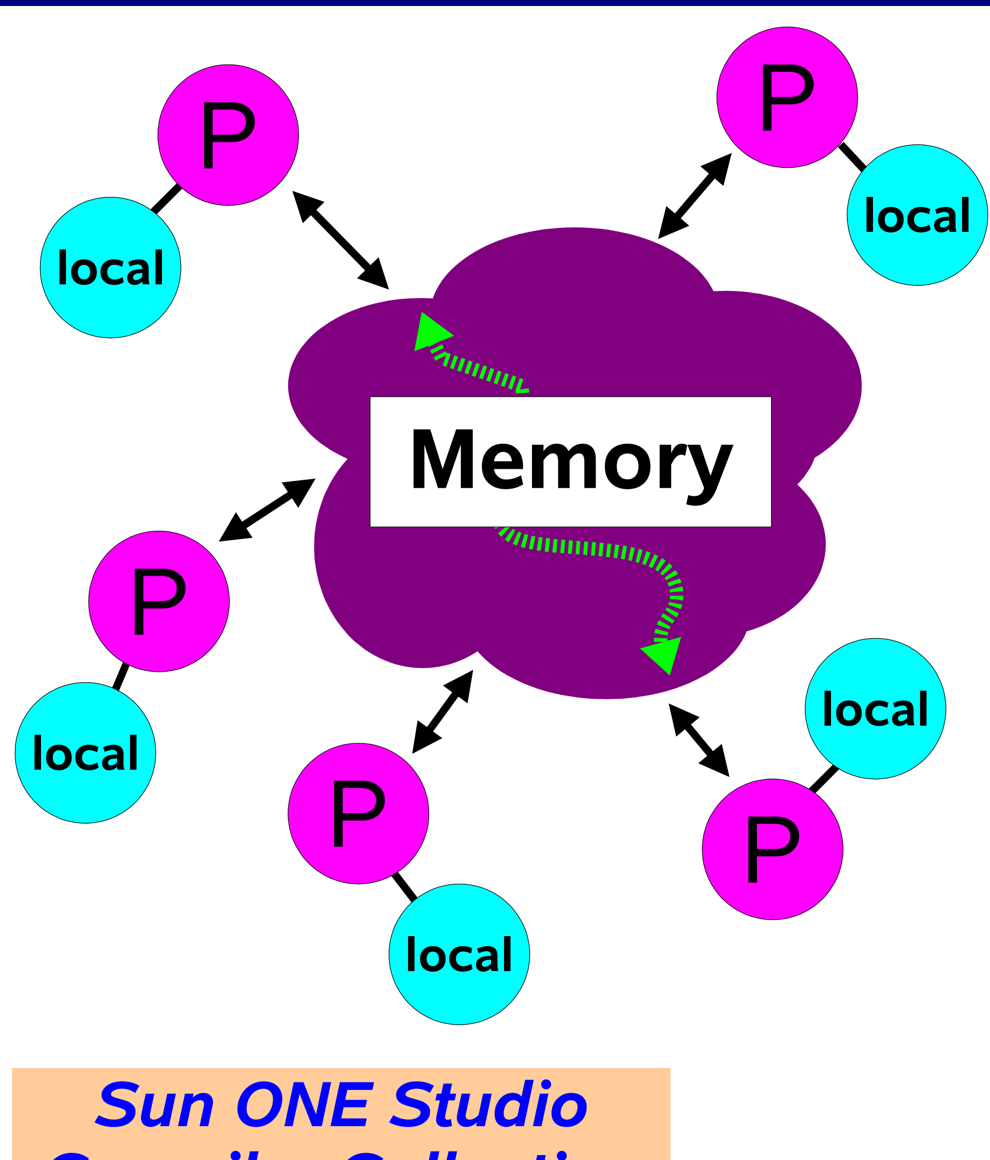


Programming Models

 Copyright©2003 Sun Microsystems

Shared Memory Model





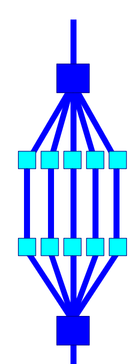
Programming Model

- ✔ All processors have access to the same, global, memory
- ✔ Data transfer is transparent to the programmer
- ✔ Synchronization takes place, but it is mostly implicit
- ✔ By default, data is shared (but one needs local data too)

This programming model makes it relatively (!) easy to develop auto-parallelizing compilers

Copyright©2003 Sun Microsystems

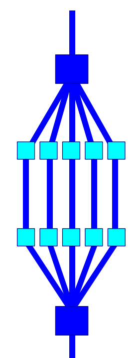
About data

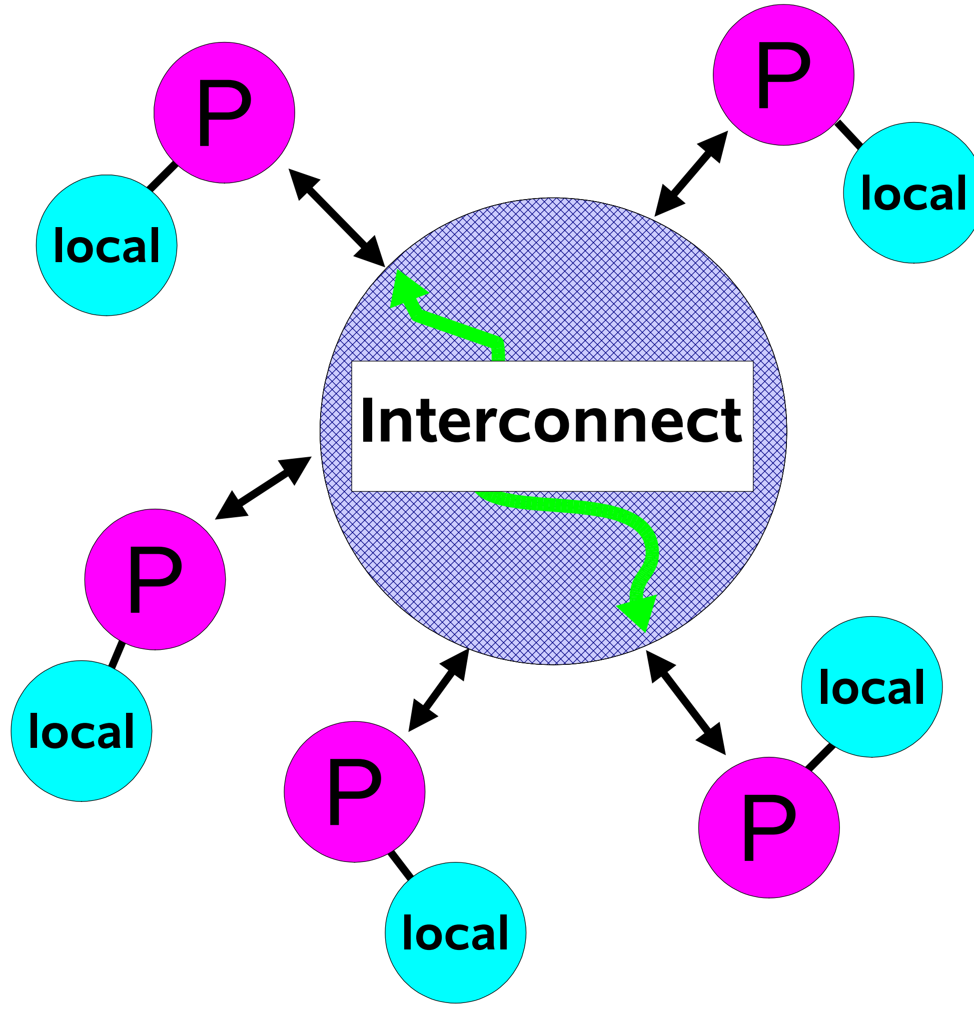


- ***You may not have realized this before, but in a shared memory parallel program your variables have a "label" attached to them:***
  - ***Labelled "Private" ⇨ Visible to you only***
    - ✔ ***Change made in local data, is not seen by others***
    - ✔ ***Example - Local variables in a function that is executed in parallel***
  - ***Labelled "Shared" ⇨ Visible to others***
    - ✔ ***Change made in global data, is seen by all others***
    - ✔ ***Example - Global data***

Copyright©2003 Sun Microsystems

Distributed Memory Model





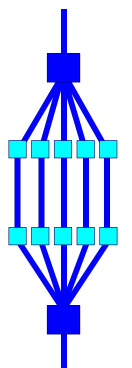
Programming Model

- *All processors have access to their own, local, memory only*
- *Data transfer and most synchronization has to be programmed explicitly*
- *By default, data is local*
- *Data is shared explicitly by exchanging buffers*

This programming model makes it very hard to develop auto-parallelizing compilers

HPC Clustertools

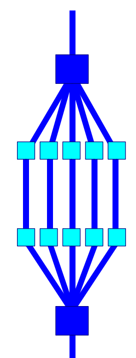
Copyright©2003 Sun Microsystems



Parallel Architectures

Copyright©2003 Sun Microsystems

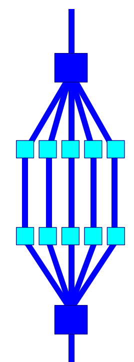
The Debate

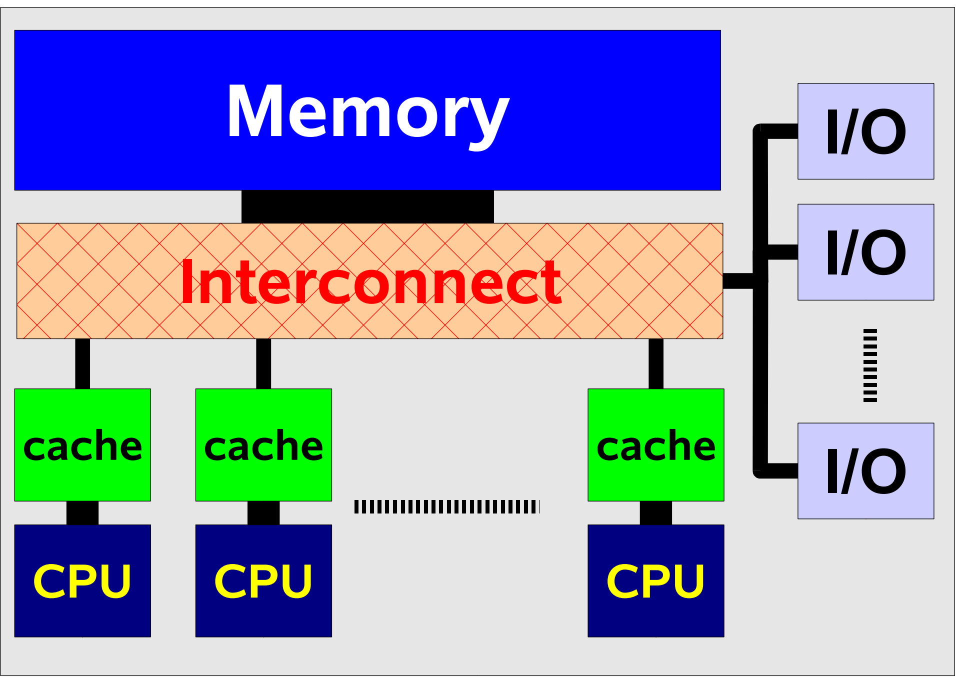


- ❑ *There is an on-going debate about labelling systems:*
  - *It is hard to classify architectures in the first place*
  - *Most systems share some characteristics, but not all*
    - ✔ *For example, when do we call a system cc-NUMA ?*
      - ◆ *Even a cache based workstation might qualify ...*
- ❑ *In the overview we're going to present, we will classify systems based on Main Memory:*
  - *Shared or Distributed ?*
    - ✔ *Can all processors access all of memory, or a subset only ?*
  - *Memory access time(s)*

Copyright©2003 Sun Microsystems

Uniform Memory Access (UMA)





Pro

- ✔ *Easy to use and to administer*
- ✔ *Efficient use of resources*

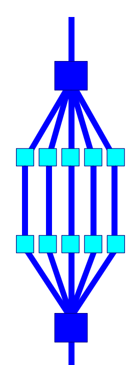
Con

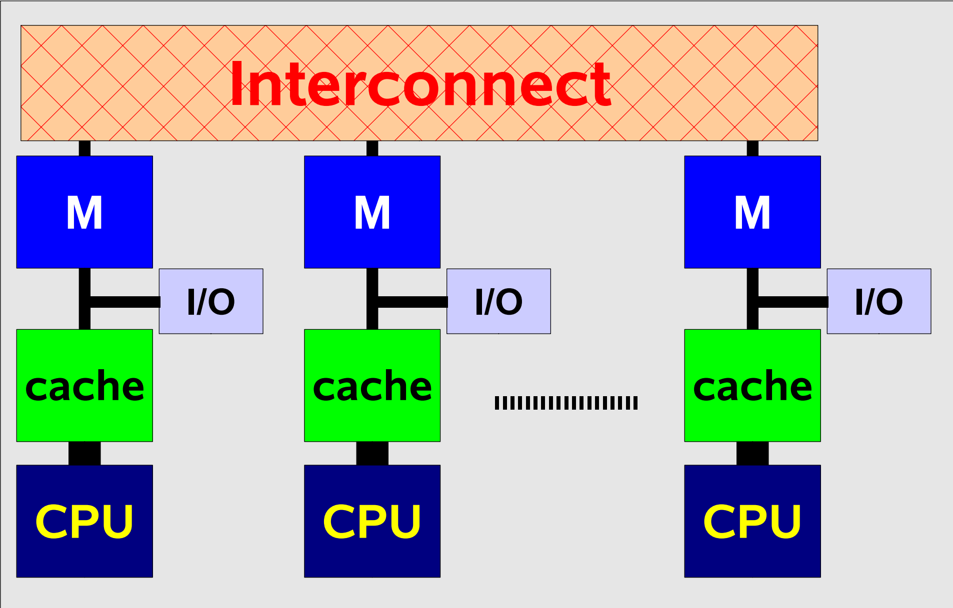
- ✔ *Said to be expensive*
- ✔ *Said to be non-scalable*

- ❑ *Also called "SMP" (Symmetric Multi Processor)*
- ❑ *Memory Access time is Uniform for all CPUs*
- ❑ *Interconnect is "cc":*
  - *Bus*
  - *Crossbar*
- ❑ *No fragmentation - Memory and I/O are shared resources*

 Copyright©2003 Sun Microsystems

NUMA





Pro

- ✔ *Said to be cheap*
- ✔ *Said to be scalable*

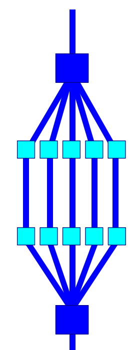
Con

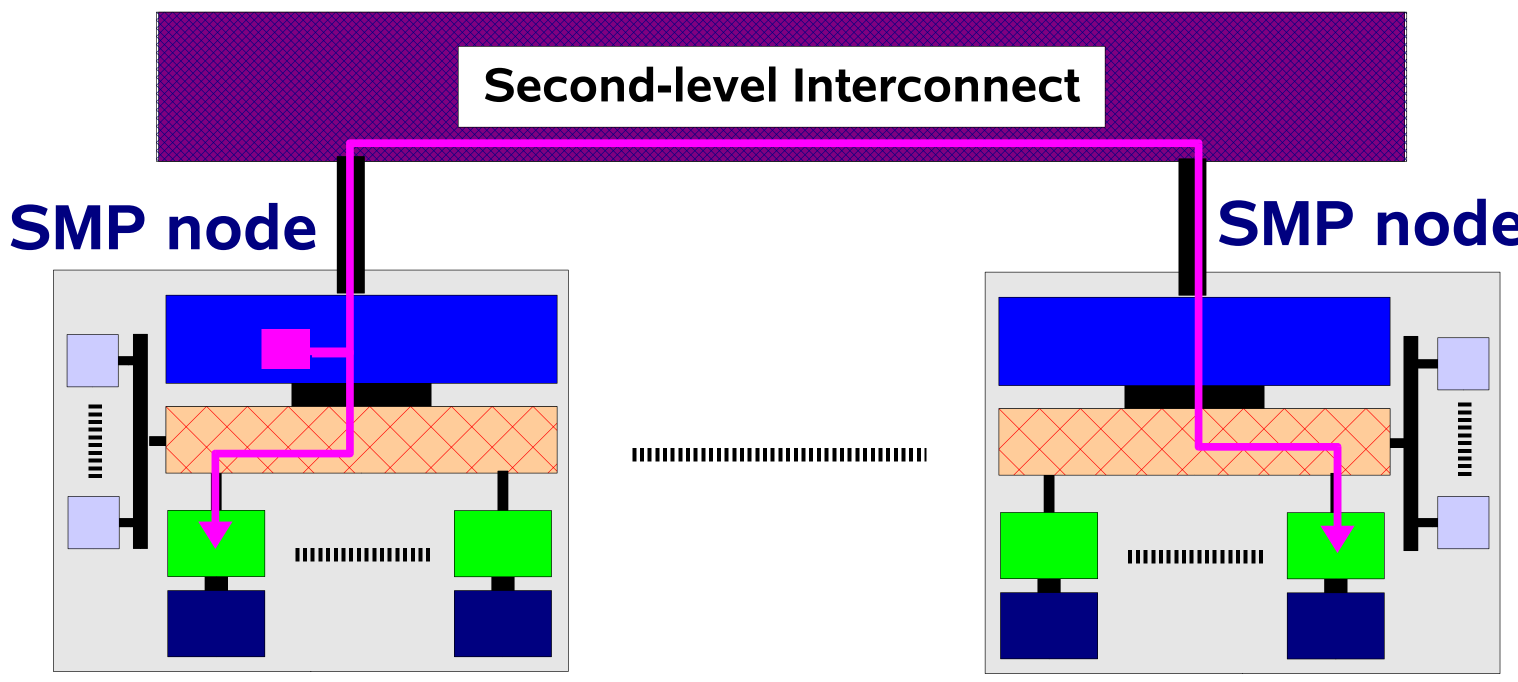
- ✔ *Difficult to use and administer*
- ✔ *In-efficient use of resources*

- ❑ *Also called "Distributed Memory"*
- ❑ *Memory Access time is Non-Uniform*
- ❑ *Hence the name "NUMA"*
- ❑ *Interconnect is not "cc":*
  - *Ethernet, ATM, Myrinet*
  - *.....*
- ❑ *Runs 'N' copies of the OS*
- ❑ *Memory and I/O are distributed resources*

Copyright©2003 Sun Microsystems

Cluster of SMP nodes

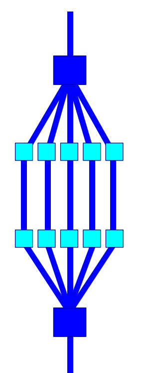


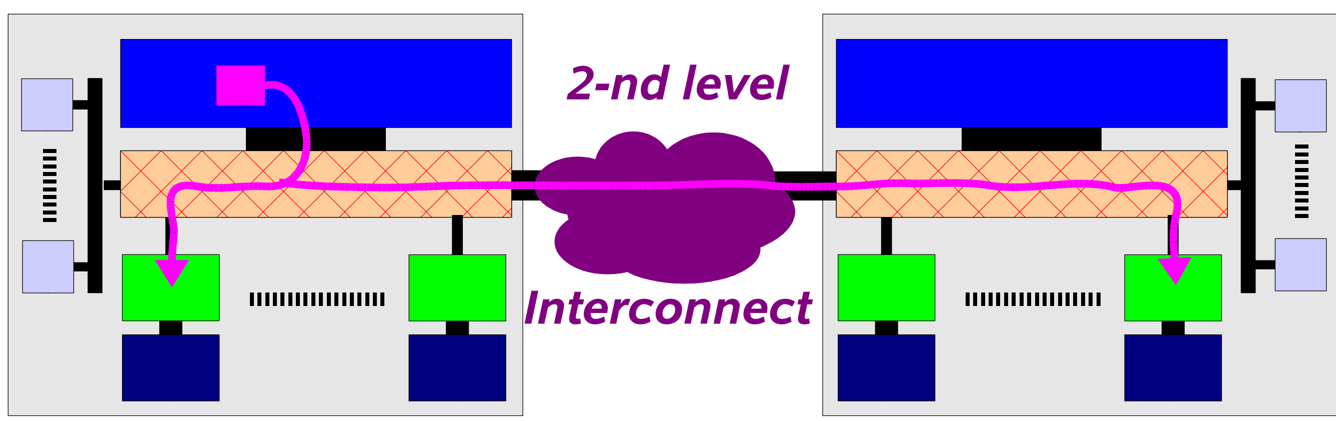


- ❑ ***Second-level interconnect is not cache coherent***
  - ***Ethernet, ATM, Myrinet, ...***
- ❑ ***Hybrid Architecture with all Pros and Cons:***
  - ***UMA within one SMP node***
  - ***NUMA across nodes***

Copyright©2003 Sun Microsystems

cc-NUMA

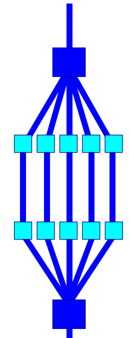




- Two-level interconnect:
  - UMA/SMP within one system
  - NUMA between the systems
- Both interconnects support cache coherency i.e. the system is fully cache coherent
- Has all the advantages ('look and feel') of an SMP
- Downside is the Non-Uniform Memory Access time

Copyright©2003 Sun Microsystems

Template Parallel Architecture *



Registers ↔ Cache

Board

P .... P

$ $

Interconnect Level 1

Memory

. . . . .

Board

P .... P

$ $

Interconnect Level 1

Memory

Cache ↔ Local Memory

Interconnect Level 2

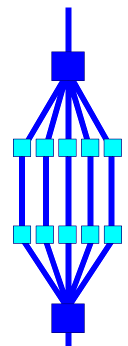
Local ↔ Remote Memory

Latency increases
Bandwidth decreases

) and simplified ....

Copyright©2003 Sun Microsystems

Programming Models Revisited



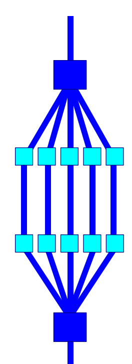
Architecture	Shared Memory Efficient ?	Distributed Memory Efficient ?
UMA/SMP	yes	yes (very !)
NUMA	no	maybe*
Cluster of SMPs	yes	maybe*
cc-NUMA	depends	yes

- ✓One can map any programming model onto any architecture
- ✓Making it efficient is the key problem to solve

*) Depends on interconnect

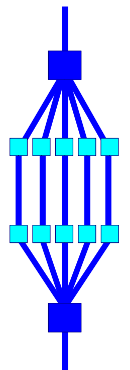
 Copyright©2003 Sun Microsystems

Parallelizing an application



- ***The question whether an application is parallel, or not, has nothing to do with the programming model***
- ***Two possibilities (for the time consuming part):***
  - ➊ ***If parallel, decide on the programming model:***
    - ☞ ***Message Passing***
      - ◆ ***Do It Yourself***
    - ☞ ***Shared Memory***
      - ◆ ***Use the compiler***
      - ◆ ***May need directives to assist the compiler***
  - ➋ ***If not parallel: Try to rewrite or change the algorithm and go back to step ➊***

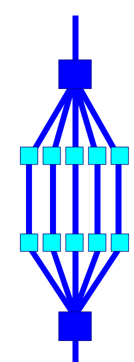
Copyright©2003 Sun Microsystems



Shared Memory Parallelization

 Copyright©2003 Sun Microsystems

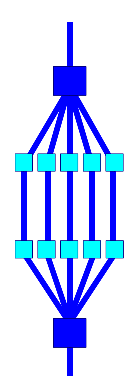
Shared Memory Programming



- ***With the Shared Memory Programming Model, one can make good use of an auto-parallelizing compiler***
- ***Success, or failure, to do so depends on:***
  - *Application area*
  - *Coding style*
  - *Quality of the compiler*
- ***One of the nice features of this programming model is the ability to "mix and match"***
  - *Get the compiler to do as much as possible*
  - *Assist the compiler through pragmas where needed*
- ***In this way, one can incrementally parallelize an application***

 Copyright©2003 Sun Microsystems

Decisions, decisions, ...



User directives or pragma's found ?

Yes

No

Analyze loop for data dependencies

Maybe

Data dependency found ?

No

Yes

No

Can the dependency be broken ?

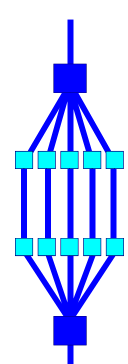
Yes

Generate serial code

Generate parallel code

Copyright©2003 Sun Microsystems

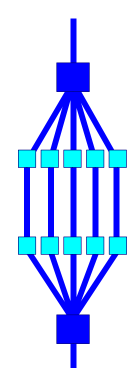
Auto-parallelization



- ***The compilers are loop oriented:***
  - ***Every (nested) loop will be analyzed for data dependencies and parallelized if safe to do so***
  - ***Non-loop code fragments will not be analyzed !***
- ***Note that one can have subtle interactions between loop transformations and parallelization***
- ***Remember that compilers have limited knowledge about the application***
- ***Check the parallelization messages with the -xloopinfo option and the er_src command***
- ***Use OpenMP directives/pragmas to override compiler behaviour***

 Copyright©2003 Sun Microsystems

Loop based parallelization



- **Loop based parallelization:**
  - *Different iterations of the loop are executed in parallel*
- **Same binary can be run on any number of processors**
- **The order in which the iterations are executed is:**
  - *Undetermined*
  - *Different from run to run*

```
for (i=0; i < N; i++)
  A[i] = B[i] + C[i];
```

CPU 1

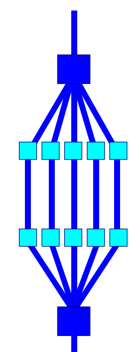
```
for (i=N/2; i < N; i++)
  A[i] = B[i] + C[i];
```

```
for (i=0; i < N/2; i++)
  A[i] = B[i] + C[i];
```

CPU 2

Copyright©2003 Sun Microsystems

Options For Parallelization



`-xautopar` *Automatic parallelization by the compiler*

(requires -xO3 or higher; includes -xdepend)

`-xreduction` *Also parallelize reductions*

(recommended to use -fsimple=2 for reductions)

`-xopenmp` *Parallelize an OpenMP application*

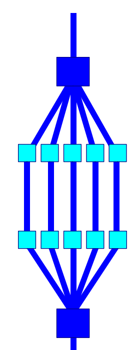
`-stackvar` *Allocate local data on stack*

(Fortran only; In C local variables are always on the stack)

`-xloopinfo` *Show parallelization messages on screen*

Copyright©2003 Sun Microsystems

Environment variables



OMP_NUM_THREADS n — ***Request n threads***

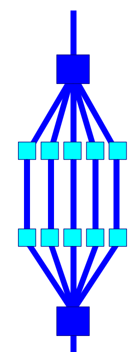
- ***Not recommended to exceed the number of processors***
- ***If the older variable PARALLEL is also set, the value should be equal***
- ***SUN Performance Library***
  - ✔ ***Checks for PARALLEL first. If not set, checks for OMP_NUM_THREADS to be set***

SUNW_MP_WARN TRUE | FALSE — ***Control printing of warnings***

- ***WARNING: The MT run-time library will not print warning messages by default***
- ***Set this environment variable to TRUE to activate the warnings***

Copyright©2003 Sun Microsystems

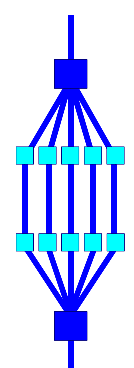
The parallel Performance Library



- ❑ ***A Shared Memory Parallel version is available:***
  - ❶ ***Compiler and explicitly parallelized user code:***
    - ✔ ***Link with -xparallel, -xautopar (or -xexplicitpar)***
      - ◆ ***Uses spin lock***
      - ◆ ***Fast, but may waste idle processors***
    - ✔ ***May consider to set SUNW_MP_THR_IDLE***
  - ❷ ***Code with Posix/Solaris threads (not considered here):***
    - ✔ ***Link with -mt***
      - ◆ ***Assumes system is shared among many tasks***
      - ◆ ***Uses threads library for synchronization***
- ❑ ***The number of threads is controlled through the OMP_NUM_THREADS (or PARALLEL environment variable)***

Copyright©2003 Sun Microsystems

Using The Compilers*



User Code	Perf. Library	Compile	Link**
Serial	Parallel		-xautopar
Auto-Parallel	Parallel	-xautopar	-xautopar
OpenMP	Parallel	-xopenmp	-xautopar
Auto+OpenMP	Parallel	-xautopar *and* -xopenmp	-xautopar
Parallel	Serial***	-xautopar	-xautopar

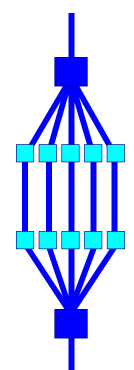
*) *It is assumed that you compile and link with -fast as well (for good serial performance)*

**) *Linking with -xautopar or -xparallel is equivalent*

***) *By default you'll get the parallel version if you're in a serial region; use Sun Perflib's routine "USE_THREADS" to override the number of threads*

Copyright©2003 Sun Microsystems

Example - Compile and Link



```
DO J = 1, N
   CALL DGEMM(.....)
END DO
CALL DGEMM
```

-fast → **DGEMM serial**

-fast -xparallel → **DGEMM parallel**

```
!$OMP PARALLEL ...
DO I = 1, N
   CALL DGEMM(.....)
END DO
(call use_threads(1))
CALL DGEMM
```

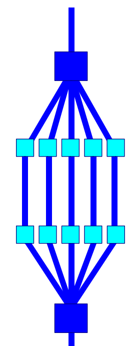
forces DGEMM to run on 1 processor

-fast → **DGEMM serial**

-fast -xopenmp → **DGEMM serial within loop, parallel outside of loop**

 Copyright©2003 Sun Microsystems

Auto-Parallelization Example

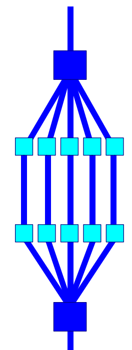


```
 1 void mxv_row(int m,int n,double *a,double *b,double *c)
 2 {
 3  int i, j;
 4  double sum;
 5
 6  for (i=0; i<m; i++)
 7  {
 8    sum = 0.0;
 9    for (j=0; j<n; j++)
10      sum += b[i*n+j]*c[j];
11     a[i] = sum;
12  }
```

```
% cc -c -fast -xrestrict -xautopar -xloopinfo mxv_row.c
"mxv_row.c", line 6: PARALLELIZED, and serial
              version generated
"mxv_row.c", line 9: not parallelized, unsafe
              dependence (sum)
```


Copyright©2003 Sun Microsystems

The Performance



Performance (Mflop/s)

Compiler - 1 CPU
Compiler - 2 CPUs
Compiler - 4 CPUs

Serial version executed

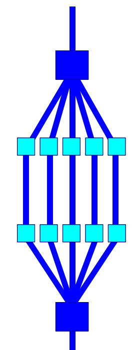
scales

Memory Footprint (KByte)

SunFire 6800
UltraSPARC-III Cu @ 900 MHz
8 MB L2-cache
S1SCC 7.0
Solaris 9

Copyright©2003 Sun Microsystems

The Sun Performance Library



Performance (Mflop/s)

4400
4000
3600
3200
2800
2400
2000
1600
1200
800
400
0

0 1 10 100 1000 10000 100000 1000000

PerfLib - 1 CPU
PerfLib - 2 CPUs
PerfLib - 4 CPUs

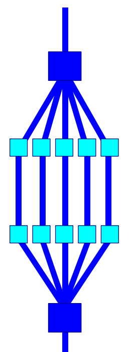
Matrix too small

scales

Memory Footprint (KByte)

SunFire 6800
UltraSPARC-III Cu @ 900 MHz
8 MB L2-cache
S1SCC 7.0
Solaris 9

Copyright©2003 Sun Microsystems

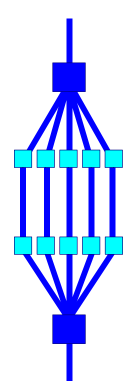


OpenMP™

http://www.openmp.org

Copyright©2003 Sun Microsystems

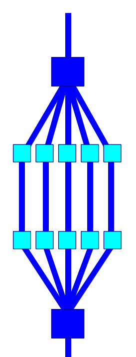
About OpenMP



- *The OpenMP programming model is a de-facto standard for Shared Memory Programming*
- *The approach chosen is based on the same informal way in which the Message Passing Interface (MPI) de-facto standard was defined*
- *OpenMP provides a compact, but powerful model*
- *Languages supported:* Fortran *and* C/C++
- *We will now present an overview of OpenMP*
- *Many details will be left out*
- *For specific information, we refer to the OpenMP language reference manuals* (http://www.openmp.org)

Copyright©2003 Sun Microsystems

The Fork and Join model



The Fork and Join model

The Master

"threads"

Parallel region

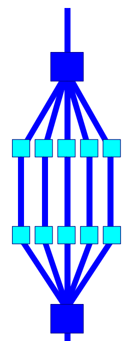
Synchronization

Parallel region

The Workers

Copyright©2003 Sun Microsystems

A loop parallelized with OpenMP



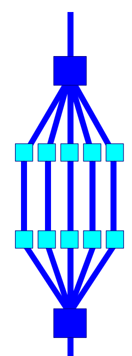
```
#pragma omp parallel default(none) \
                shared(n,x,y) private(i)
{
 #pragma omp for
   for (i=0; i<n; i++)
       x[i] += y[i];
}
```

clauses

```
!$omp parallel default(none)    &
!$omp shared(n,x,y) private(i)
!$omp do
      do i = 1, n
         x(i) = x(i) + y(i)
      end do
!$omp end do
!$omp end parallel
```

 Copyright©2003 Sun Microsystems

Components of OpenMP

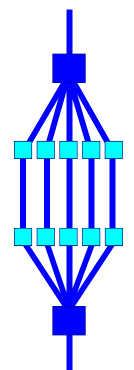


Directives, pragmas	Runtime library	Environment variables
◆ Parallel regions ◆ Work sharing ◆ Synchronization ◆ Data scope attributes ☞ private ☞ firstprivate ☞ lastprivate ☞ shared ☞ reduction ◆ Orphaning	◆ Number of threads ◆ Thread ID ◆ Dynamic thread adjustment ◆ Nested parallelism ◆ Timers ◆ API for locking	◆ Number of threads ◆ Scheduling type ◆ Dynamic thread adjustment ◆ Nested parallelism

The fork-join execution model is used

Copyright©2003 Sun Microsystems

A more elaborate example



```
do i = 1, n
   z(i) = x(i) + y(i)
end do
```

This is a parallel loop

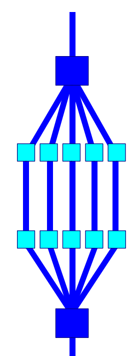
```
do i = 1, n
   a(i) = b(i) + c(i)
end do
```

This is a parallel loop

```
      ....
scale = sum(a(1:n)) + sum(z(1:n))
      ....
```


Copyright©2003 Sun Microsystems

Parallelized with OpenMP



```
!$omp parallel if ( n > 10000 ) default(none) &
!$omp shared(n,a,b,c,x,y,z) private(f,i,scale)
    f = 1.0
!$omp do
    do i = 1, n
       z(i) = x(i) + y(i)
    end do
!$omp end do nowait
!$omp do
    do i = 1, n
       a(i) = b(i) + c(i)
    end do
!$omp end do nowait
!$omp barrier
          ....
    scale = sum(a(1:n)) + sum(z(1:n)) + f
          ....
!$omp end parallel
```

Statement is executed by all threads

parallel loop

parallel loop

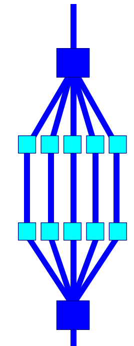
synchronization

Statement is executed by all threads

parallel region

Copyright©2003 Sun Microsystems

Another OpenMP example

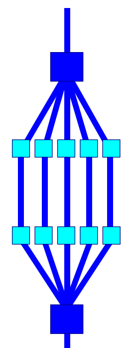


```
 1 void mxv_row(int m,int n,double *a,double *b,double *c)
 2 {
 3  int i, j;
 4  double sum;
 5
 6 #pragma omp parallel for default(none) \
 7             private(i,j,sum) shared(m,n,a,b,c)
 8  for (i=0; i<m; i++)
 9  {
10    sum = 0.0;
11    for (j=0; j<n; j++)
12      sum += b[i*n+j]*c[j];
13     a[i] = sum;
14  }
```

```
% cc -c -fast -xrestrict -xopenmp -xloopinfo mxv_row.c
"mxv_row.c", line  8: PARALLELIZED, user pragma used
"mxv_row.c", line 11: not parallelized
```


Copyright©2003 Sun Microsystems

OpenMP Performance



Performance (Mflop/s)

2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0

0 1 10 100 1000 10000 100000 1000000

Memory Footprint (KByte)

- OpenMP - 1 CPU
- OpenMP - 2 CPUs
- OpenMP - 4 CPUs

*Matrix too small **

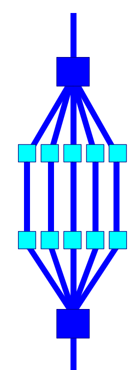
scales

SunFire 6800
UltraSPARC-III Cu @ 900 MHz
8 MB L2-cache
S1SCC 7.0
Solaris 9

****) With the IF-clause in OpenMP this performance degradation can be avoided***

Copyright©2003 Sun Microsystems

Pointers To More Information



Developer Portal
http:/developer.sun.com

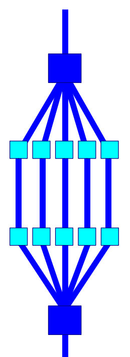
Compiler Collection Portal
http:/developer.sun.com/prodtech/cc

Introduction on the memory hierarchy:
http://www.sun.com/solutions/blueprints/1102/817-0742.pdf

How to compile on Sun:
http:
//developer.com/tools/cc/articles/US3Cu/US3Cu.content.html

More information on Solaris and 64-bit:
The "Solaris 7 64-bit Developer's Guide", part no 805-6250-10
(can be download from http://docs.sun.com)

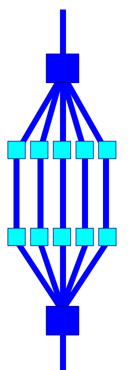
Copyright©2003 Sun Microsystems



That's It !

Thanks !

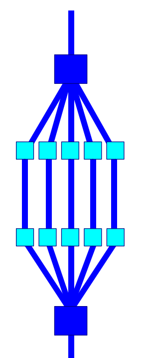
Copyright©2003 Sun Microsystems

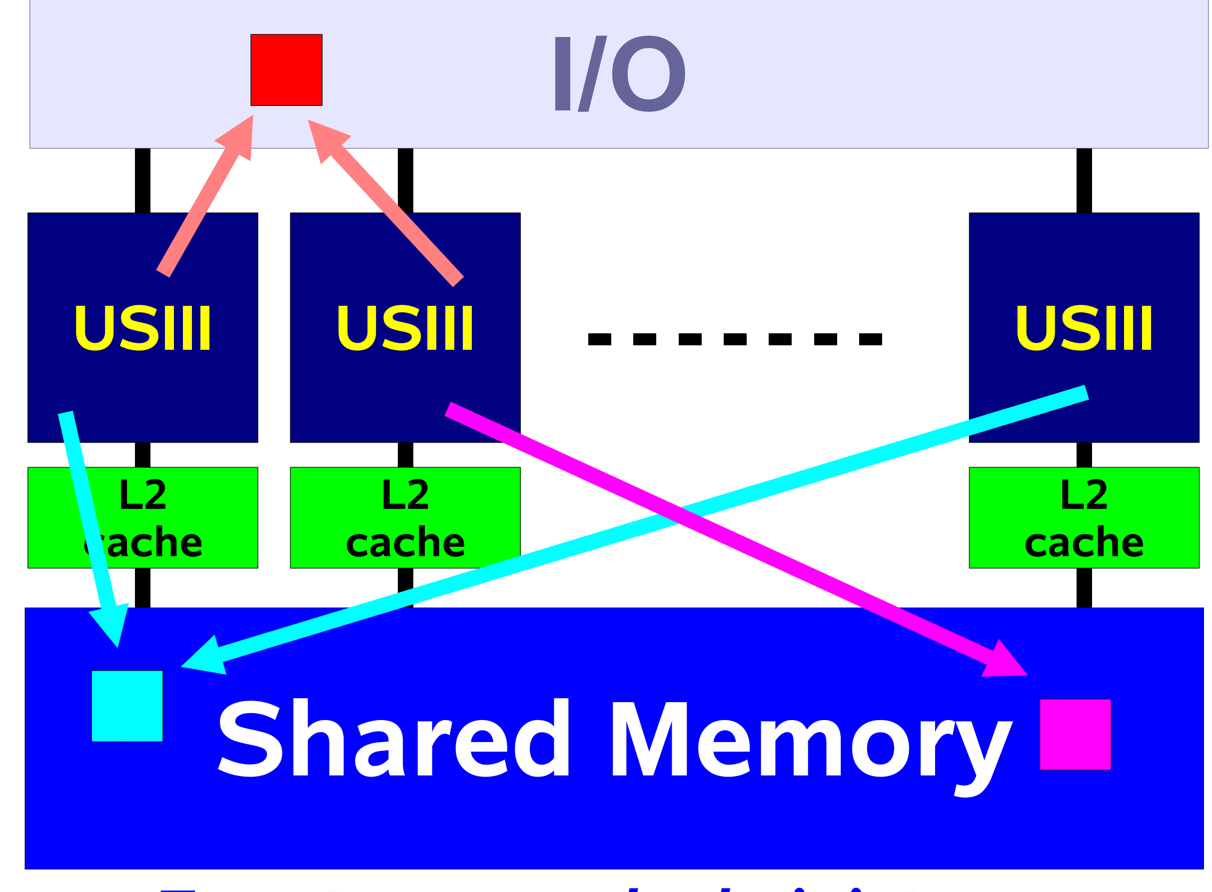


The SunFire Server Architecture

Copyright©2003 Sun Microsystems

Shared Memory Architecture

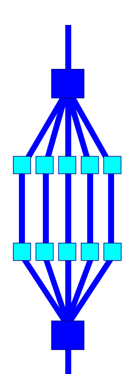




- ✔ *Easy to use and administer*
- ✔ *Efficient use of resources*
- ✔ *Scales as needs grow*

Copyright©2003 Sun Microsystems

Basic Building Block



L2

9.6 GB/s
(32B@300MHz)

Fireplane Address Router

USIII

Address &
Control

M

2.4 GB/s

2.4 GB/s

Fireplane Data Port

4.8 GB/s

CPU Data Switch
(5 ports)

2.4 GB/s

2.4 GB/s

USIII

Address &
Control

M

Fireplane Address Router

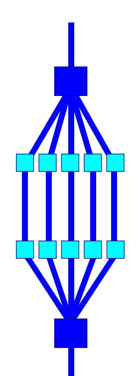
9.6 GB/s
(32B@300MHz)

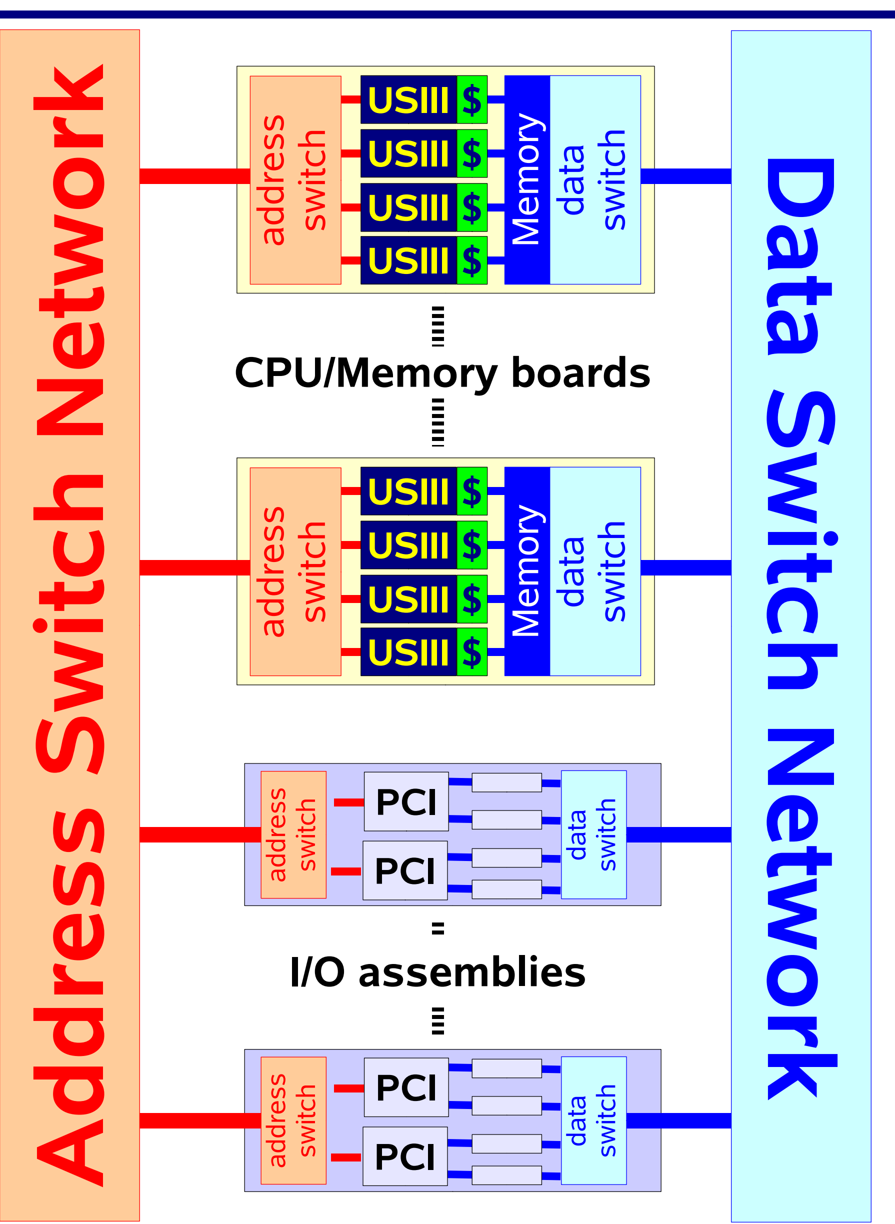
L2

Note: Second memory and data port not present on workstations

Copyright©2003 Sun Microsystems

The Simplified Big Picture

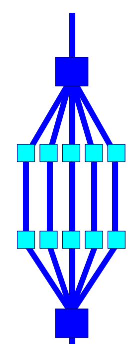




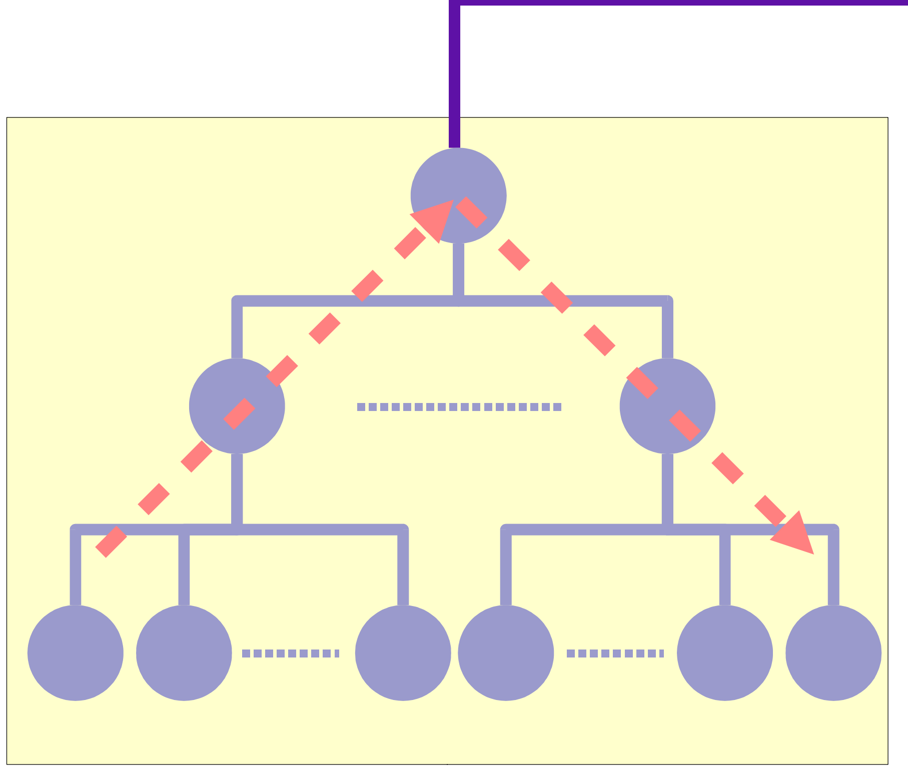
- The SMP model is preserved throughout product line
- Architectural details of the switch networks depends on Sun Fire model
- A hierarchical tree is used to build the interconnect
- Smaller systems, have less switch layers
- Largest system, the Sun Fire 15K, can have up to 106 processors

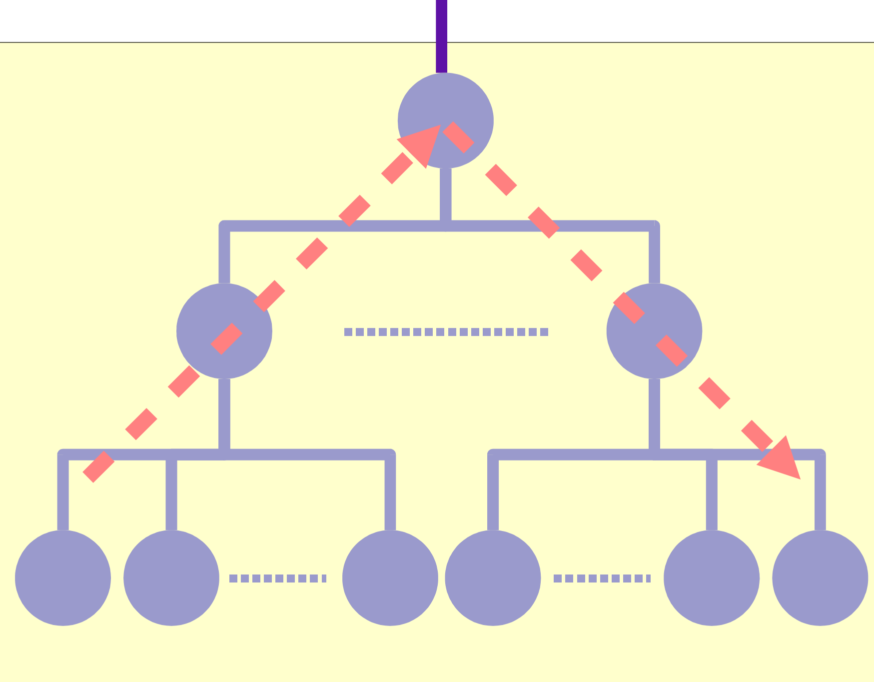
Copyright©2003 Sun Microsystems

A Hierarchical Tree



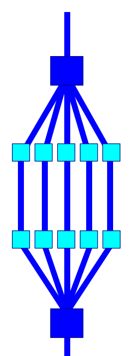
Point-To-Point
(SunFire 15K only)





Copyright©2003 Sun Microsystems

Interconnect Levels



Level 3

18x18 Address Crossbar

18x18 Response Crossbar

18x18 Data Crossbar

2 cycles/ address

1-2 cycles/ response

4.8 GB/s

Level 2

System Address Repeater

System Data Controller

System Data Switch

150 M snoops/sec

4.8 GB/s

150 M snoops/sec

2.4 GB/s

Level 1

Address Repeater

Datapath Controller

Data Switch

Address Repeater

Datapath Controller

Data Switch

4.8 GB/s

4.8 GB/s

1.2 GB/s

1.2 GB/s

Level 0

USIII L2 M

2.4 GB/s 2.4 GB/s

CPU Data Switch

2.4 GB/s 2.4 GB/s

USIII L2 M

USIII L2 M

2.4 GB/s 2.4 GB/s

CPU Data Switch

2.4 GB/s 2.4 GB/s

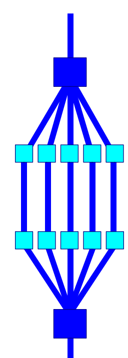
USIII L2 M

PCI — 200 MB/s — 33 MHz card(s); 400 MB/s — 66 MHz card(s)

PCI — 200 MB/s — 33 MHz card(s); 400 MB/s — 66 MHz card(s)

Copyright©2003 Sun Microsystems

Example - V880 WG Server



I/O Board

PCI 66MHz/64 bit

SCSI

PCI 33MHz/64 bit

PCI 33MHz/64 bit

10/100 Ethernet

USB

Secondary I/O bridge

Ebus

Serial I/O

Front Panel

PCI 66MHz/64 bit

Gbit Ethernet

FC-AL

I/O Bridge Connector

Motherboard

PCI Bridge

PCI Bridge

1.2 GB/s

1.2 GB/s

Address Repeater

Data Switch

Control Switch

4.8 GB/s

4.8 GB/s

4.8 GB/s

4.8 GB/s

VHDM Connector

USIII

L2

M

2.4 GB/s

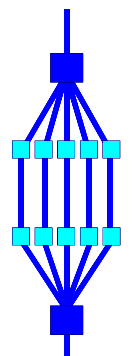
CPU Data Switch

Up to 4 CPU/Memory Boards

Note: Control paths not shown here

Copyright©2003 Sun Microsystems

Example - SunFire 6800



4 Fireplane
Switch Boards

System
Address
Repeater

(9.6 GB/s
peak)

System
Data
Controller

System
Data
Switch

(19.2 GB/s
bisection
bandwidth)

150 M
snoops/sec

6 CPU/Memory
Ports

150 M
snoops/sec

4 I/O
Ports

4.8 GB/s

6 CPU/Memory
Ports

2.4 GB/s

4 I/O
Ports

Up to 6 CPU/Memory Boards

Address
Repeater

Datapath
Controller

Data
Switch

4.8 GB/s

4.8 GB/s

USIII

L2

M

2.4 GB/s

2.4 GB/s

CPU Data Switch

2.4 GB/s

2.4 GB/s

USIII

L2

M

USIII

L2

M

2.4 GB/s

2.4 GB/s

CPU Data Switch

2.4 GB/s

2.4 GB/s

USIII

L2

M

Up to 4 I/O Assemblies

Address
Repeater

Datapath
Controller

Data
Switch

PCI

200 MB/s

33 MHz card(s)

400 MB/s

66 MHz card(s)

1.2 GB/s

PCI

200 MB/s

33 MHz card(s)

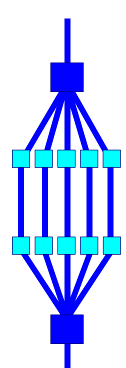
400 MB/s

66 MHz card(s)

1.2 GB/s

Copyright©2003 Sun Microsystems

System Board Set



SSM address transactions
(2 cycles/address)

SSM response transactions
(1-2 cycles/transaction)

Data transfers
4.8 GB/s

Expander board
- System Address Repeater
- System Data Controller
- System Data Switch

150 M snoops/sec

150 M snoops/sec

4.8 GB/s

2.4 GB/s

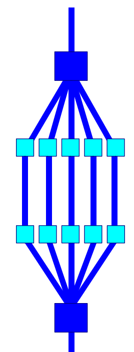
Slot 0 - CPUs, memory and switches
- Address Repeater
- Datapath Controller
- Data Switch
- 4.8 GB/s
- 4.8 GB/s
- USIII, L2, M (×4)
- CPU Data Switch (×2)
- 2.4 GB/s

Slot 1:
- PCI assembly *or*
- MaxCPU board

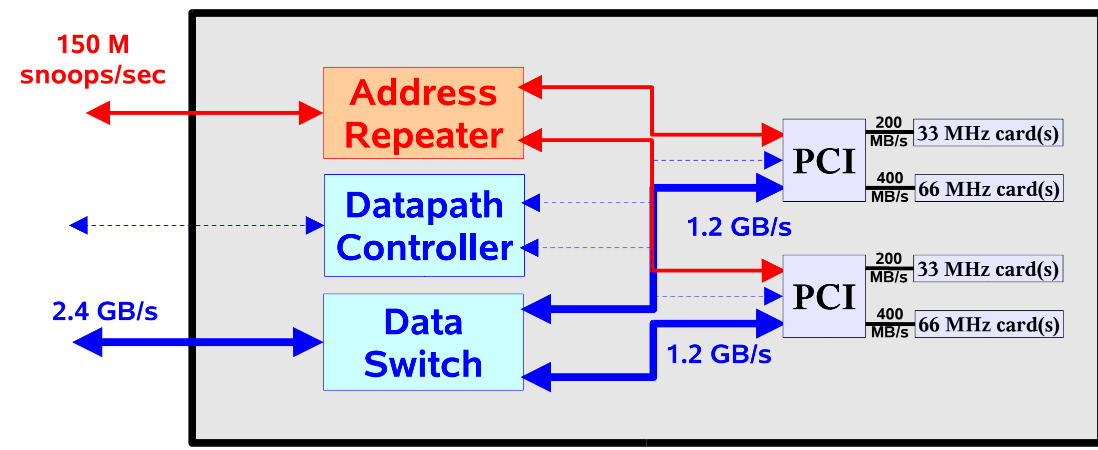
System Board Set

Copyright©2003 Sun Microsystems

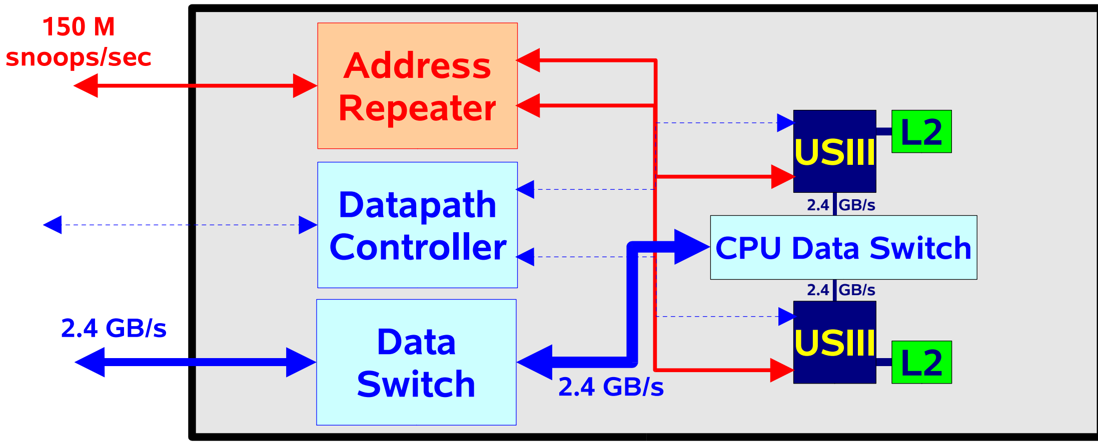
Slot 1 Boards



PCI Assembly

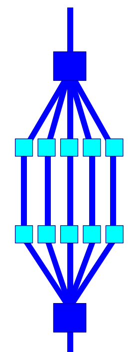


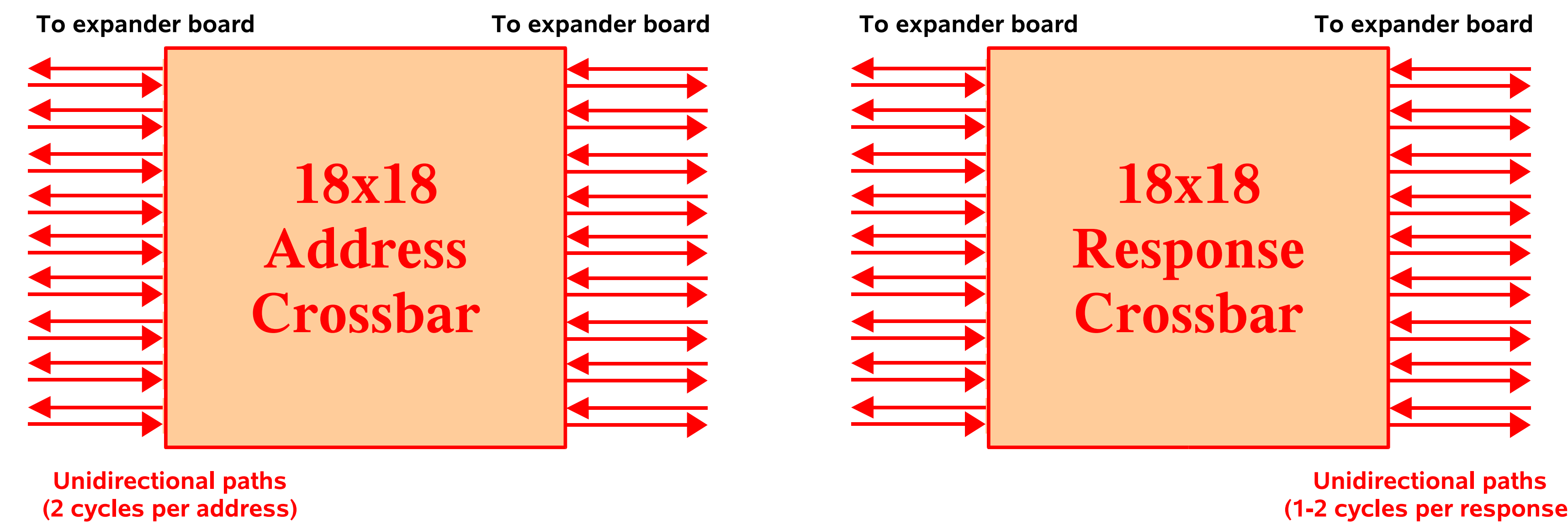
MaxCPU Board

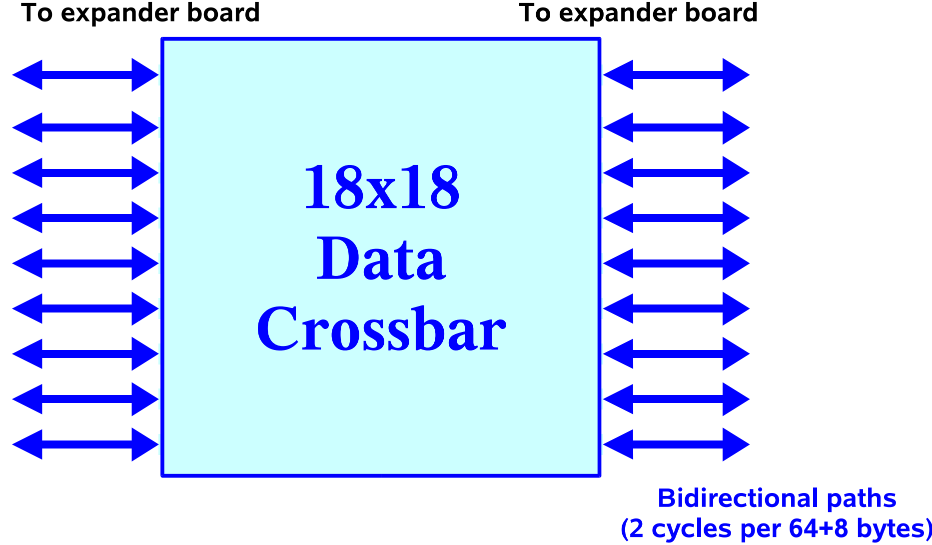


Copyright©2003 Sun Microsystems

Three Centerplane Crossbars

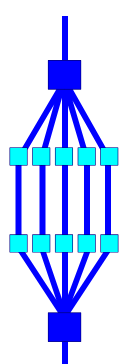






 Copyright©2003 Sun Microsystems

The Big Picture - SF15K



System Boards | Centerplane | System Boards

18x18 Address Crossbar

18x18 Response Crossbar

18x18 Data Crossbar

Copyright©2003 Sun Microsystems