

# Macro Architecture Resources and pSeries Specific Optimization

ScicomP9  
March, 2004

Charles Grassl  
IBM

1  
© 2004 IBM Corporation



## Agenda

- **Part 1: Macro Architecture and Resources**
  - Architecture Trends
  - Impact on Programming
  - POWER4 Processors
  - System Resources
- **Part 2: pSeries Specific Optimization**
  - Strategies
  - Techniques
  - Bandwidth Exploitation
  - Pipelining

2  
© 2004 IBM Corporation



## Performance Expectations

- **Floating point:**
  - Hundreds to 1000's Mflop/s
  - **Limitations:**
    - System bandwidth
    - Program model
      - Floating point operations
      - Adds and Multiplies
      - Divides
    - Memory access
      - Copying
- **Bandwidth:**
  - 0.100 - 5 Gbyte/s
  - **Limitations:**
    - Strided access is slow
    - Multiple streams

3

© 2004 IBM Corporation



## Performance Expectations Example: program POP

- Structured Fortran 90
- Data movement
- Low Computational Intensity
- Low FMA Percentage

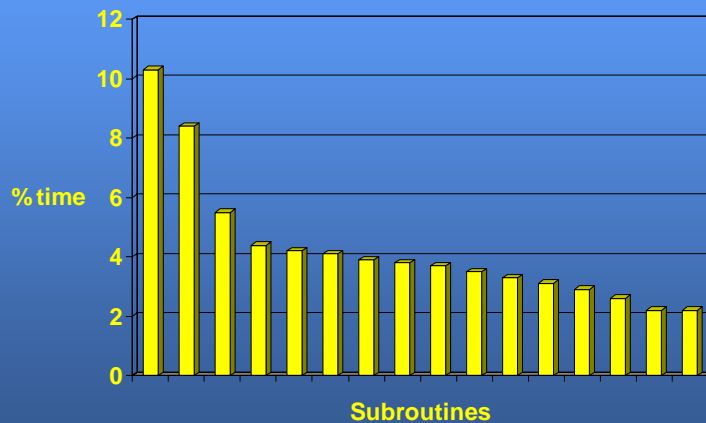
Instructions per cycle	0.9
HW Float points Inst. per Cyle	0.3
Floating point Instructions	10141 M
Float point instruction rate	372 Mflop/s
FMA percentage	53 %
Computation intensity	0.87

4

© 2004 IBM Corporation



## Program POP: Computation Time Distribution



5

© 2004 IBM Corporation



## Performance Expectations Example: program SPPM

- Fortran 77
- Optimized
- High Computational Intensity
- Vector intrinsics
- Low FMA Percentage

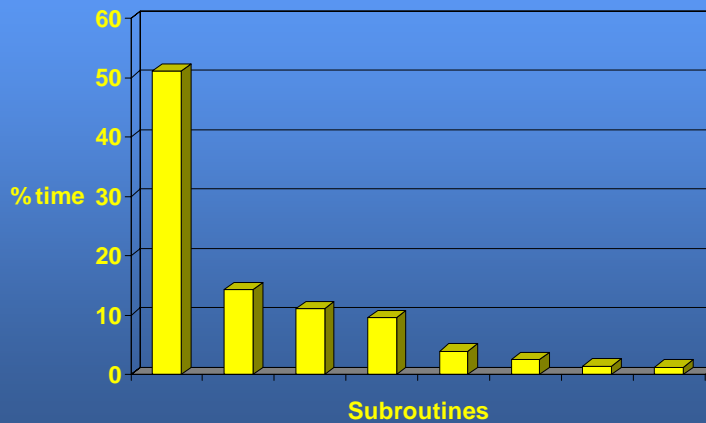
Instructions per cycle	1.2
HW Float points Inst. per Cycle	0.7
Floating point Instructions	191752 M
Float point instruction rate	979 Mflop/s
FMA percentage	55 %
Computation intensity	1.8

6

© 2004 IBM Corporation



## Program SPPM: Computation Time Distribution



7

© 2004 IBM Corporation



## Part 1: Macro Architecture and Resources

- Architecture Trends
  - Impact on Programming
- POWER4 Processors
  - System Resources

8

© 2004 IBM Corporation



## Underlying Technology

- Transistor construction
- Lithography
- Material Science
- Electronic structures
- Circuit density
- Fabrication

9

© 2004 IBM Corporation



## Integrated Circuit Trends

- Smaller line widths
  - Nearly 0.1 micron
  - Lower voltages
- Higher transistor count per chip
  - ~200 millions transistors per chip
- Area not growing fast
  - ~400 sq. mm
- High signal speed
  - SiO<sub>2</sub>
  - Copper
  - Low - k dielectric

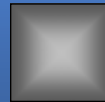
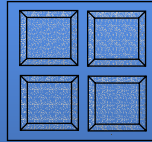
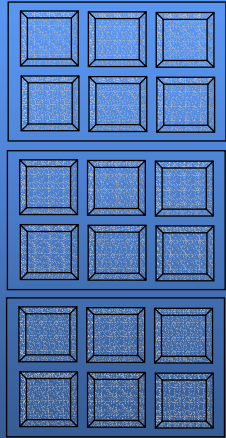
10

© 2004 IBM Corporation



# Increasing Transistor Density

Processor, 1990



Processors, 2002

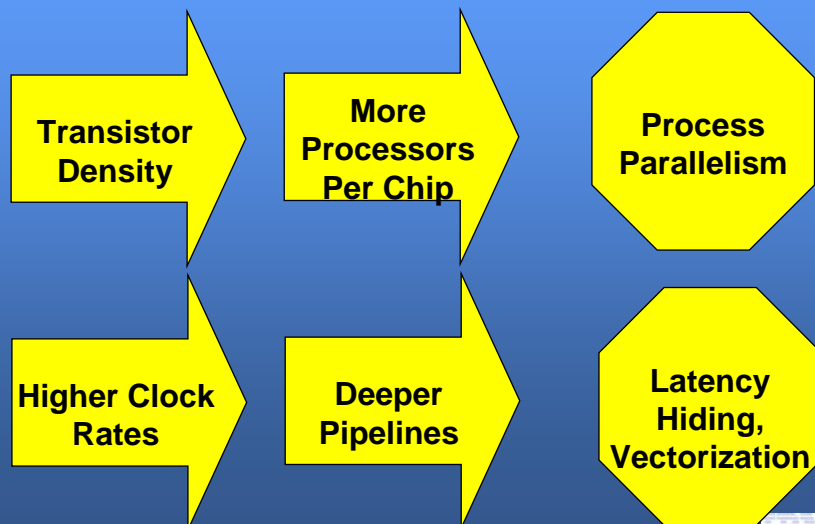


11

© 2004 IBM Corporation



# Effect of Transistor Technology



12

© 2004 IBM Corporation



## Effects on Software

- **More Processors**
  - **Shared Memory Parallelism**
    - Detection
    - Explicit
- **Compilers**
  - Automatic parallelism
  - OpenMP
- **Scalability**
- **Communication**
  - Barriers
- **Deep Pipelines**
- **More registers**
- **Vectorization techniques**
- **Latency hiding**
- **Instruction Level Parallelism**

13

© 2004 IBM Corporation



## Microprocessor Architecture Trends

- Multithread
  - Cray MTA
  - Intel
    - Pentium
    - Xeon
  - Sun
  - IBM
    - P660
    - POWER5
- Multiple processors per chip
  - POWER4
  - IBM Blue Gene
  - POWER5
  - HP PA
  - Future Itanium

14

© 2004 IBM Corporation



## Contrarian Trends

- **Smaller processors**
  - Blue Gene
- **Fewer processors per chip**
- **Tighter coupling**
- **Hardware synchronization**
- **Clustered processors**
- **MSP**
- **ViVa**

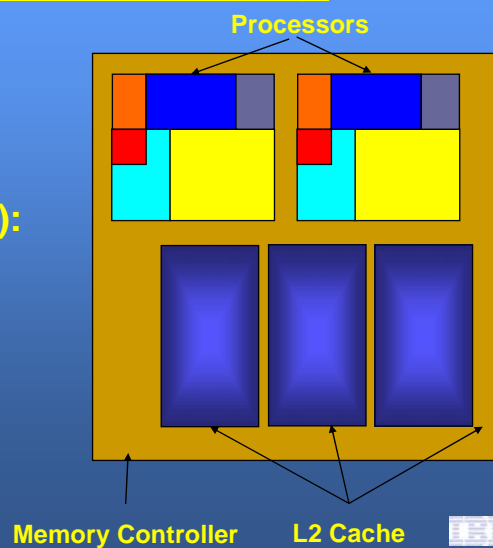
15

© 2004 IBM Corporation



## POWER4: Multi-processor Chip

- **Each processor:**
  - L1 caches
  - Registers
  - Functional units
- **Each chip (shared):**
  - L2 cache
  - L3 cache
  - Path to memory



16

© 2004 IBM Corporation



## POWER4 Processor: Performance Features

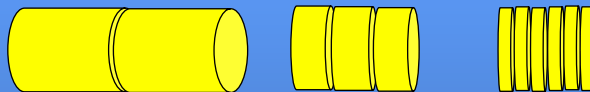
- Multiprocessor chip
- High clock rate
  - Long pipelines
- Three cache levels
  - Latency hiding
- Shared Memory
  - Large memory size

17

© 2004 IBM Corporation



## POWER Architecture FMA Trend



	POWER2	POWER3	POWER4
Clocks periods	2	3	6
Clock Rate	125 MHz	375 MHz	1.3 GHz
Transit Time	16 nanosec	8 nanosec	4.4 nanosec
Registers	32	32	32

18

© 2004 IBM Corporation



## POWER4 Floating Point Functional Units

- Two symmetric floating point units
  - Divide and square-root sub-units
- Double precision (64-bit) data path

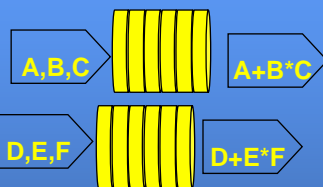
Instruction	Single	Double	Pipelined
Fma	6	6	YES
Fdiv	32	32	No
Fsqrt	38	38	No

19

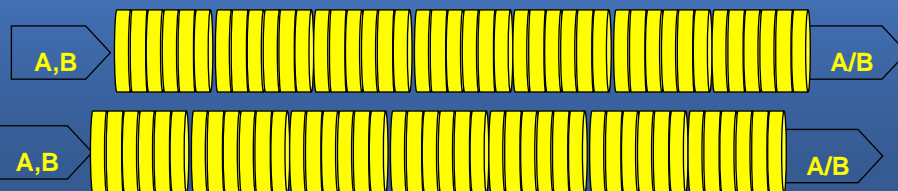
© 2004 IBM Corporation



## Pipelined Functions



	Results per clock period
Fma	12/6
Divide	2/32
Sqrt	2/38



20

© 2004 IBM Corporation



## Deep Pipelines

- Operations limited by functional unit transit time:
  - Divide
  - Square root
  - Intrinsic functions
  - Recursion

21

© 2004 IBM Corporation



## Attaining Peak Speed (Demonstration Kernel)

- Independent scalar triads
- Test code:

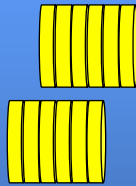
```
do i=1,n
  y1 = x1 + s*x1
  y2 = x2 + s*x2
  ....
  x1 = y1 + s*y1
  x2 = y2 + s*y2
  ....
end do
```

22

© 2004 IBM Corporation



# Fma Functional Unit



Clock Periods	6
Results per Clock	4
Results in Transit	24

Triads	Rate (Gflop/s)
1	433
2	866
4	1733
6	2600
8	3466
10	4333
12	5200

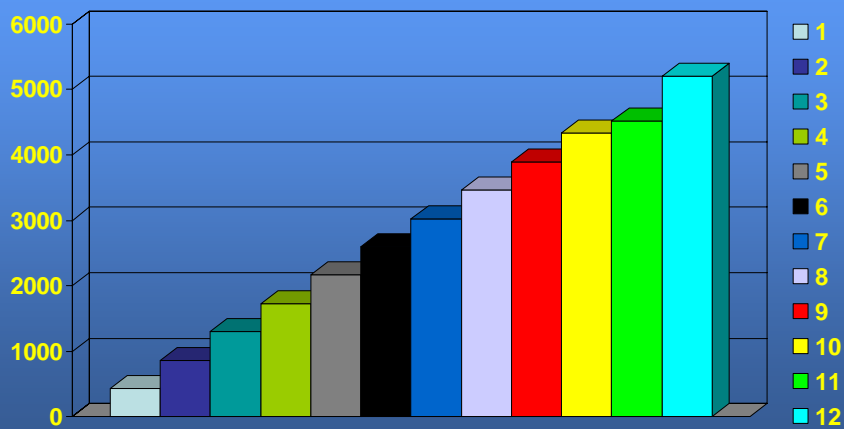
1.3GHz POWER4

23

© 2004 IBM Corporation



# Independent Triad Performance



24

© 2004 IBM Corporation



## Effect of Registers

Processor	DEGEMM Speed (Mflop/s)	% Peak
POWER2	600	98
POWER3	1480	98
POWER4	3500	70
POWER5		~90

- **POWER4 registers**
  - Architecture: 32
  - Rename: 72

25

© 2004 IBM Corporation



## POWER4 Macro Architecture

- **Three Levels of cache**
  - **L1**
    - 32 kbyte data
    - 64 kbyte instruction
  - **L2**
    - 1.5 Mbyte
      - Shared by two processors
  - **L3**
    - 4 x 32 Mbyte per Multi Chip Module (MCM)
    - Shared across system

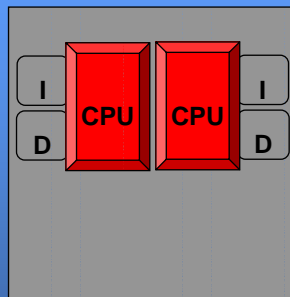
26

© 2004 IBM Corporation



## L1 Cache

- **Size:**
  - 32 kbyte data
- **Bandwidth:**
  - 2 words x 8 bytes x clock\_rate (Same as POWER3)
  - 20 Gbyte/s
- **Latency:**
  - 1 clock period



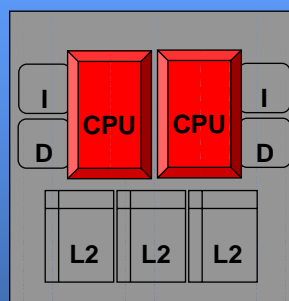
27

© 2004 IBM Corporation



## L2 Cache

- **Size:**
  - 1.5 Mbyte
- **Bandwidth:**
  - 100 Gbyte/s
- **Latency:**
  - 8 - 10 clock periods
- **Three “slices”**
- **Shared by pairs of processors**



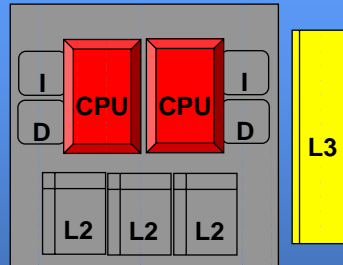
28

© 2004 IBM Corporation



## L3 Cache

- **Size:**
  - 32 Mbyte
- **Bandwidth:**
  - 11 Gbyte/s
- **Latency:**
  - ~100 clock periods
- **Shared by pairs of processors (bandwidth)**
- **Shared with other processors on node**



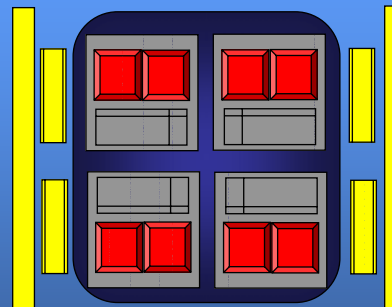
29

© 2004 IBM Corporation



## Memory: Single MCM

- **Size:**
  - Up to 128 Mbyte per MCM
- **Bandwidth:**
  - 11 Gbyte/s per physical L3
- **Latency:**
  - ~300 clock periods



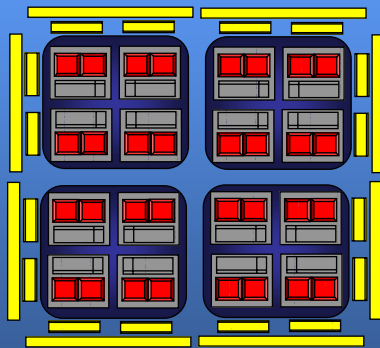
30

© 2004 IBM Corporation



## Memory: Multiple MCMs

- **Size:**
  - Up to 512 Mbyte
- **Bandwidth:**
  - 16 x 11 Gbyte/s
  - 50 Gbyte/s attainable
- **Latency:**
  - ~300 clock periods



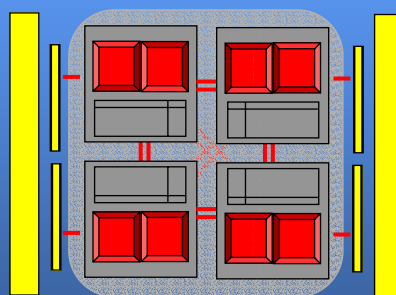
31

© 2004 IBM Corporation



## Intra-module Communications

- 4 busses, 16 bytes, 2:1
- 41 Gbyte/s



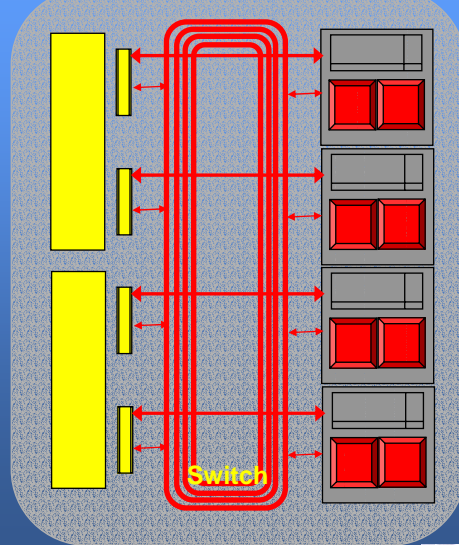
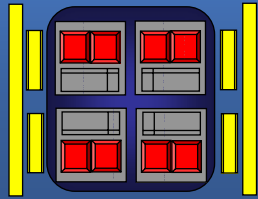
32

© 2004 IBM Corporation



## Intra-module Communications

- 4 busses, 16 bytes, 2:1
- 41 Gbyte/s



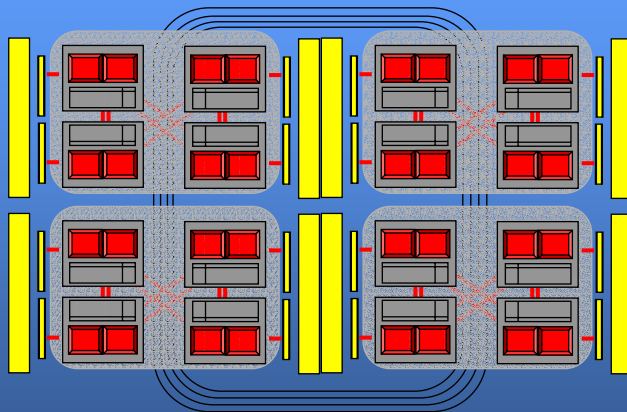
33

© 2004 IBM Corporation



## Inter-module Communications

- 4 busses
- 8 bytes
- 2:1
- 21 Gbyte/s



34

© 2004 IBM Corporation



## pSeries Special Features

- Large Pages
- Memory Affinity
- Process Binding

35

© 2004 IBM Corporation



## Large Page Effects

- Additional bandwidth
- Additional Translation Lookaside Buffer (TLB) coverage
- Poor interactive performance
  - Scripts
  - Compile
  - Forks

36

© 2004 IBM Corporation



## Large Page Motivation

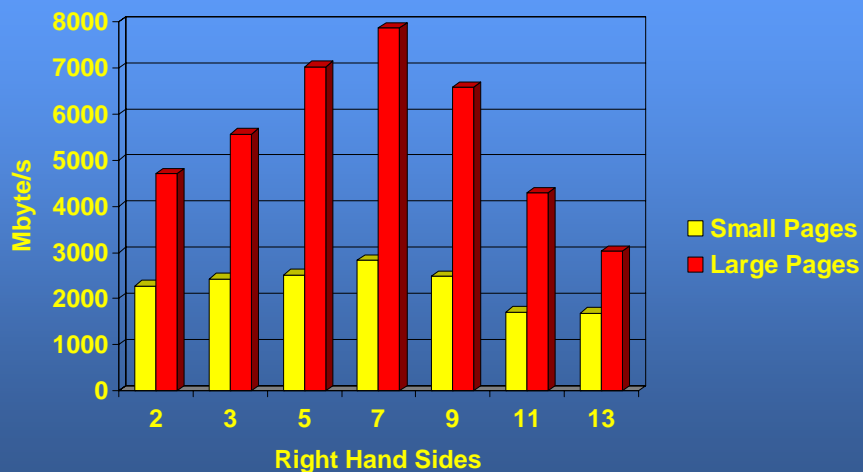
- Enhance memory bandwidth
  - Prefetch performance is limited by page size
- Bandwidth increase of up to 3x
- Enhance Translation Lookaside Buffer (TLB) coverage
  - POWER3: 128x2 TLB entries
  - POWER4: 1024 TLB entries

37

© 2004 IBM Corporation



## Memory Bandwidth

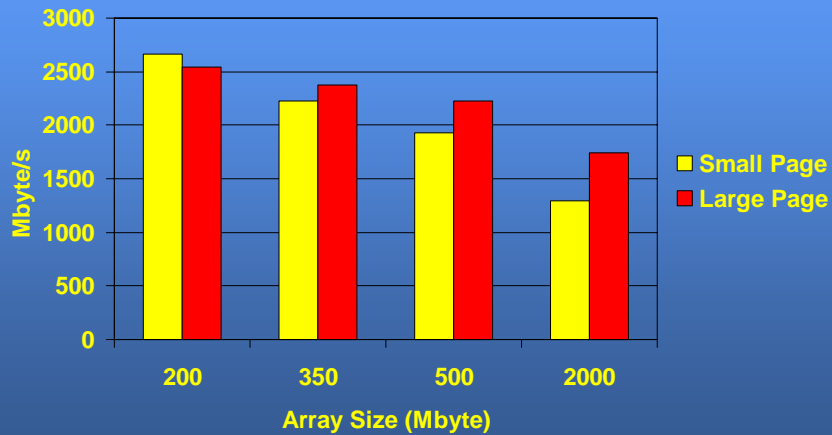


38

© 2004 IBM Corporation



## TLB Span



A(:) = A(:) + B(index(:))

39  
© 2004 IBM Corporation

## Large Page Usage

- **Detection**
  - Old:
    - \$ vmtune
    - ..... -lp .....
  - AIX 5.2:
    - \$ vmstat -l # Small L
    - ..... alp flp
    - ..... 50 650

40  
© 2004 IBM Corporation

## Large Page Usage

- **Loader:**
  - \$ xlf .... -blpdata -o a.out
- **ldedit:**
  - \$ /usr/ccs/bin/ldedit -blpdata a.out
- **Environment variable:**
  - LDR\_CNTRL=LARGE\_PAGE\_DATA={Y,N,M}
  - Y: Yes ("advisory" or "maybe") mode
    - Use large pages if available
    - Mode used by loader and ldedit
  - N: No large pages
  - M: Mandatory
    - Do not run if large pages not available

41

© 2004 IBM Corporation



## Large Page Usage

Method	Usage	Comment
Linker	Xlf -blpdata ...	Advisory
ldedit	Ldedit -blpdata ...	Advisory
Env. Variable	LDR_CNTRL= LARGE_PAGE_DATA = [Y,M,N]	Y: Advisory M: Mandatory N: No

42

© 2004 IBM Corporation



## Expected Application Performance

- **Memory intensive applications**
  - Single CPU performance increase: typically 5-20%
- **Multiple CPU performance not as much**
  - Nothing to gain if all memory bandwidth already used

43

© 2004 IBM Corporation



## Memory Affinity

- **Allocate memory pages to local module**
- **Access to local boards is robust**
  - Little contention
  - Except for processor pairs on chip
  - Lower latency
- **Access to remote modules uses buses**
  - Contention
  - Slightly higher latency (~10%)

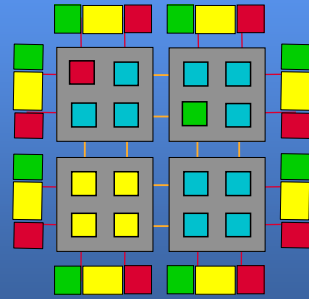
44

© 2004 IBM Corporation



## Default Memory Allocation

- Pages allocated to all modules
- Independent of process location (approximately)
- Periodic placement ("approximate round robin")



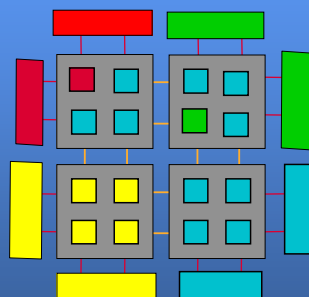
45

© 2004 IBM Corporation



## Memory Affinity

- Pages allocated preferentially to local module



46

© 2004 IBM Corporation



## Memory Affinity Concerns

- **Less system-wide contention**
  - Works well for MPI
  - Memory localization
- **Difficulty with threads**
  - New threads may require references to remote memory
  - "First touch" strategies

47

© 2004 IBM Corporation



## Memory Affinity Implementations

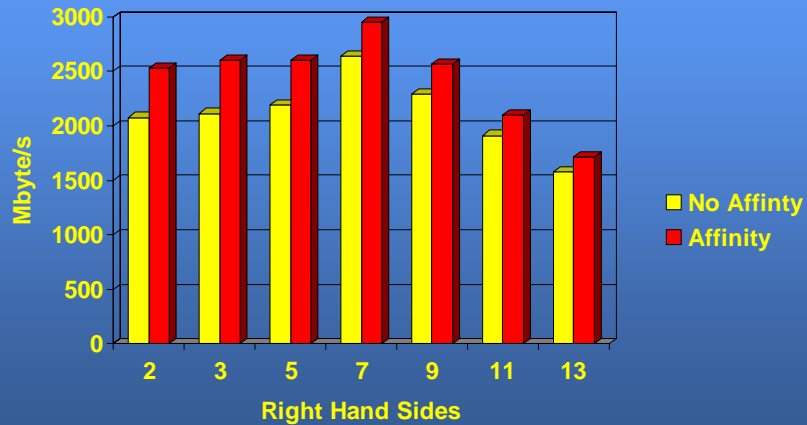
- **Export MEMORY\_AFFINITY=MCM**

48

© 2004 IBM Corporation



## Memory Bandwidth: Affinity



49

© 2004 IBM Corporation



## Process Binding

- **Memory does not migrate**
  - Processes migrate over time
  - Some processor affinity is in effect from AIX
- **Process binding can be detrimental in shared usage**
  - Bound process might share processor with other users processes

50

© 2004 IBM Corporation



## Process Binding

- **Binds or unbinds the kernel threads of a process to a processor**
  - /usr/sbin/binprocessor
- **Examples:**

```
$ bindprocessor -q
The available processors are: 0 1 2 3 4 5 6 7
$ ps
PID  TTY  TIME CMD
35036 pts/6 2:02 cg.B
44268 pts/6 0:00 -ksh
$ bindprocessor 35036 4
```

51

© 2004 IBM Corporation



## bindUtils library

- **NOT A PRODUCT**
- **Library utility which maps tasks or threads to specific processors**
- **Does not map memory (affinity)**
- **Usage:**

```
$ XI{fc} .. -I {OMP,MPI,MPIOMP}bin
$ export TARGET_CPU_LIST="0 2 4 6"
$ a.out
```

52

© 2004 IBM Corporation



## Part 2: Optimization

- **Bandwidth Exploitation**
- **Pipelining**

53

© 2004 IBM Corporation



## Bandwidth Exploitation

- **Computational intensity**
  - Increase number of flop/s per memory reference
- **Blocking**
  - Reuse cache
- **Load streams**
  - Expose up to 8 memory patterns

54

© 2004 IBM Corporation



## Computational Intensity Examples

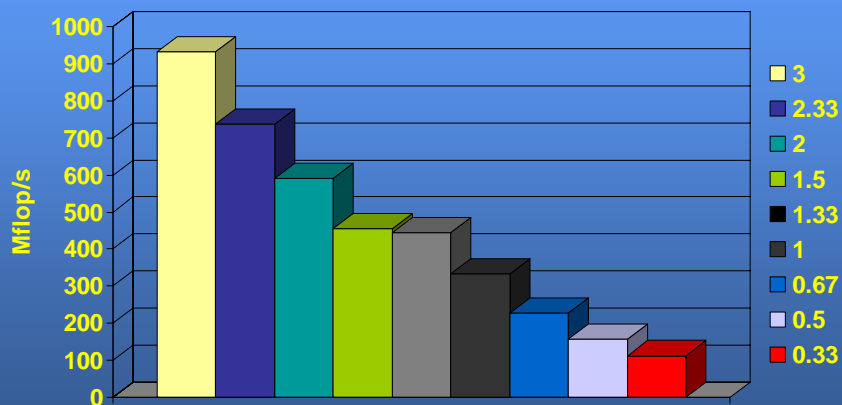
Loop	Comp. Inten.	Comment
$A(i)=r+B(i)*(s+t*B(i))$	2	
$A(i)=r*B(i)+s*C(i)$	1	Triad
$A(i)=r*B(i)+s$	1	Scale
$A(i)=A(i)+s*B(i)$	0.67	AXPY
$A(i)=B(i)+C(i)$	0.33	SUM

55

© 2004 IBM Corporation



## Computational Intensity Tests



Data from 1.3 GHz p690

56

© 2004 IBM Corporation



## Computational Intensity

- **Nominal bandwidth is 2.5 Gbyte/s for single line loops**
  - 300 Mword/s
  - ...
  - 300 Mflop/s for Comp. Intens. of 1
  - 600 Mflop/s for Comp. Intens. of 2
  - ...
- **Performance is limited by bandwidth**
  - NOT functional unit performance

57

© 2004 IBM Corporation



## Computational Intensity Strategy: Loop Unrolling

- **Outer Loop Strategy:**
  - Increase computational intensity
  - Minimizes load/stores
- **Find variable which is constant with respect to outer loop**
  - Unroll such that this variable is loaded once, but used multiple times

58

© 2004 IBM Corporation



## Outer Loop Unrolling Example

```

DO I = 1, N
  DO J = 1, N
    s = s + X(J)*A(J,I)
  END DO
END DO
    
```



```

DO I = 1, N, 4
  DO J = 1, N
    s = s + X(J)*A(J,I+0)
      + X(J)*A(J,I+1)
      + X(J)*A(J,I+2)
      + X(J)*A(J,I+3)
  END DO
END DO
    
```

2 flops / 2 loads  
Comp. Int.: 1

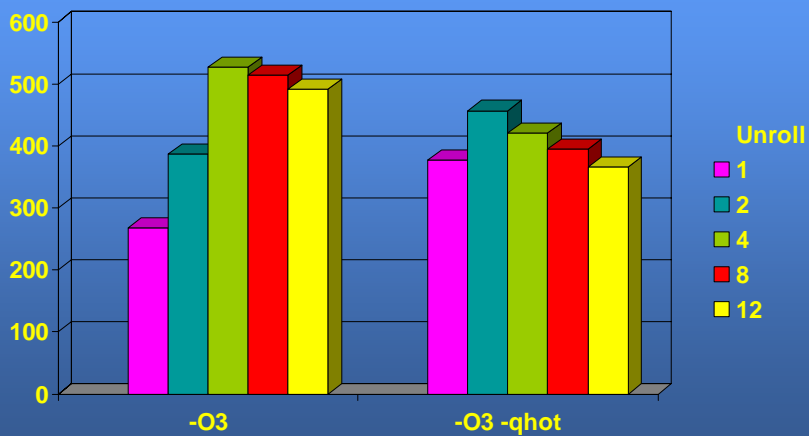
8 flops / 5 Loads  
Comp. Int.: 1.6

59

© 2004 IBM Corporation



## Outer Loop Unroll Test



60

© 2004 IBM Corporation



## Loop Unroll Analysis

- **Strategy:**
  - Unroll up to 8 times
  - Expose 8 prefetch streams
- **Compiler:**
  - Unrolls 4 times
  - Near optimal performance
  - Combines inner and outer loop unrolling

61

© 2004 IBM Corporation



## Loop Unrolling Strategies

- **Inner loop strategy:**
  - Reduces data dependency
  - Eliminate intermediate loads and stores
  - Expose functional units
  - Expose registers
- **Examples:**
  - Linear recurrences
  - Simple loops
    - Single operation

62

© 2004 IBM Corporation



## Inner Loop Unroll

```
do i=2,n-1  
  A(i+1) = A(i)*s1 +  
          A(i-1)*s2  
end do
```

→

```
do i=2,n-2,2  
  A(i+1) = A(i )*s1 +  
          A(i-1)*s2  
  A(i+2) = A(i+1)*s1  
          + A(i )*s2  
end do
```

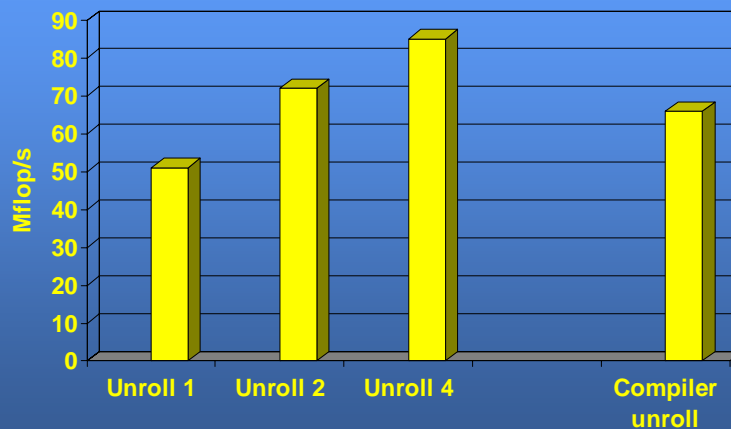
- Eliminate data dependence (half)
- Eliminate intermediate loads and stores
- Compiler will do some of this at -O3

63

© 2004 IBM Corporation



## Inner Loop Unroll Test



Data form 200 MHz POWER3

64

© 2004 IBM Corporation



## Inner Loop Unrolling

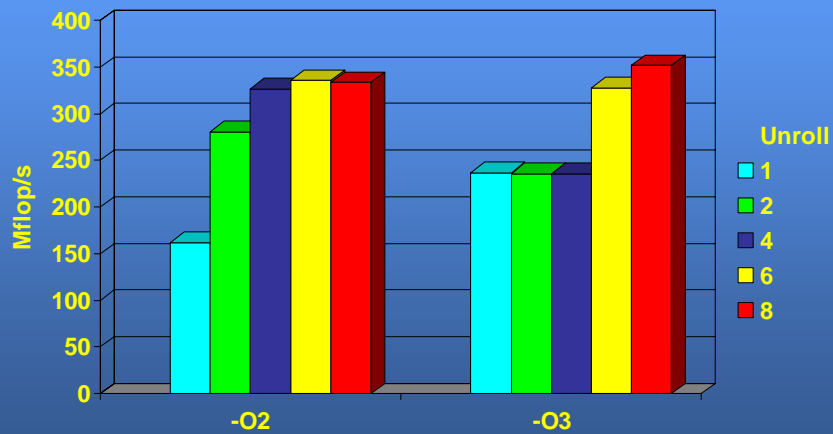
- Expose functional units
- Expose registers
- Example:  
do i=1,n  
    sum = sum + A(i)  
end do

65

© 2004 IBM Corporation



## Inner Loop Unrolling Test



Data from 1.3 GHz POWER4

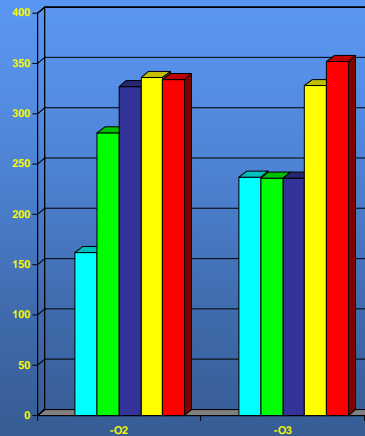
66

© 2004 IBM Corporation



## Inner Loop Unrolling: Conclusion

- Compiler unrolling is adequate
- Use manual unrolling for special cases



67

© 2004 IBM Corporation



## Memory Access Strides

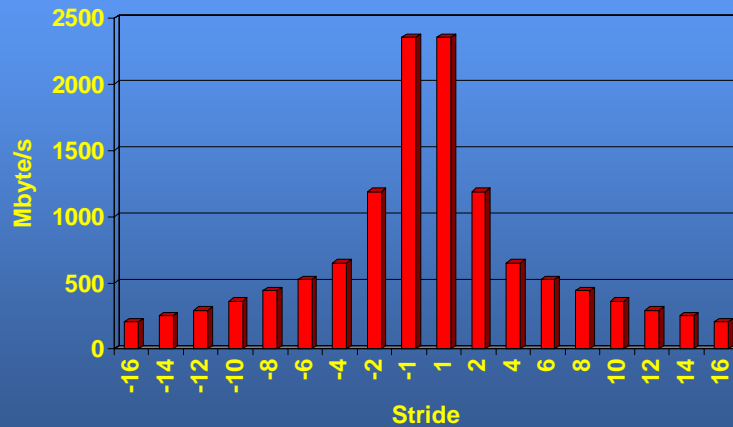
- Strided memory access:
  - Fewer words per cache line
  - Reduced cache line utilization
  - Reduced bandwidth
  - Memory is access by cache line
  - 16 REAL\*8 or double words

68

© 2004 IBM Corporation



## Stride Test Bandwidth



1.3 GHz POWER4

© 2004 IBM Corporation



## Strides and TBL

- **Strided memory access:**
  - Fewer memory reference per memory page
  - Increased TLB misses
- **TLB signature obtained from:**
  - `hpmcount -g 59`
  - Memory references per TLB

```
do i=1,m
  do j=1,n
    A(i,j)=A(i,j)+...
  end do
end do
```

70

© 2004 IBM Corporation



## Large Stride Test



71

© 2004 IBM Corporation



## Cache Blocking

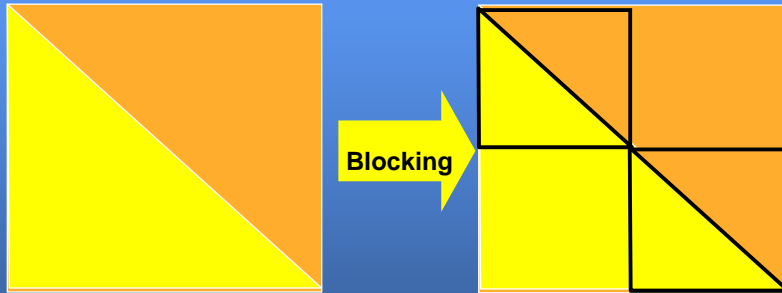
- **Common technique in Linear Algebra**
  - Similar to unrolling
  - Utilize cache lines
  - Linear Algebra NB:
    - Typically 96-256

72

© 2004 IBM Corporation



## Blocking



73

© 2004 IBM Corporation



## Blocking

```
do i = 1, n
  do j = 1, m
    B(j,i) = A(i,j)
  end do
end do
```



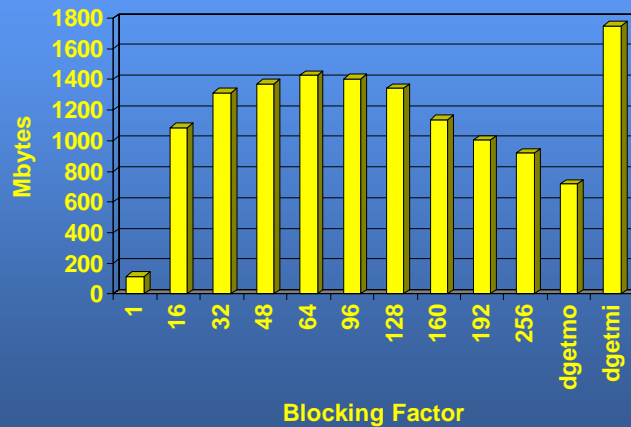
```
do j1 = 1, n-nb+1, nb
  j2 = min(j1+nb-1, n)
  do i1 = 1, m-nb+1, nb
    i2 = min(i1+nb-1, m)
    do i = i1, i2
      do j = j1, j2
        B(j,i) = A(i,j)
      end do
    end do
  end do
end do
```

74

© 2004 IBM Corporation



## Blocking Example: Transpose



75

© 2004 IBM Corporation



## Software Pipelining

- **Exploit registers**
  - Number of registers in POWER4 is critical
  - Some assistance from rename registers
- **Compiler does unrolling at -O3 and higher**
  - User unrolling can conflict with compiler

76

© 2004 IBM Corporation



## Software Pipelining

```
do i=1,n
  sum = sum + X(i)
end do
```

→

```
do i=1,n-3,4
  sum1 = sum1 + X(i )
  sum2 = sum2 + X(i+1)
  sum3 = sum3 + X(i+2)
  sum4 = sum4 + X(i+3)
end do
sum=sum1+sum2+
sum3+sum4
```

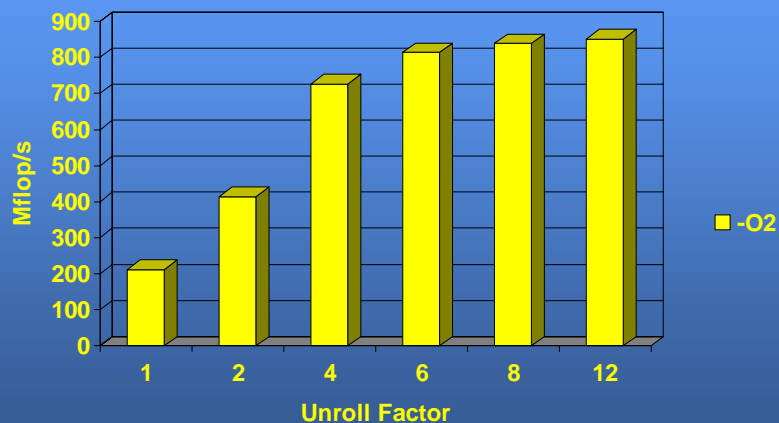
Explicit unrolling:  
Expose more variables for register use

77

© 2004 IBM Corporation



## Software Pipelining Example

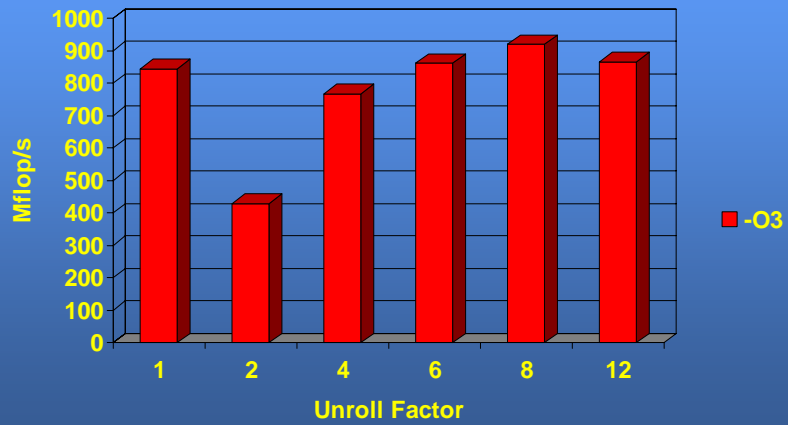


78

© 2004 IBM Corporation



## Software Pipelining Example

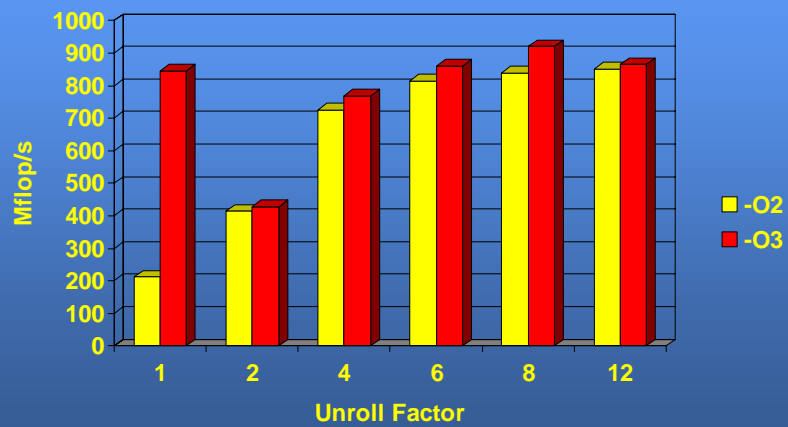


79

© 2004 IBM Corporation



## Software Pipelining Example



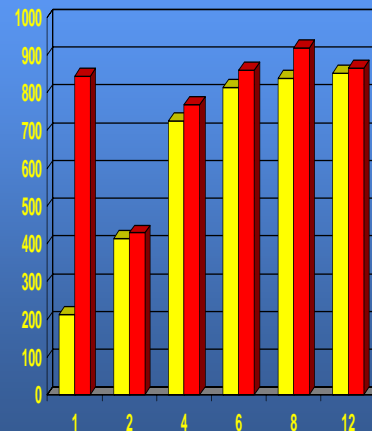
80

© 2004 IBM Corporation



## Software Pipelining Example

- **Conclusion:**
- **Avoid explicit user unrolling at -O3**
- **Allow compiler to perform unrolling**



81

© 2004 IBM Corporation



## Prefetch Strategies

- **Merge loops**
  - Combine conforming loops
    - Compiler can do much of this
- **Folding**
  - Useful for very long loops
    - Compiler can do much of this

82

© 2004 IBM Corporation



## Merge Loops

- Overlap cache line fetches
- Expose memory prefetching

```
for (j=1; j<= n; j++)  
    A[j] = A[j-1]+B[j]  
for (j=1; j<= n; j++)  
    D[j] = D[j+1]+C[j]*s
```



```
for (j=1; j<= n; j++)  
{  
    A[j] = A[j-1]+B[j]  
    D[j] = D[j+1]+C[j]*s  
}
```

Two streams per loop

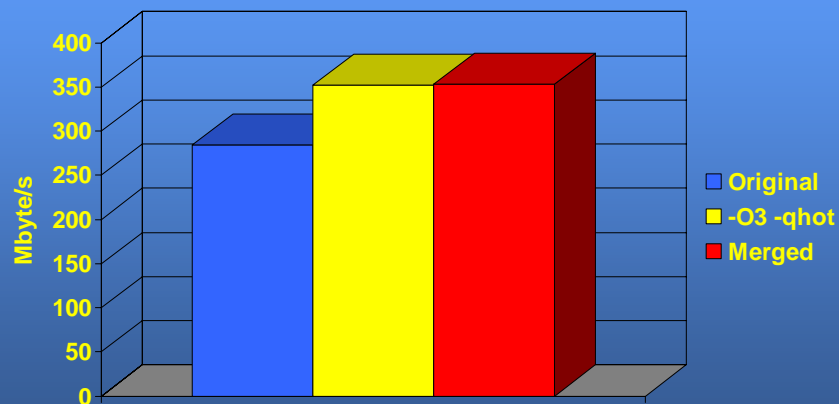
Four streams

83

© 2004 IBM Corporation



## Loop Merge Example



1.3 GHz POWER4

84

© 2004 IBM Corporation



## Fold Loops

- Increase number of streams at the expense of loop length

```
do i = 1,n  
  sum = sum +A(i)  
end do
```



```
do i = 1,n/4  
  sum = sum +A(i  
    + A(i+1*n/4)  
    + A(i+2*n/4)  
    + A(i+3*n/4)  
end do
```

One stream per loop

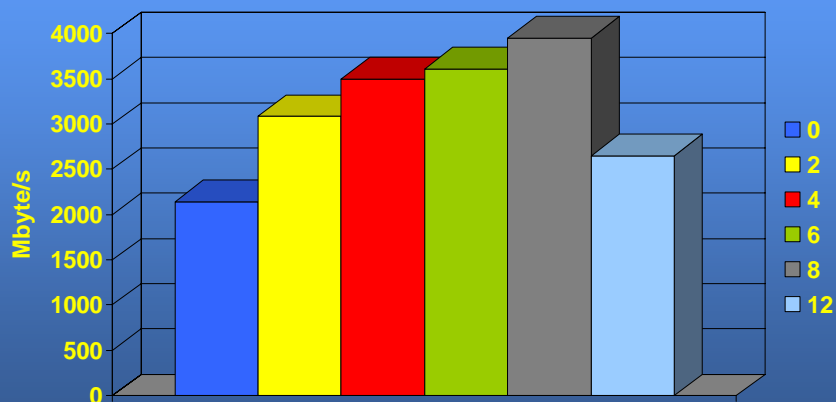
Four streams

85

© 2004 IBM Corporation



## Folding Example



1.3 GHz POWER4

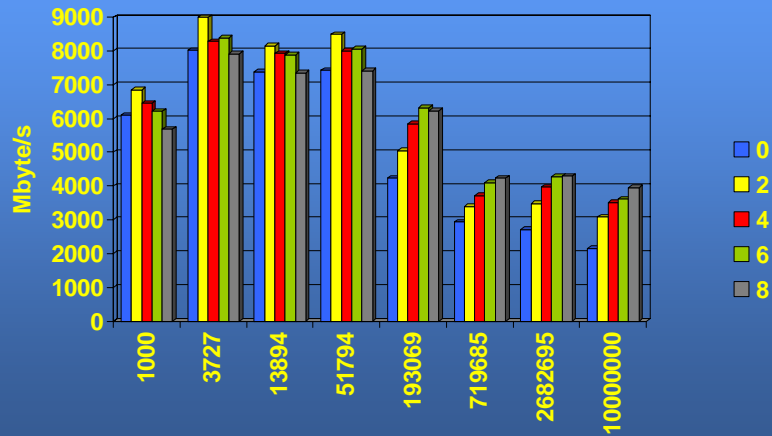
Single RHS; size 1000000

86

© 2004 IBM Corporation



## Folding Example



1.3 GHz POWER4

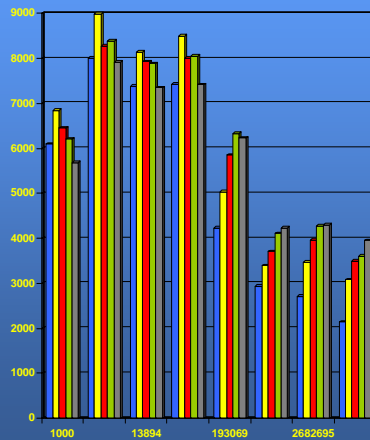
87

© 2004 IBM Corporation



## Folding Example

- Beneficial for long loops
  - Works at ~100,000 loads
- Do not fold past factor of 8



88

© 2004 IBM Corporation



## Stride Folding

- Fold loop to increase number outstanding cache loads:

```
do i = 1,n,m  
  sum = sum +A(i)  
end do
```



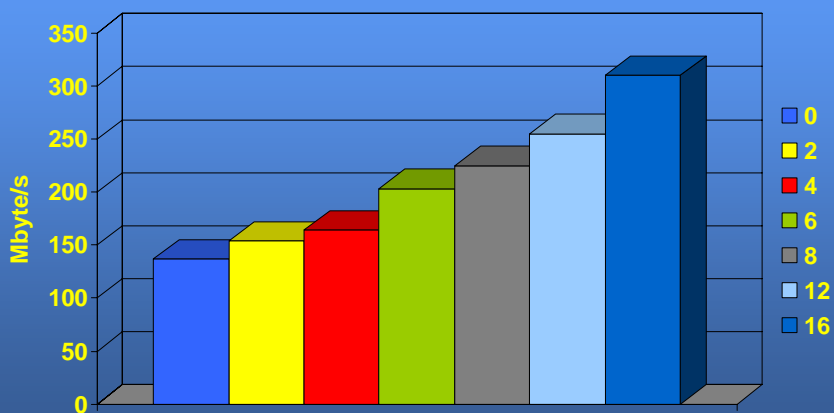
```
do i = 1,n/4,m  
  sum = sum +A(i )  
            + A(i+1*n/4)  
            + A(i+2*n/4)  
            + A(i+3*n/4)  
end do
```

89

© 2004 IBM Corporation



## Stride Folding Example



1.3 GHz POWER4

Single RHS; size 5,000,000, stride 32

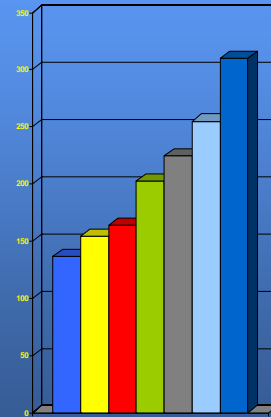
90

© 2004 IBM Corporation



## Stride Folding Example

- **Overlap out standing loads**
  - Not limited by number of stream buffers
- **More than 8 RHS**



91

© 2004 IBM Corporation



## Managing Pipelines

- **Deep pipeline functional units**
  - FMA
  - Divide
  - Square Root

92

© 2004 IBM Corporation



## Divide and Square Root

- POWER4 special functions:
  - Divide
  - Sqrt
- Use FMA functional unit
  - 2 simultaneous divide or sqrt (or rsqrt)
  - NOT pipelined

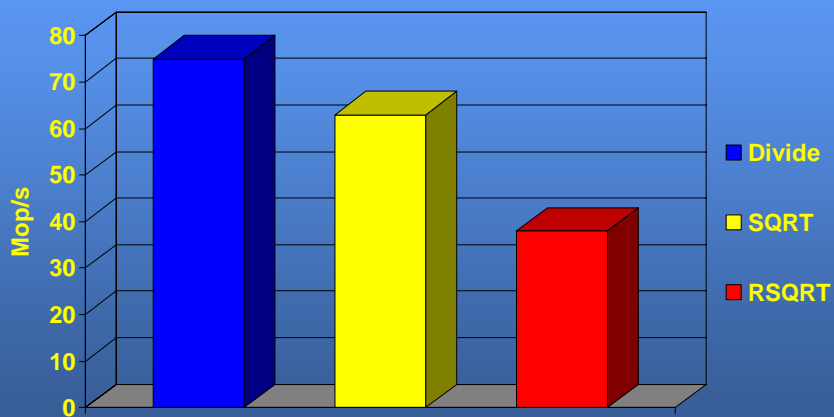
Instruction	Single	Double
Fma	6	6
Divide	32	32
Sqrt	38	38

93

© 2004 IBM Corporation



## Hardware DIV, SQRT, RSQRT



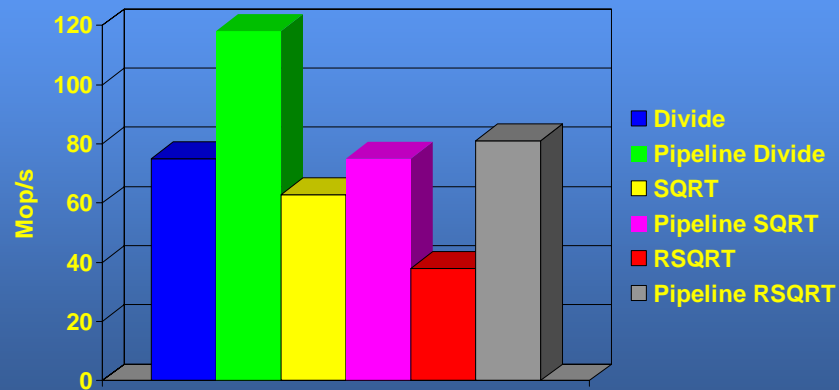
94

© 2004 IBM Corporation

1.3 GHz p690



## Hardware DIV, SQRT, RSQRT



1.3 GHz p690

## Example of Pipelined Functions

```
do i = -nbdy+3,n+nbdy-1
  prl = qrprl(i)
  pll = qrmrl(i)
  pavg = vtmp1(i)
  wllfac(i) = 5*gammp1*pavg + gamma * pll
  wrlfac(i) = 5*gammp1 * pavg +gamma *prl
  hrholl = rho(1,i-1)
  hrhorl = rho(1,i)
  wll(i) = 1/sqrt(hrholl * wllfac(i))
  wrl(i) = 1/sqrt(hrhorl * wrlfac(i))
end do
```

## Example of Pipelined Functions

```
allocate(t1,n+2*nbdy-3)
allocate(t2,n+2*nbdy-3)
do i = -nbdy+3,n+nbdy-1
  prl = qrprl(i)
  ...
  t1(i) =hrholl * wllfac(i)
  t2(i) =hrhorl * wrlfac(i)
end do
call __vrsqrt(t1,wrl,n+2*nbdy-3)
call __vrsqrt(t2,wll,n+2*nbdy-3)
```

97

© 2004 IBM Corporation



## Vectorization Analysis

- Dependencies
- Compiler overhead:
  - Generate (malloc) local temporary arrays
  - Extra memory traffic
- Moderate vector lengths required

	REC	SQRT	RSQRT
Cross Over	20	80	25
$N_{1/2}$	45	25	30

98

© 2004 IBM Corporation



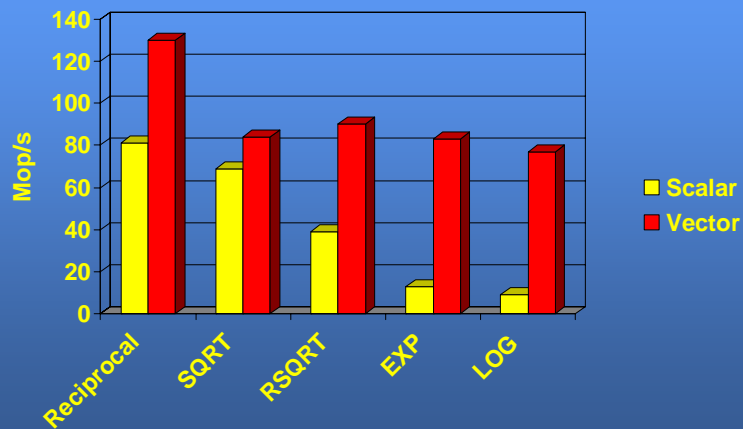
## Function Timings

Function	Hardware or Scalar Function		Pipelines	
	Clocks	Rate	Clocks	Rate
Reciprocal	32	81	20	130
SQRT	36	69	31	84
RSQRT	66	39	29	90
EXP	200	13	32	83
LOG	288	9	34	77

99  
© 2004 IBM Corporation



## Function Rates

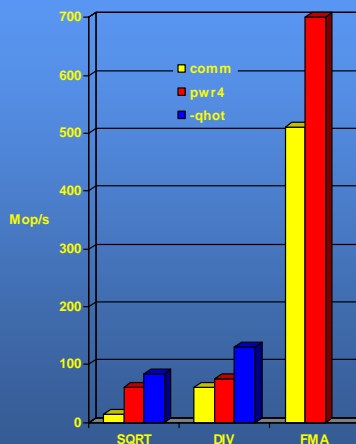


10  
© 2004 IBM Corporation



## SQRT

- POWER4 has hardware SQRT available
- Default -qarch=comm uses software library
- Use: -qarch=pwr4



1.1 GHz POWER4

10  
1  
© 2004 IBM Corporation

## Divide

- IEEE divide specifies actual divide
- Do not use multiply by reciprocal (default)
- Optimize with -O3

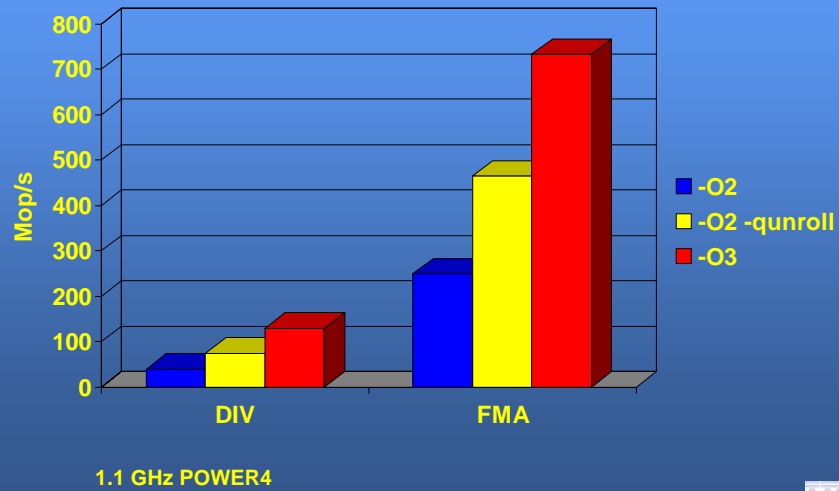
```
do i=1,n  
  B(i) = A(i)/s  
end do
```



```
rs=1/s  
do i=1,n  
  B(i) = A(i)*rs  
end do
```

10  
2  
© 2004 IBM Corporation

## Divide



10  
3  
© 2004 IBM Corporation

## Vector Intrinsic

- -qhot generates "vector" call to vector intrinsic functions
- Monitor with "-qreport=hotlist"

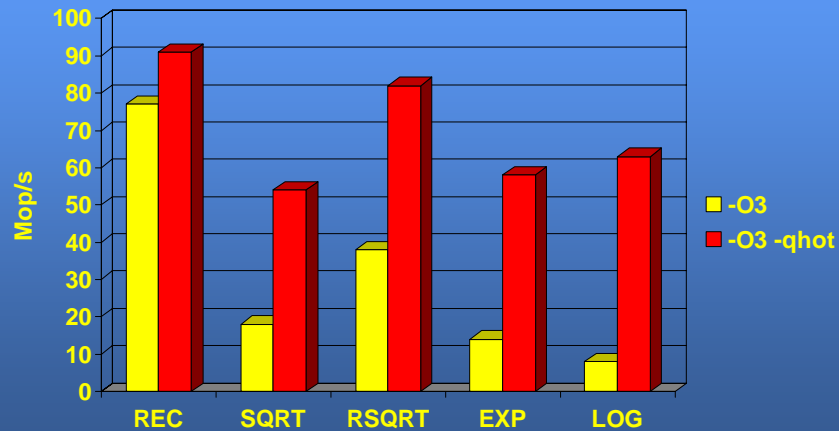
```
do i=1,n  
  B(i) = func(A(i))  
end do
```

→

```
call __vfunc(B,A,n)
```

10  
4  
© 2004 IBM Corporation

## Vector Intrinsic



1.3 GHz POWER4

10  
5  
© 2004 IBM Corporation



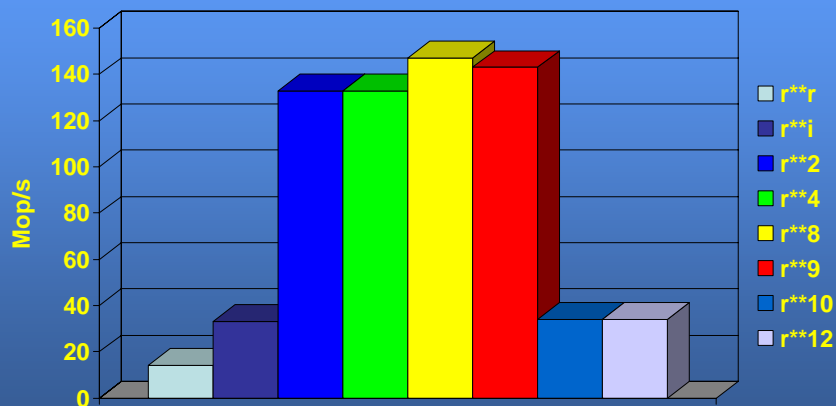
## Power Function (r to r)

- Computing power function:  $a^{**}b$
- if (  $b < 10$  and has integer value at compile time)
- Use unrolling and successive multiply
- else
- ... `__pow(...)`
- Test case:  
do i=1,n  
   $A(i)=B(i)^{**}b$   
end do

10  
6  
© 2004 IBM Corporation



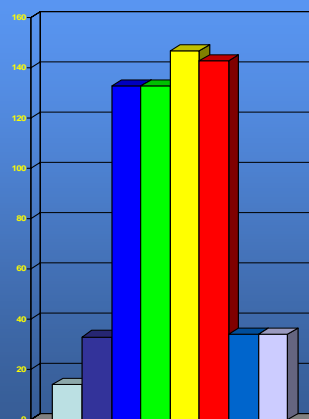
## Power Function



10  
7  
© 2004 IBM Corporation

## Power Function

- Compiler can transform integer power to multiply's
- Real to Real is expensive
- Real to Integer is less expensive



10  
8  
© 2004 IBM Corporation

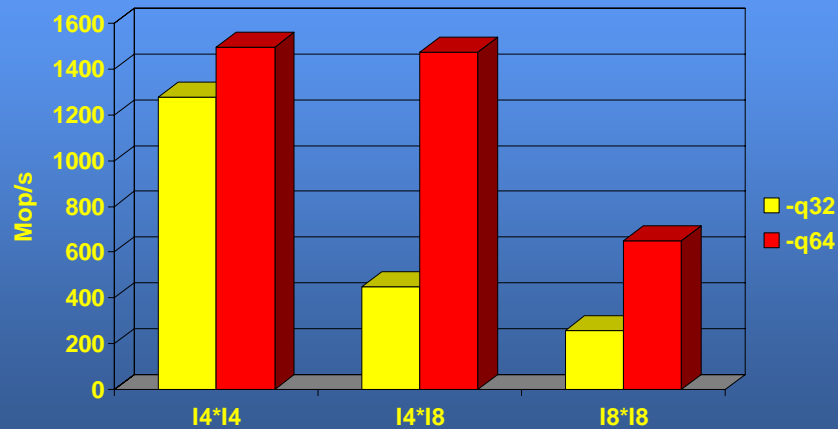
## 32-bit versus 64-bit Addressing

- 64-bit address mode enable use of 64-bit integer arithmetic
- Integer Arithmetic, especially (kind=8), or long, is much faster with -q64

10  
9  
© 2004 IBM Corporation



## Integer Arithmetic



11  
0  
© 2004 IBM Corporation



## 32-bit Floating Point Arithmetic

- Faster
- Less bandwidth required
- Arithmetic operations are same speed as 64-bit
- More efficient use of cache

