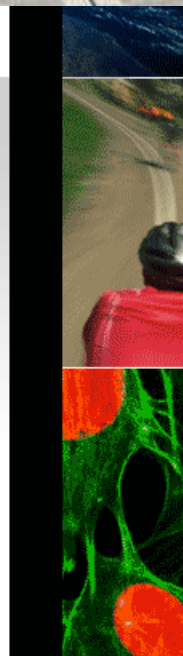
Optimization for the Cray XT4™ MPP Supercomputer

John M. Levesque

May, 2008

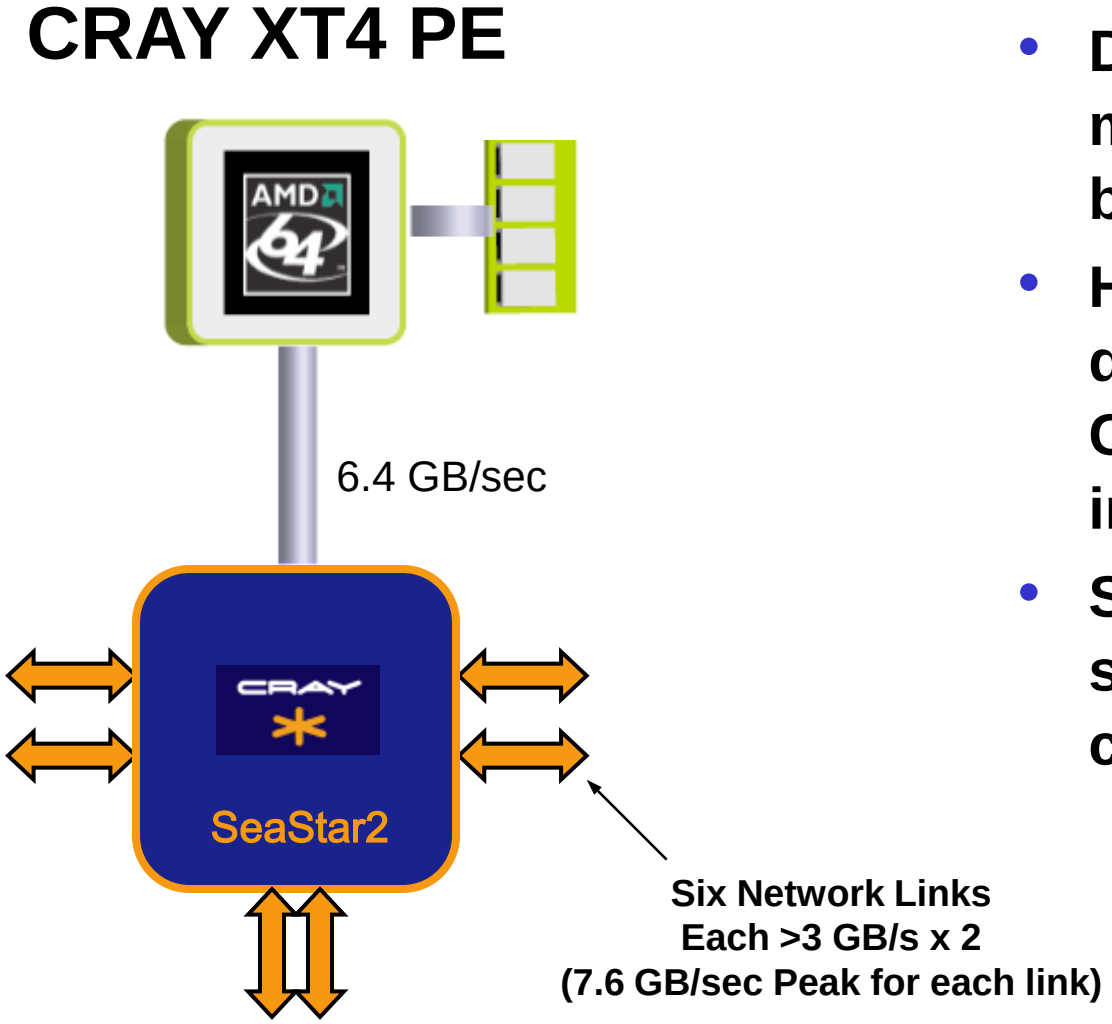
The Cray XT4 System



Recipe for a good MPP

1. Select Best Microprocessor
2. Surround it with a balanced or "bandwidth rich" environment
3. "Scale" the System
   - Eliminate Operating System Interference (OS Jitter)
   - Design in Reliability and Resiliency
   - Provide Scaleable System Management
   - Provide Scalable I/O
   - Provide Scalable Programming and Performance Tools
   - System Service Life (provide an upgrade path)

AMD Opteron: Why we selected it

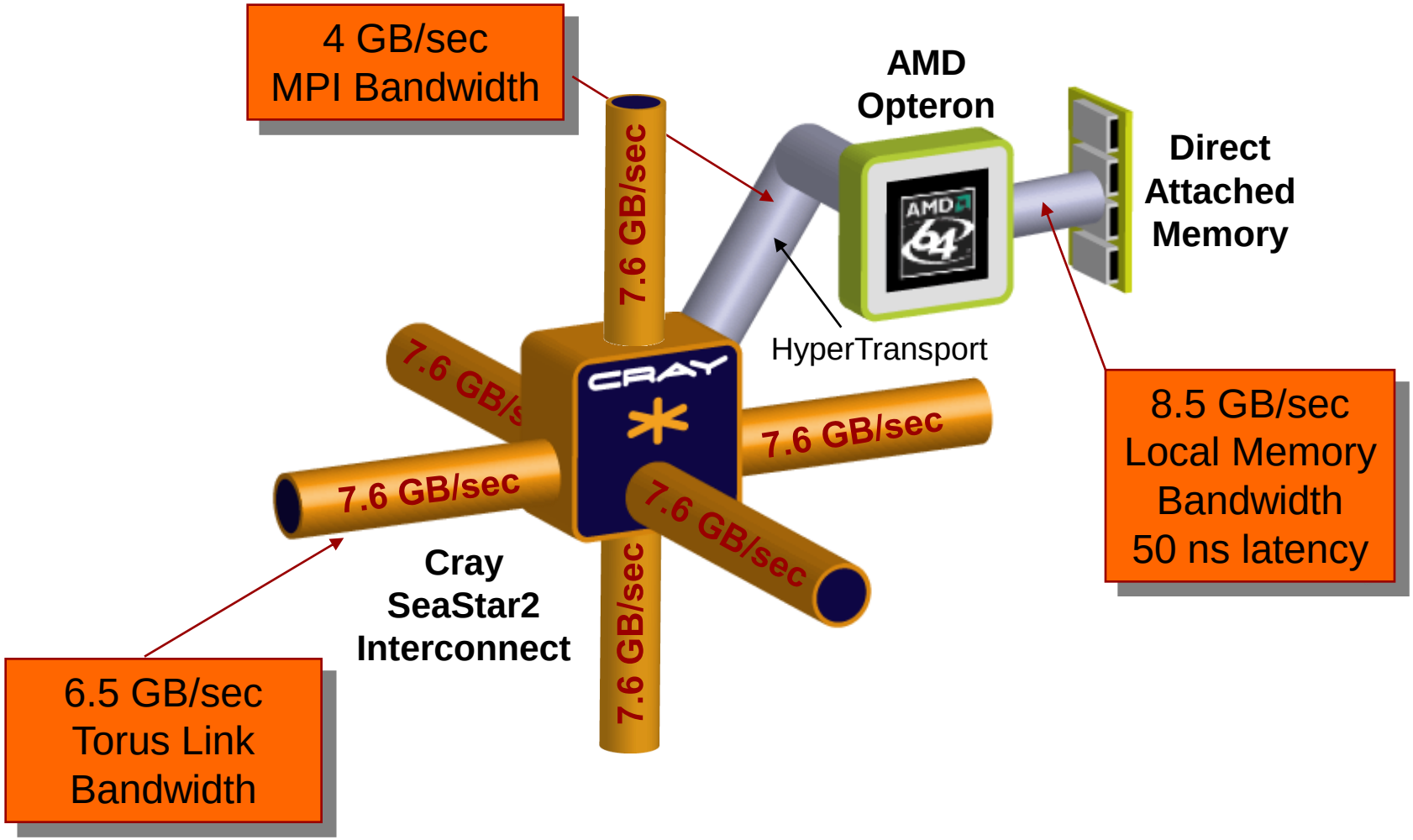


- **Direct attached local memory for leading bandwidth and latency**
- **HyperTransport can be directly attached to Cray SeaStar2 interconnect**
- **Simple two-chip design saves power and complexity**

Recipe for a good MPP

1. Select Best Microprocessor
2. Surround it with a balanced or "bandwidth rich" environment
3. "Scale" the System
   - Eliminate Operating System Interference (OS Jitter)
   - Design in Reliability and Resiliency
   - Provide Scalable System Management
   - Provide Scalable I/O
   - Provide Scalable Programming and Performance Tools
   - System Service Life (provide an upgrade path)

The Cray XT4 Processing Element: Providing a bandwidth-rich environment

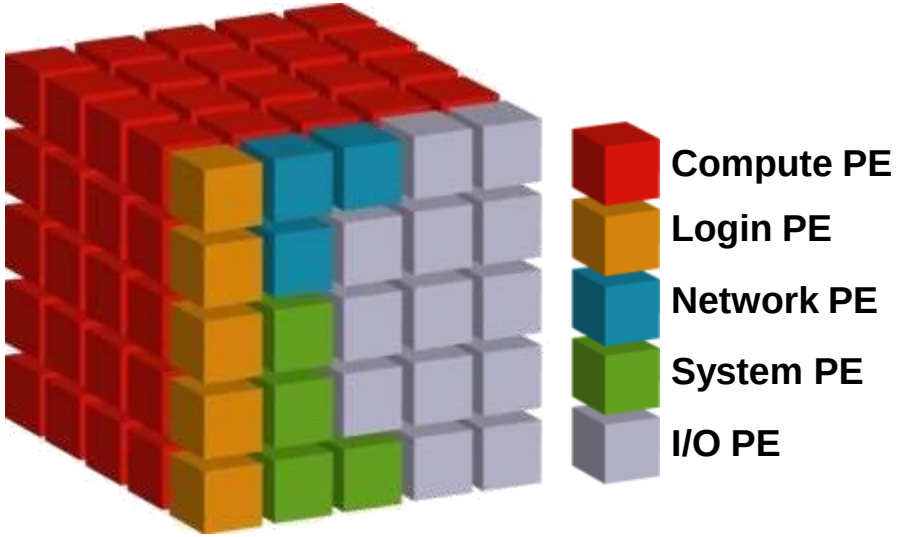


Recipe for a good MPP

1. Select Best Microprocessor
2. Surround it with a balanced or "bandwidth rich" environment
3. "Scale" the System
   - Eliminate Operating System Interference (OS Jitter)
   - Design in Reliability and Resiliency
   - Provide Scalable System Management
   - Provide Scalable I/O
   - Provide Scalable Programming and Performance Tools
   - System Service Life (provide an upgrade path)

Scalable Software Architecture: UNICOS/lc

"Primum non nocere"



Service Partition

Specialized Linux nodes

- Microkernel on Compute PEs, full featured Linux on Service PEs.
- Service PEs specialize by function
- Software Architecture eliminates OS "Jitter"
- Software Architecture enables reproducible run times
- Large machines boot in under 30 minutes, including filesystem

This is the real reason the XT4 will scale to a Petaflop

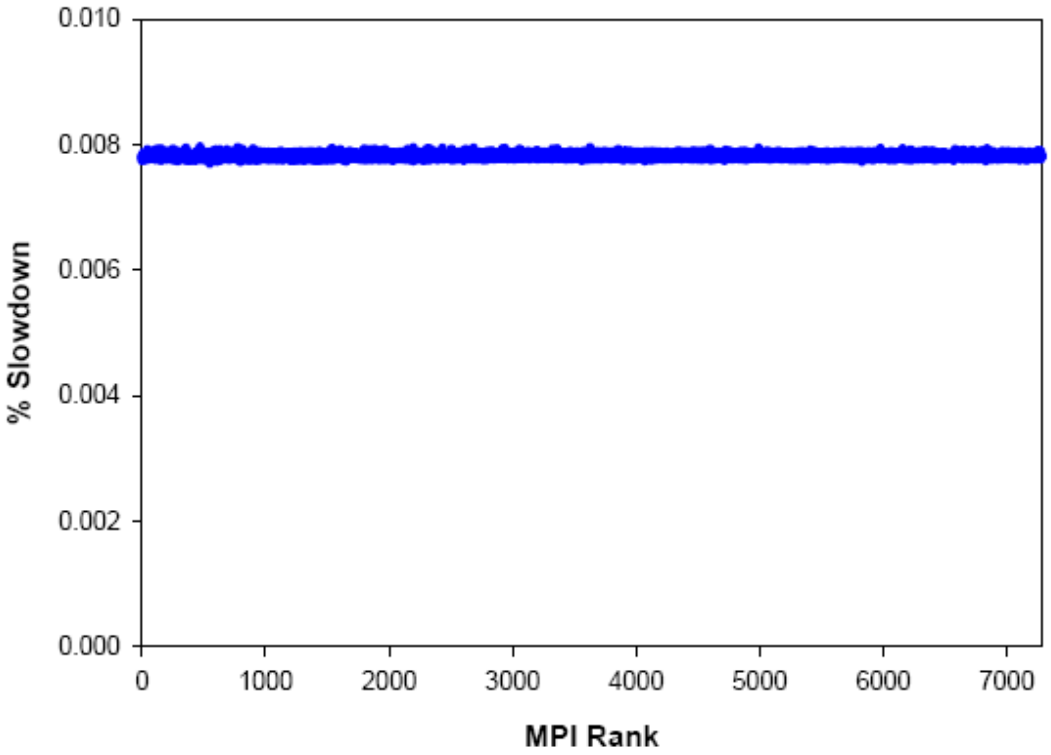


Figure 3: Slowdown caused by computational "noise"

Download P-SNAP from the web and try it on your system

AMD Core Optimizations

- Cache Optimizations
  - Cache Associativity and Cache Thrashing
  - Cache Alignment
  - Cache Blocking
  - OpenMP
- Prefetching
  - Types of Prefetching
  - When to use
- Programmatic Examples
  - Striding
  - Function Calls
  - Importance of Vectorization
  - Matrix Multiplies
  - Loop Optimizations
- Things to remember

Let's Review: Dual Core v. Quad Core

Dual Core

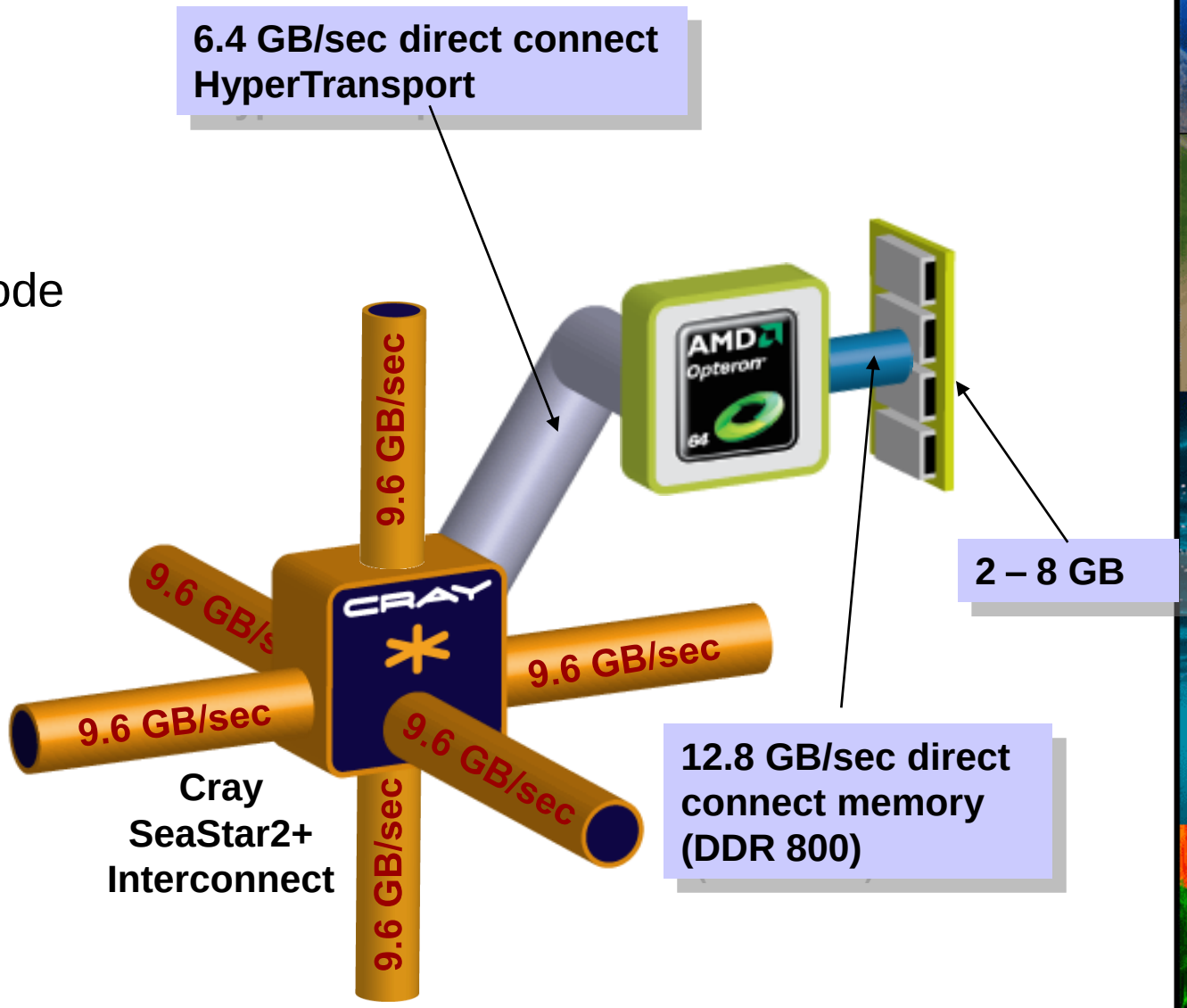
- Core
  - 2.6Ghz clock frequency
  - SSE SIMD FPU (2flops/cycle = 5.2GF peak)
- Cache Hierarchy
  - L1 Dcache/Icache: 64k/core
  - L2 D/I cache: 1M/core
  - SW Prefetch and loads to L1
  - Evictions and HW prefetch to L2
- Memory
  - Dual Channel DDR2
  - 10GB/s peak @ 667MHz
  - 8GB/s nominal STREAMs

Quad Core

- Core
  - 2.1Ghz clock frequency
  - SSE SIMD FPU (4flops/cycle = 8.4GF peak)
- Cache Hierarchy
  - L1 Dcache/Icache: 64k/core
  - L2 D/I cache: 512 KB/core
  - L3 Shared cache 2MB/Socket
  - SW Prefetch and loads to L1,L2,L3
  - Evictions and HW prefetch to L1,L2,L3
- Memory
  - Dual Channel DDR2
  - 12GB/s peak @ 800MHz
  - 10GB/s nominal STREAMs

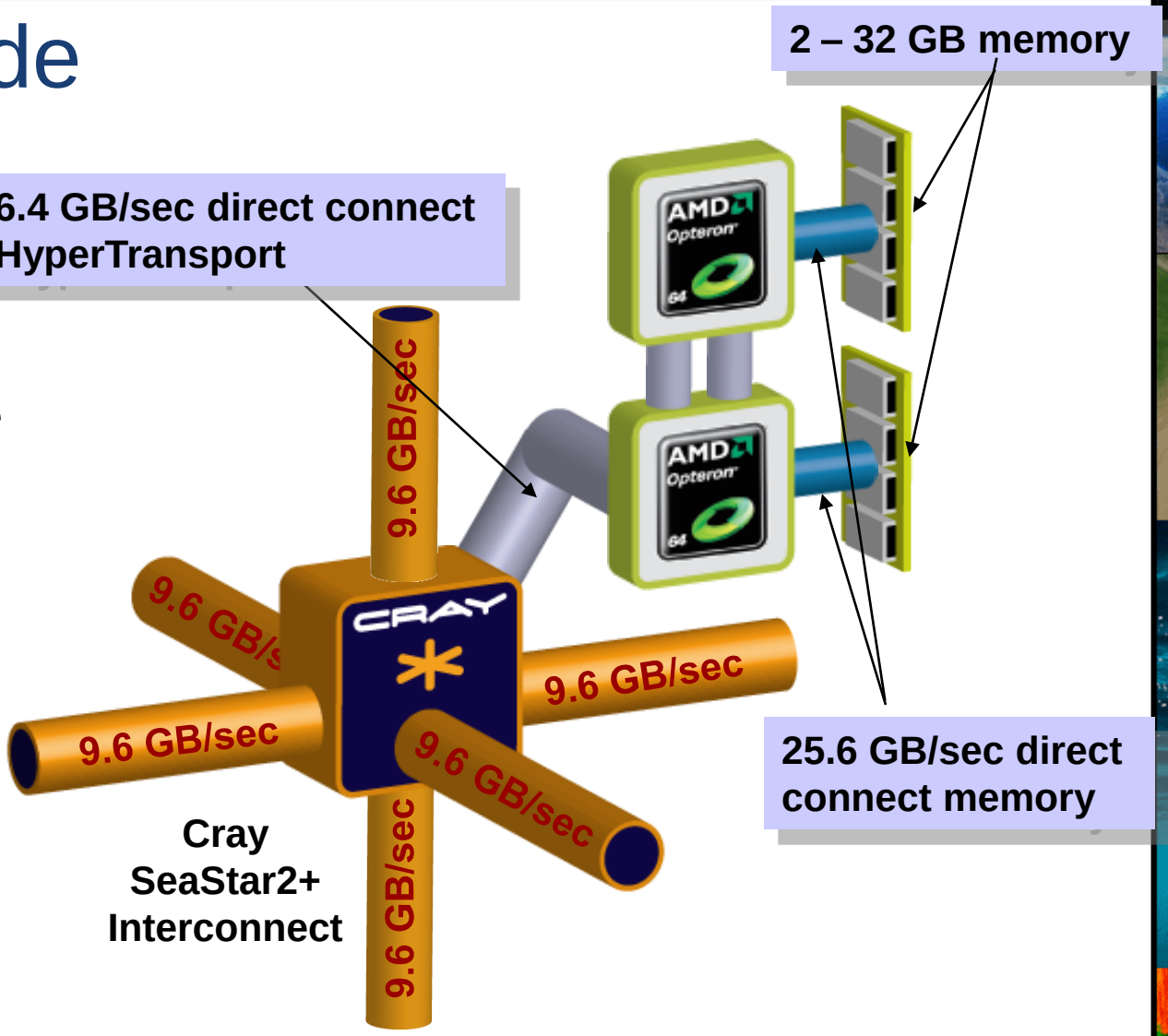
Cray XT4 Node

- 4-way SMP
- >35 Gflops per node
- Up to 8 GB per node
- OpenMP Support within socket



Cray XT5 Node

- 8-way SMP
- >70 Gflops per node
- Up to 32 GB of shared memory per node
- OpenMP Support



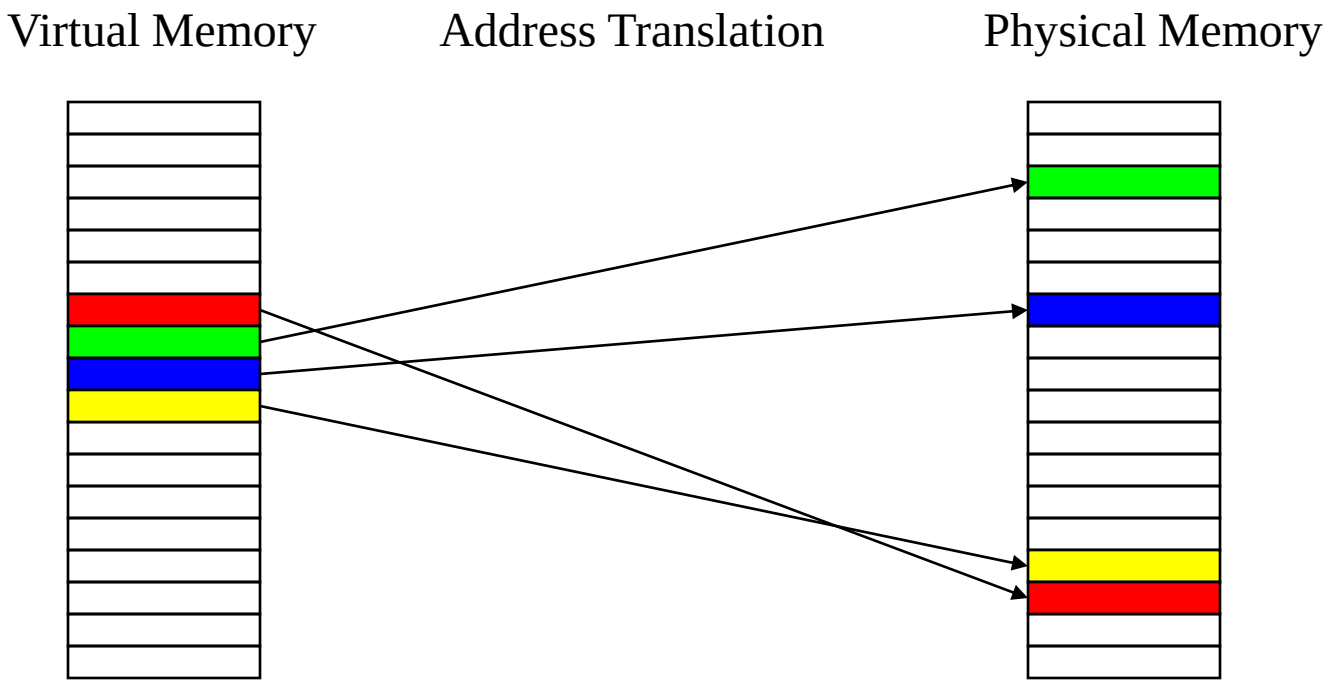
Quad Core, More to the Point

- Doubled flops/clock, only if you use SSE128
  - Very-Short-Vector Instructions
- Clock reduced from 2.6GHz – 2.2 GHz
  - 18% clock speed reduction
- L2 Cache size has been reduced per-core, but shared L3 has been added
  - Essentially, more cache visible to the cores
- DDR2-800 memory has replaced DDR2-667
  - 10.6GB/s -> 12.8GB/s (21% improvement)
  - More total memory bandwidth and 2X as many memory controllers, also 2X as many cores to use the BW.

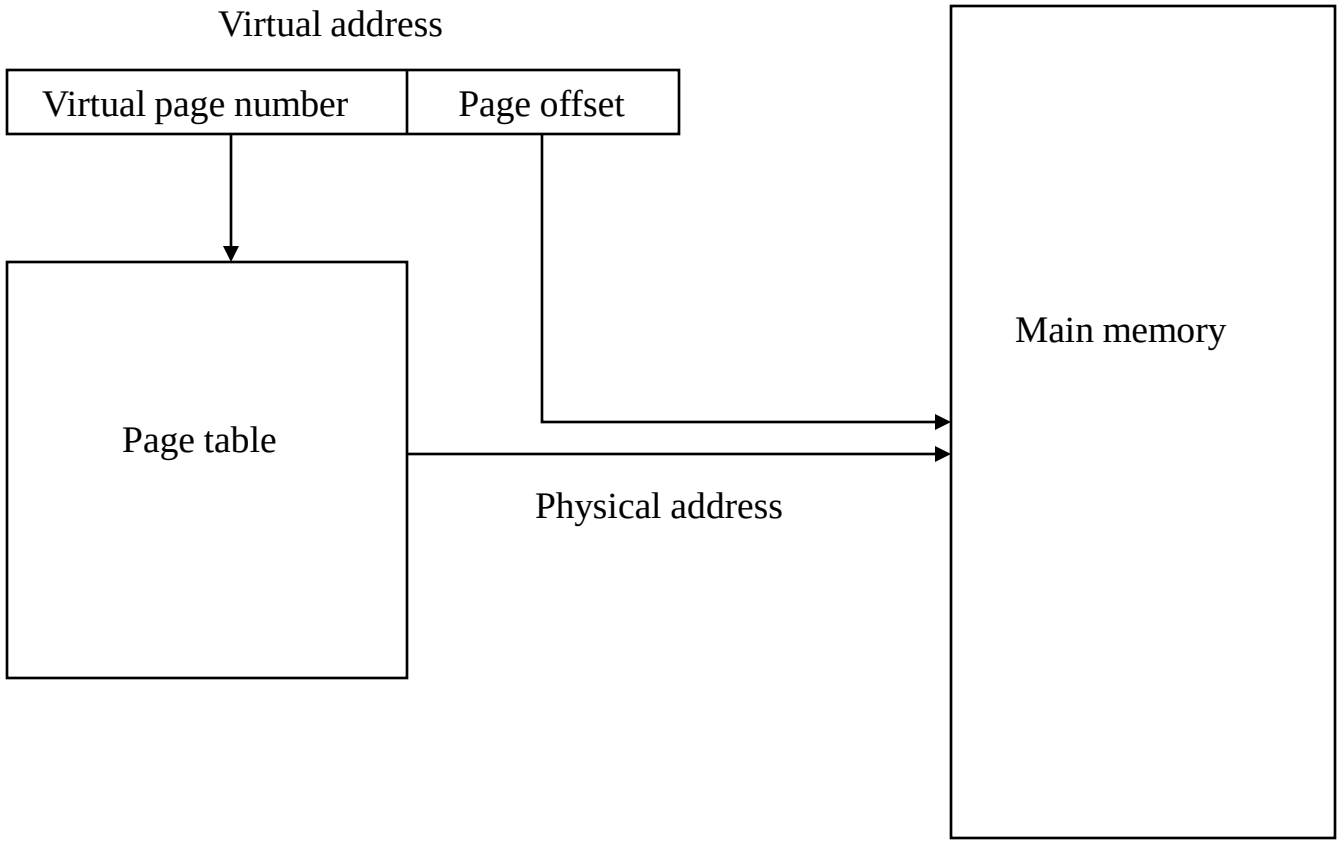
Background: Virtual Memory

- Modern programs operate in “virtual memory”
  - Each program thinks it has all of memory to itself
  - Fixed sized blocks (“pages”) vs variable sized blocks (“segments”)
- Virtual Memory benefits
  - Allow a program that is larger than physical memory to run
    - Programmer does not have to manually create overlays
  - Allow many programs to share limited physical memory
- Virtual Memory problems
  - Each virtual memory reference must be translated into a physical memory reference

Virtual Memory vs Physical Memory



Address Translation



Source: Computer Architecture A Quantitative Approach, by John L. Hennessy and David A. Patterson

Translation Speed

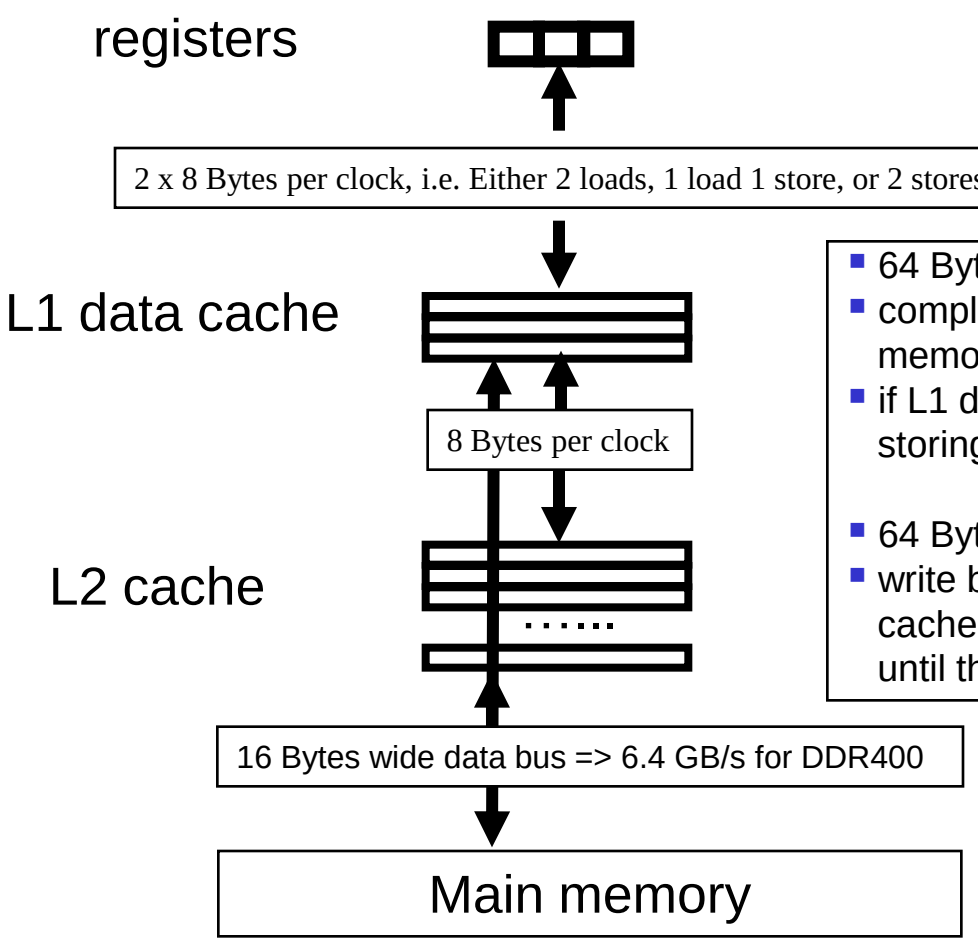
- Translation page table is stored in main memory
  - Each memory access logically takes twice as long – once to find the physical address, once to get the actual data
- Use a hardware cache of least recently used addresses
  - Called a Translation Lookaside Buffer or TLB

Other Quad-core improvements

- 2MB memory pages no longer limited to an 8 entry TLB
  - 2MB pages are not currently supported on XT, but support being tested and will be released soon
- 1GB memory pages
  - Cray currently has no plans to support these
- Improved Out-of-Order loads/stores
- Improved branch prediction

CACHE OPTIMIZATIONS

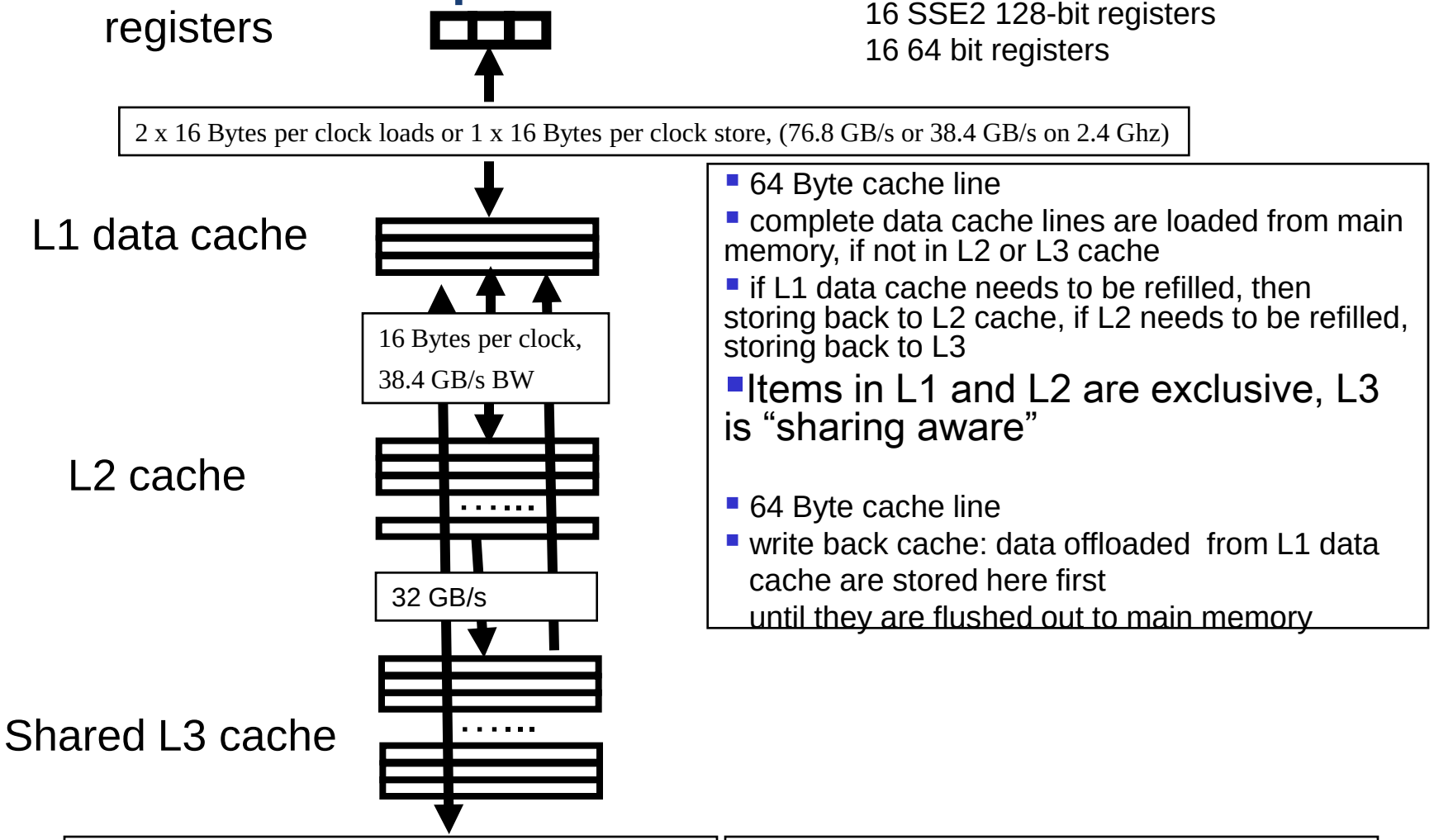
Simplified memory hierachy on the AMD Opteron



- 64 Byte cache line
- complete data cache lines are loaded from main memory, if not in L2 cache
- if L1 data cache needs to be refilled, then storing back to L2 cache

- 64 Byte cache line
- write back cache: data offloaded from L1 data cache are stored here first until they are flushed out to main memory

Simplified memory hierachy on the Quad Core AMD Opteron

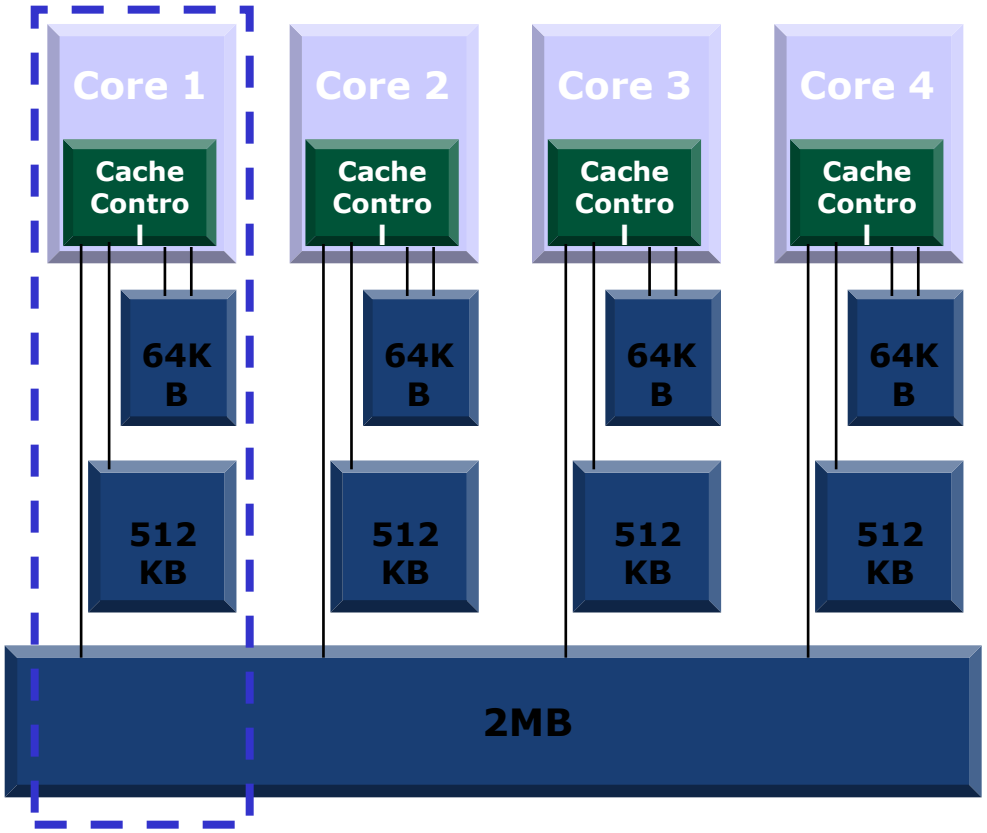


- 64 Byte cache line
- complete data cache lines are loaded from main memory, if not in L2 or L3 cache
- if L1 data cache needs to be refilled, then storing back to L2 cache, if L2 needs to be refilled, storing back to L3
- Items in L1 and L2 are exclusive, L3 is “sharing aware”

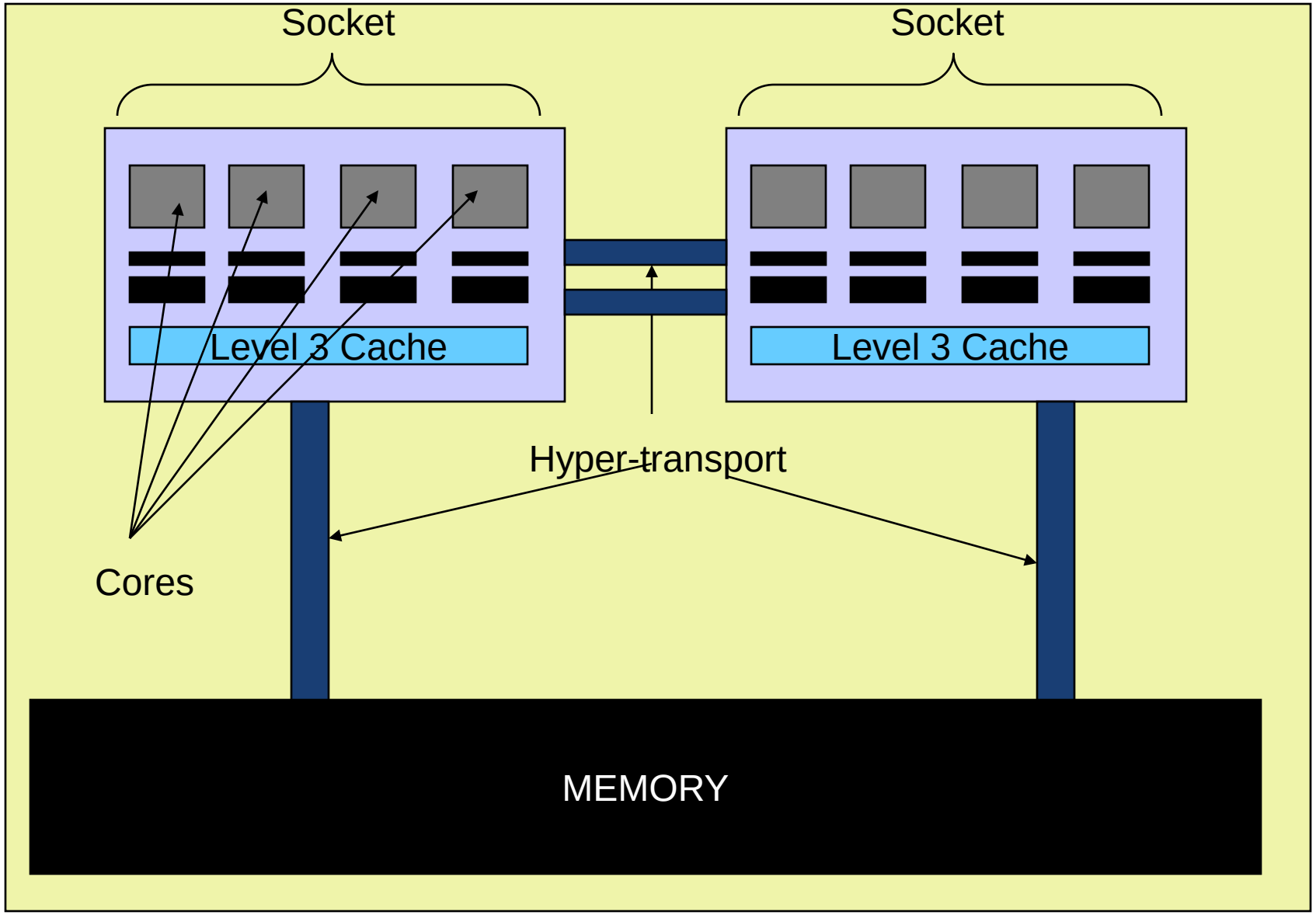
- 64 Byte cache line
- write back cache: data offloaded from L1 data cache are stored here first until they are flushed out to main memory

Cache Hierarchy

- Dedicated L1 cache
  - 2 way associativity.
  - 8 banks.
  - 2 128bit loads per cycle.
- Dedicated L2 cache
  - 16 way associativity.
- Shared L3 cache
  - fills from L3 leave likely shared lines in L3.
  - sharing aware replacement policy.



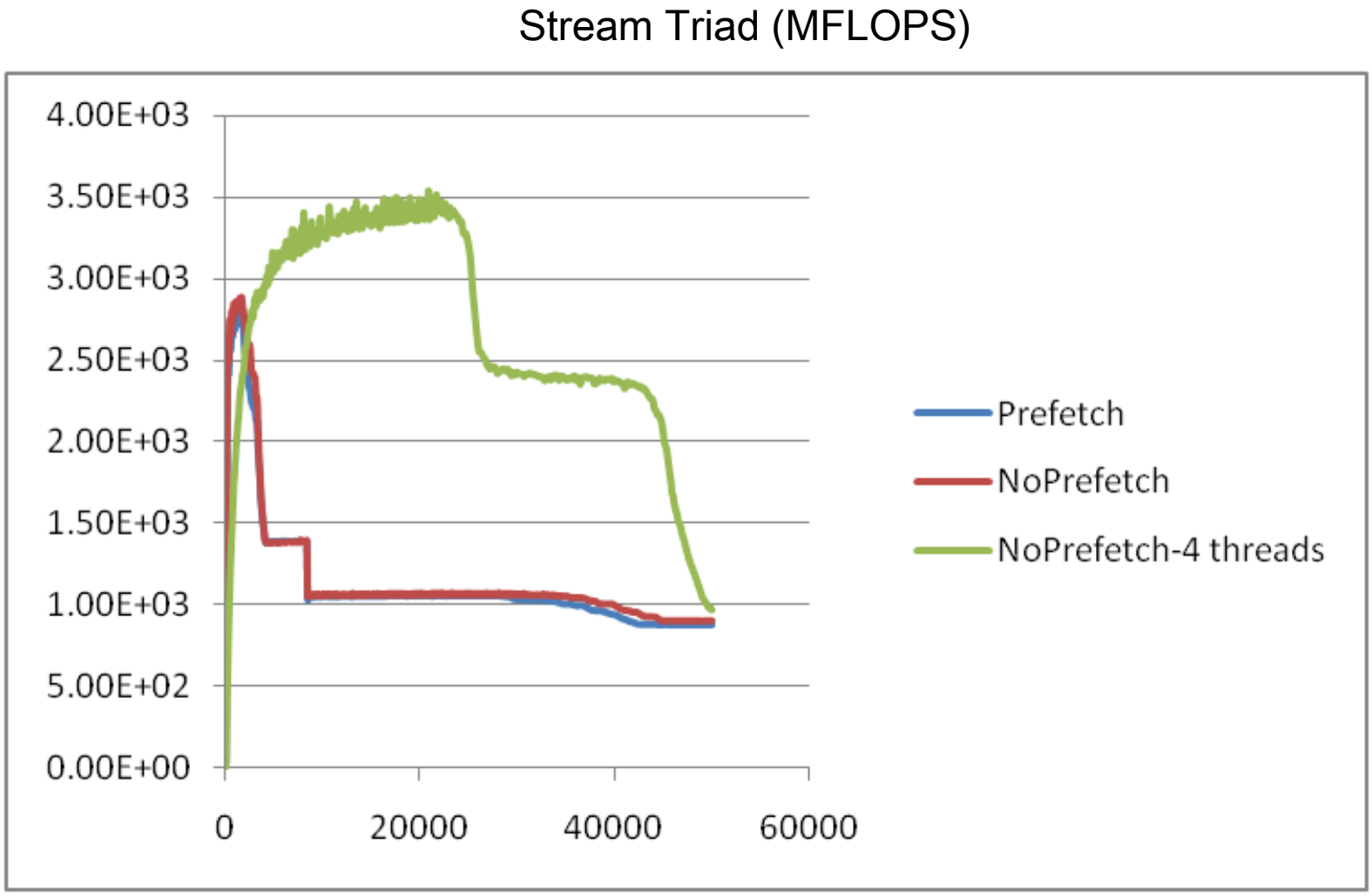
The Barcelona Node (XT5)



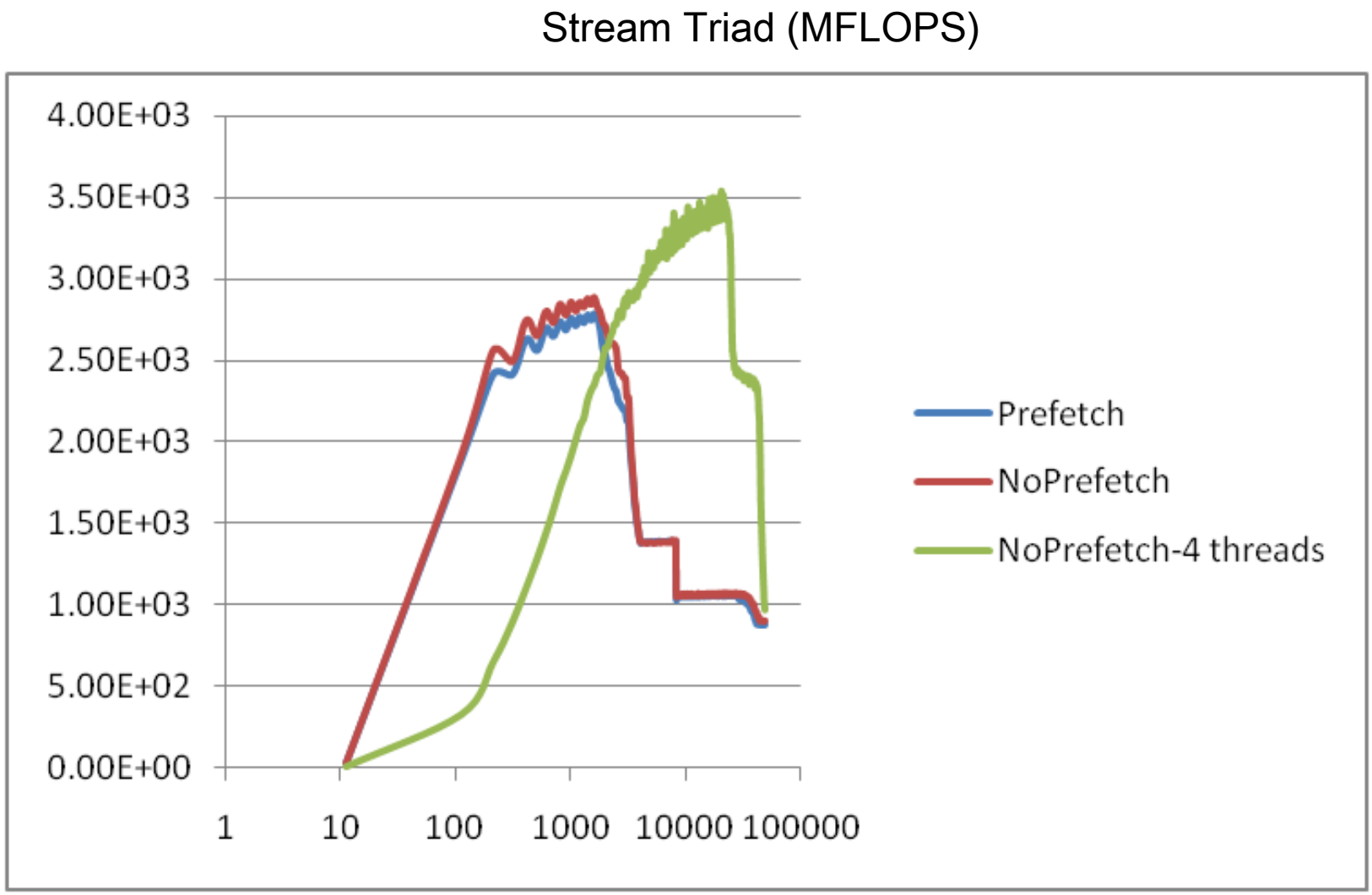
Memory Alignment Issues

- Cache Boundaries
- Page Boundaries
- Memory Banks

Performance = F(Cache Utilization)

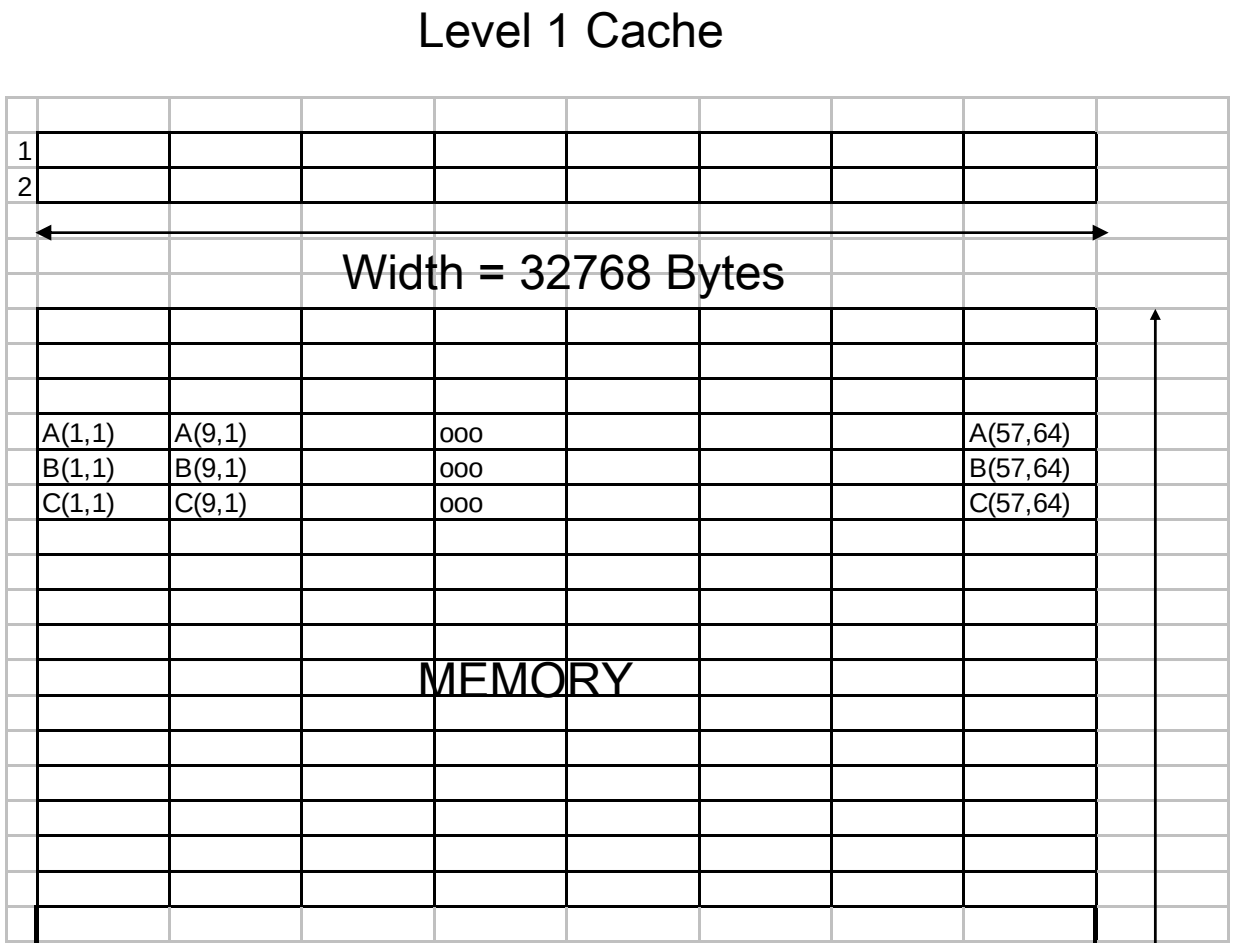


Performance = F(Cache Utilization)



Real * 8	A(64,64),B(64,64),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					

Cache Visualization



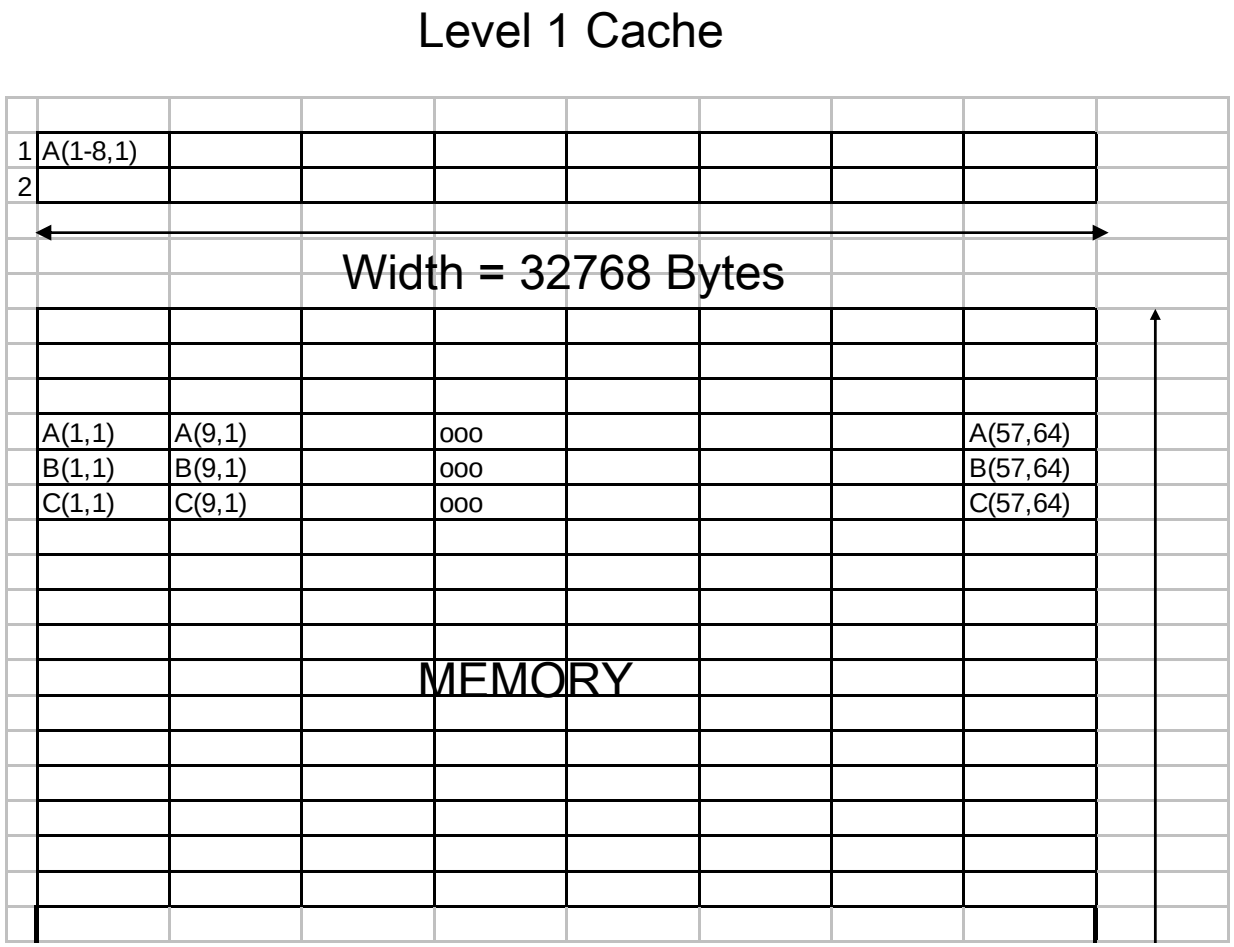
Level 1 Cache
- 65536 B
- 1024 Lines
- 8192 8B Ws
- 16384 4B Ws
- 2 way Assoc

Associativity Class
- 32768 B
- 512 Lines
- 4096 8B Ws
- 8192 4B Ws

64*64*8 = 32768 B

Consider the following example

Real * 8	A(64,64),B(64,64),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					
Fetch A(1,1)		Fetch from M	Uses 1 Associativity Class		

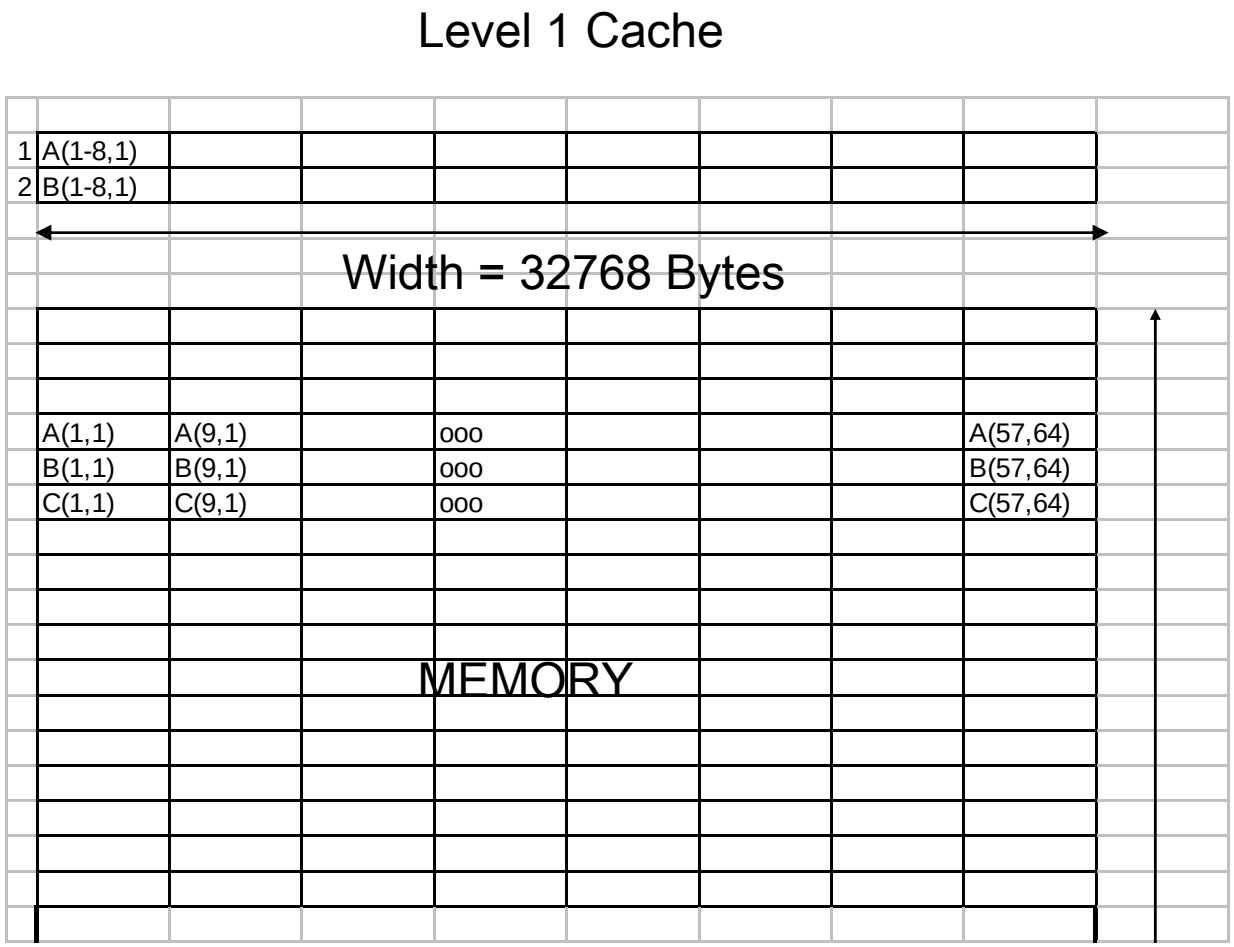


Level 1 Cache
- 65536 B
- 1024 Lines
- 8192 8B Ws
- 16384 4B Ws
- 2 way Assoc

Associativity Class
- 32768 B
- 512 Lines
- 4096 8B Ws
- 8192 4B Ws

64*64*8 = 32768 B

Real * 8	A(64,64),B(64,64),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					
Fetch A(1,1)		Fetch from M	Uses 1 Associativity Class		
Fetch B(1,1)		Fetch from M	Uses 2 Associativity Class		

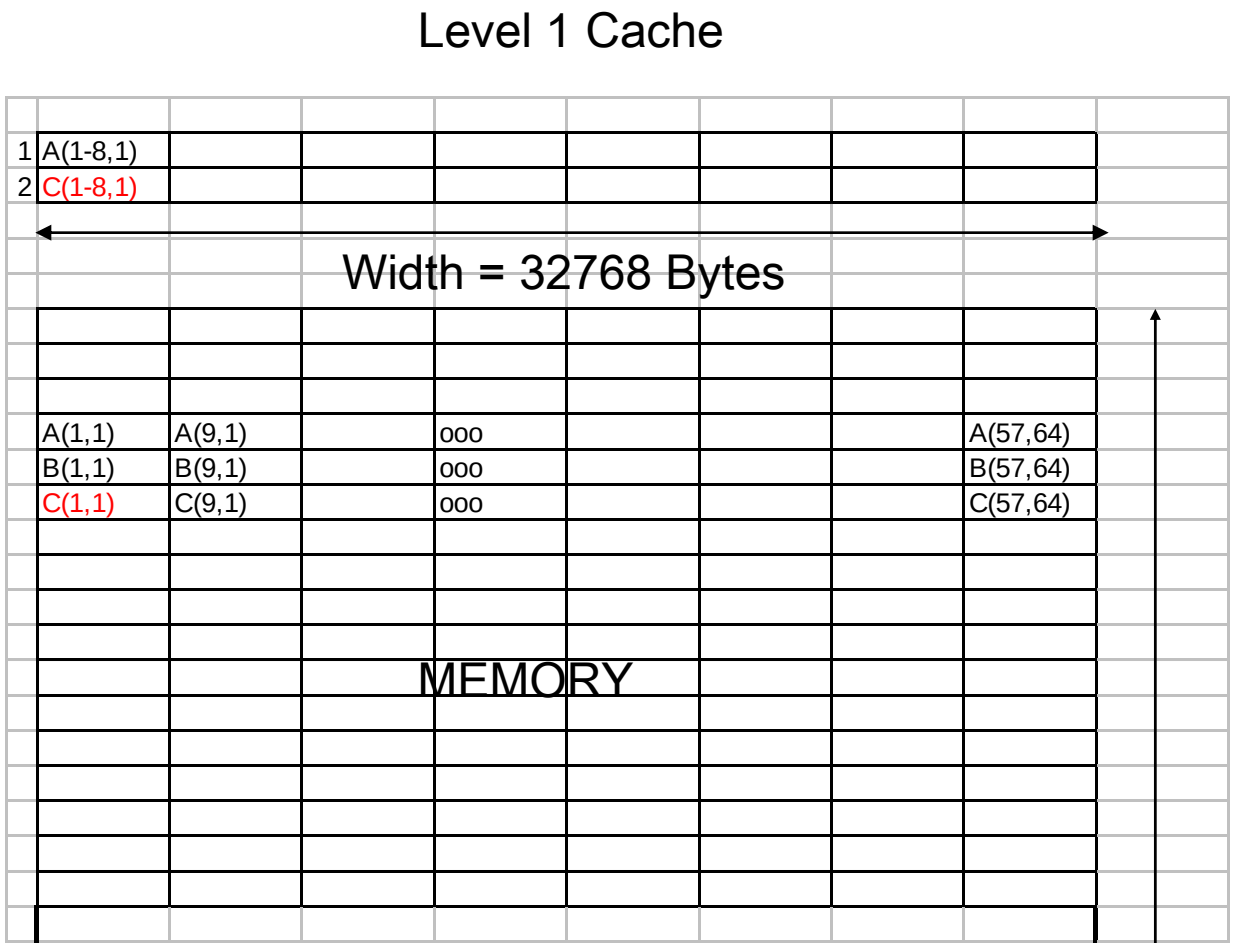


Level 1 Cache
- 65536 B
- 1024 Lines
- 8192 8B Ws
- 16384 4B Ws
- 2 way Assoc

Associativity Class
- 32768 B
- 512 Lines
- 4096 8B Ws
- 8192 4B Ws

64*64*8 = 32768 B

Real * 8	A(64,64),B(64,64),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					
Fetch A(1,1)		Fetch from M	Uses 1 Associativity Class		
Fetch B(1,1)		Fetch from M	Uses 2 Associativity Class		
Add A(1,1) + B(1,1)					
Store C(1,1)		Fetch from M	Overwrites either 1 or 2 Associativity Class		



Level 1 Cache
- 65536 B
- 1024 Lines
- 8192 8B Ws
- 16384 4B Ws
- 2 way Assoc

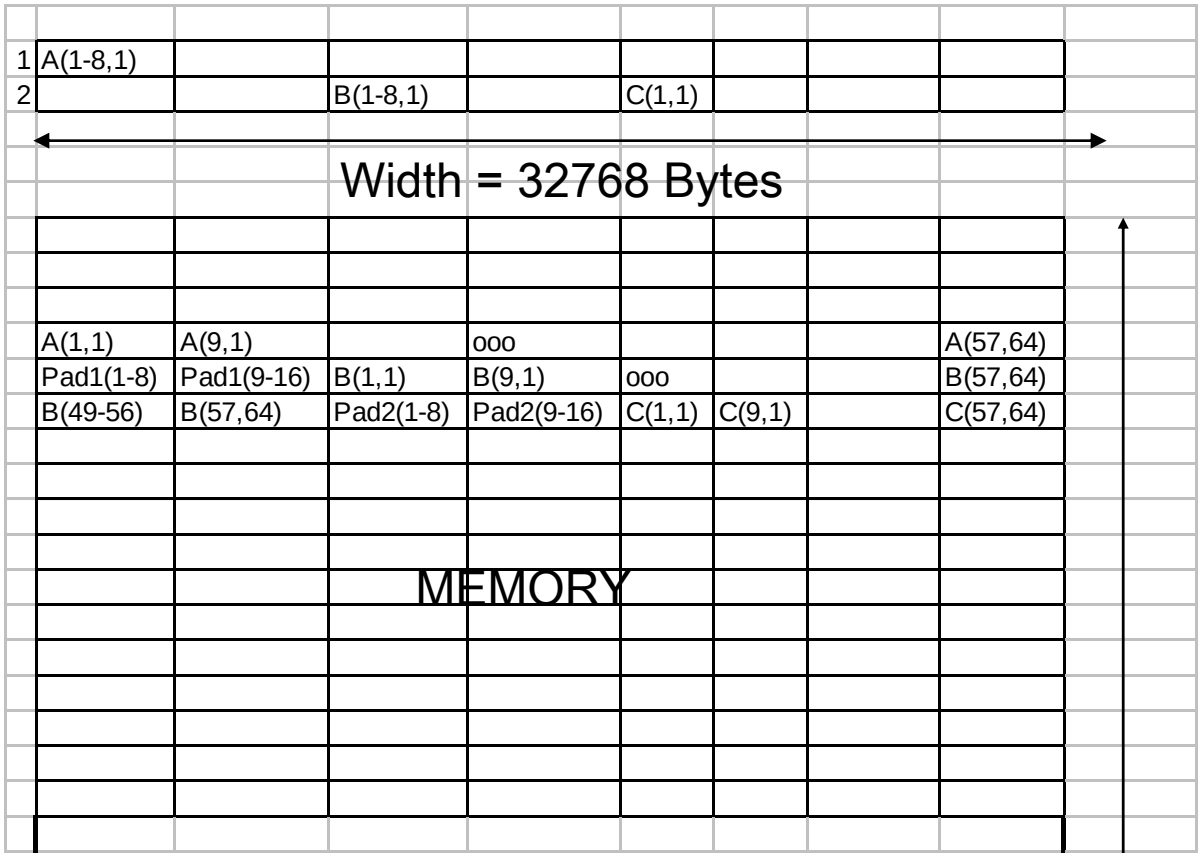
Associativity Class
- 32768 B
- 512 Lines
- 4096 8B Ws
- 8192 4B Ws

64*64*8 = 32768 B

Real * 8	A(64,64),B(64,64),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					
Fetch A(1,1)		Fetch from M	Uses 1 Associativity Class		
Fetch B(1,1)		Fetch from M	Uses 2 Associativity Class		
Add A(1,1) + B(1,1)					
Store C(1,1)		Fetch from M	Overwrites either 1 or 2 Associativity Class		
Fetch A(2,1)		Fetch from L2	Overwrites either 1 or 2 Associativity Class		
Fetch B(2,1)		Fetch from L2	Overwrites either 1 or 2 Associativity Class		
Add A(2,1) + B(2,1)					
Store C(2,1)		Fetch from L2	Overwrites either 1 or 2 Associativity Class		

Must be a better Way

Real * 8	**A(64,64),pad1(16),B(64,64),pad2(16),C(64,64)**				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					

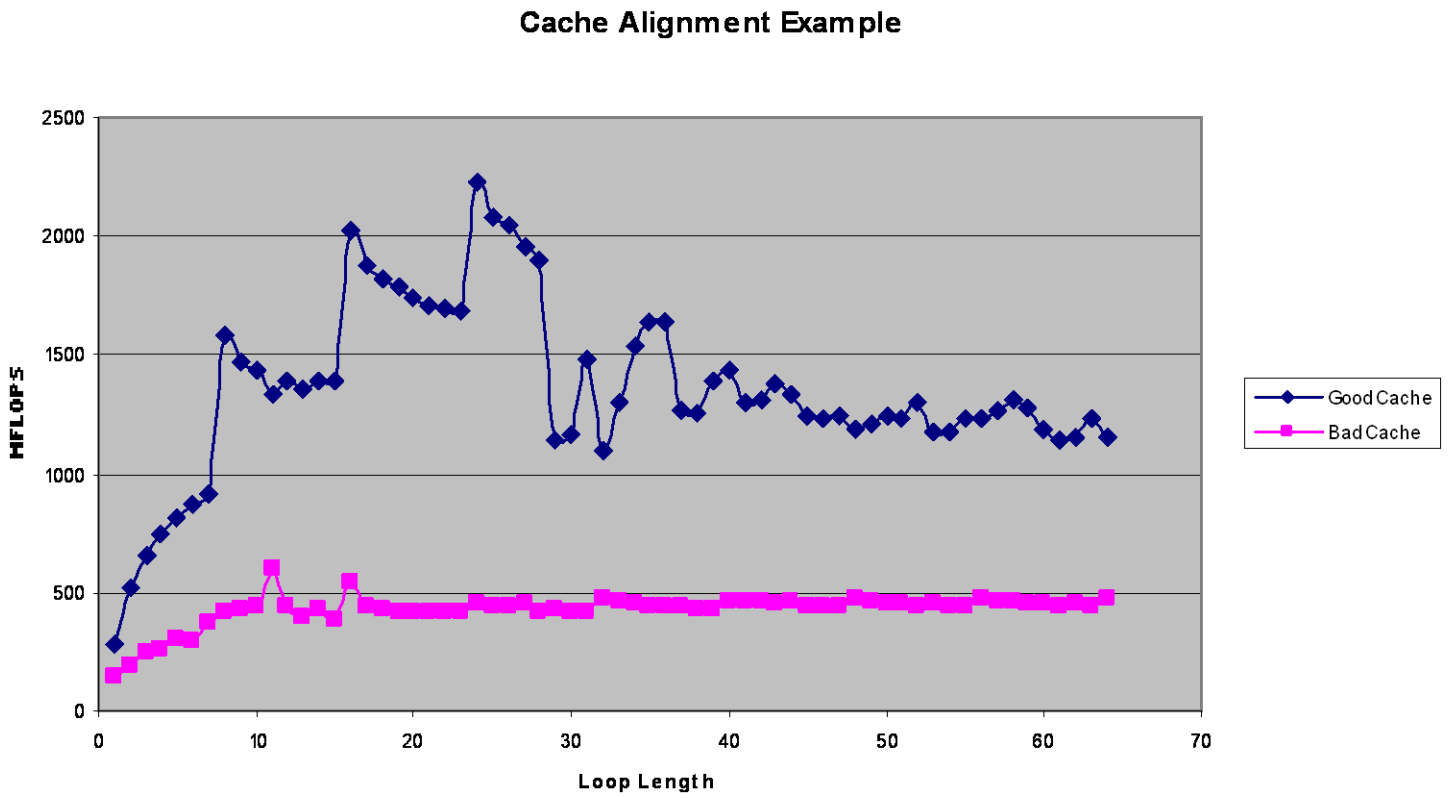


Level 1 Cache
- 65536 B
- 1024 Lines
- 8192 8B Ws
- 16384 4B Ws
- 2 way Assoc

Associativity Class
- 32768 B
- 512 Lines
- 4096 8B Ws
- 8192 4B Ws

64*64*8 = 32768 B

Real * 8	A(64,64),pad1(16),B(64,64),pad2(16),C(64,64)				
DO I = 1,N					
C(I,1) = A(I,1) +B(I,1)					
ENDDO					
Fetch A(1)		Uses 1 Associativity Class			
Fetch B(1)		Uses 2 Associativity Class			
Add A(1) + B(1)					
Store C(1)		Uses 1 Associativity Class			
Fetch A(2)		Gets from L1 Cache			
Fetch B(2)		Gets from L1 Cache			
Add A(2) + B(2)					
Store C(2)		Gets from L1 Cache			


Cache Alignment Example
MFLOPS
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
Loop Length
Good Cache
Bad Cache

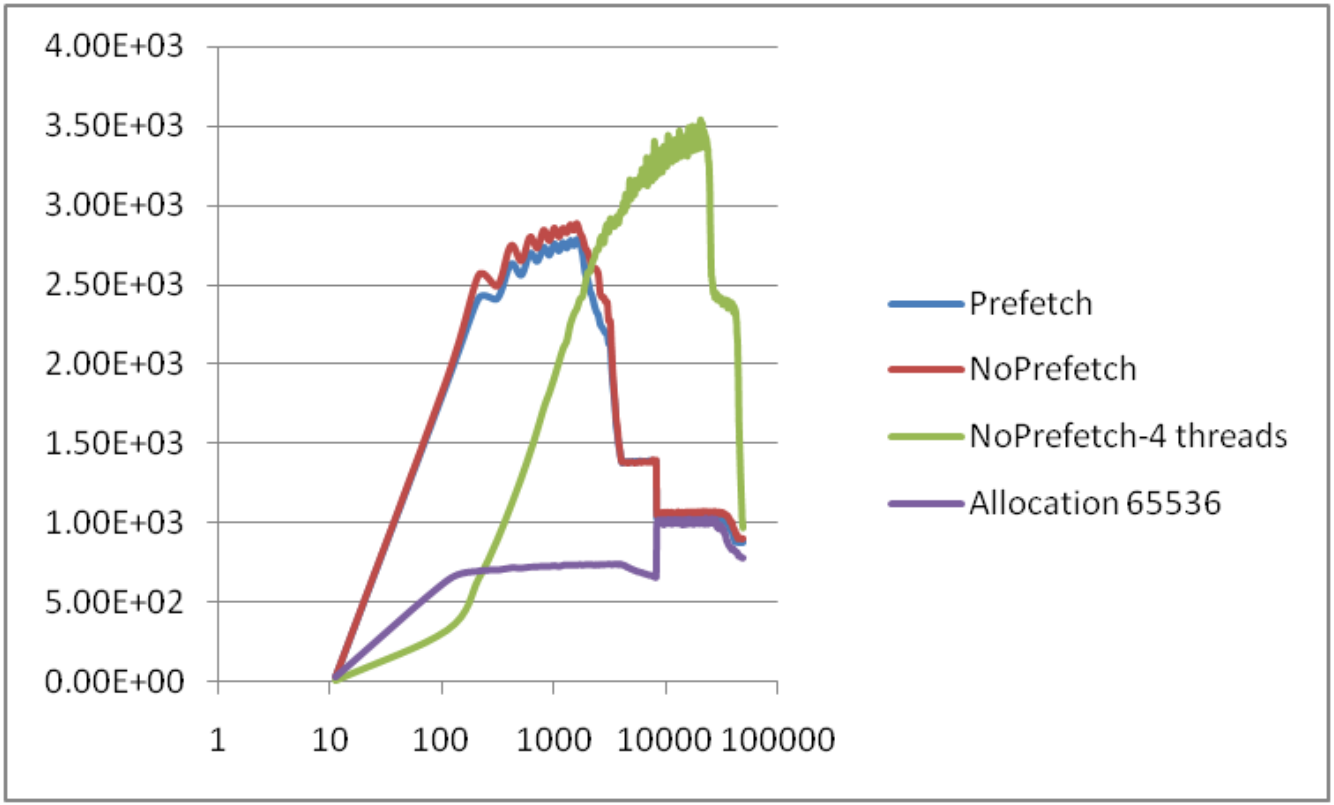
Bad Cache Alignment

Time%		0.2%	
Time		0.000003	
Calls		1	
PAPI_L1_DCA	455.433M/sec	1367	ops
DC_L2_REFILL_MOESI	49.641M/sec	149	ops
DC_SYS_REFILL_MOESI	0.666M/sec	2	ops
BU_L2_REQ_DC	74.628M/sec	224	req
User time	0.000 secs	7804	cycles
Utilization rate		97.9%	
L1 Data cache misses	50.308M/sec	151	misses
LD & ST per D1 miss		9.05	ops/miss
D1 cache hit ratio		89.0%	
LD & ST per D2 miss		683.50	ops/miss
D2 cache hit ratio		99.1%	
L2 cache hit ratio		98.7%	
Memory to D1 refill	0.666M/sec	2	lines
Memory to D1 bandwidth	40.669MB/sec	128	bytes
L2 to Dcache bandwidth	3029.859MB/sec	9536	bytes

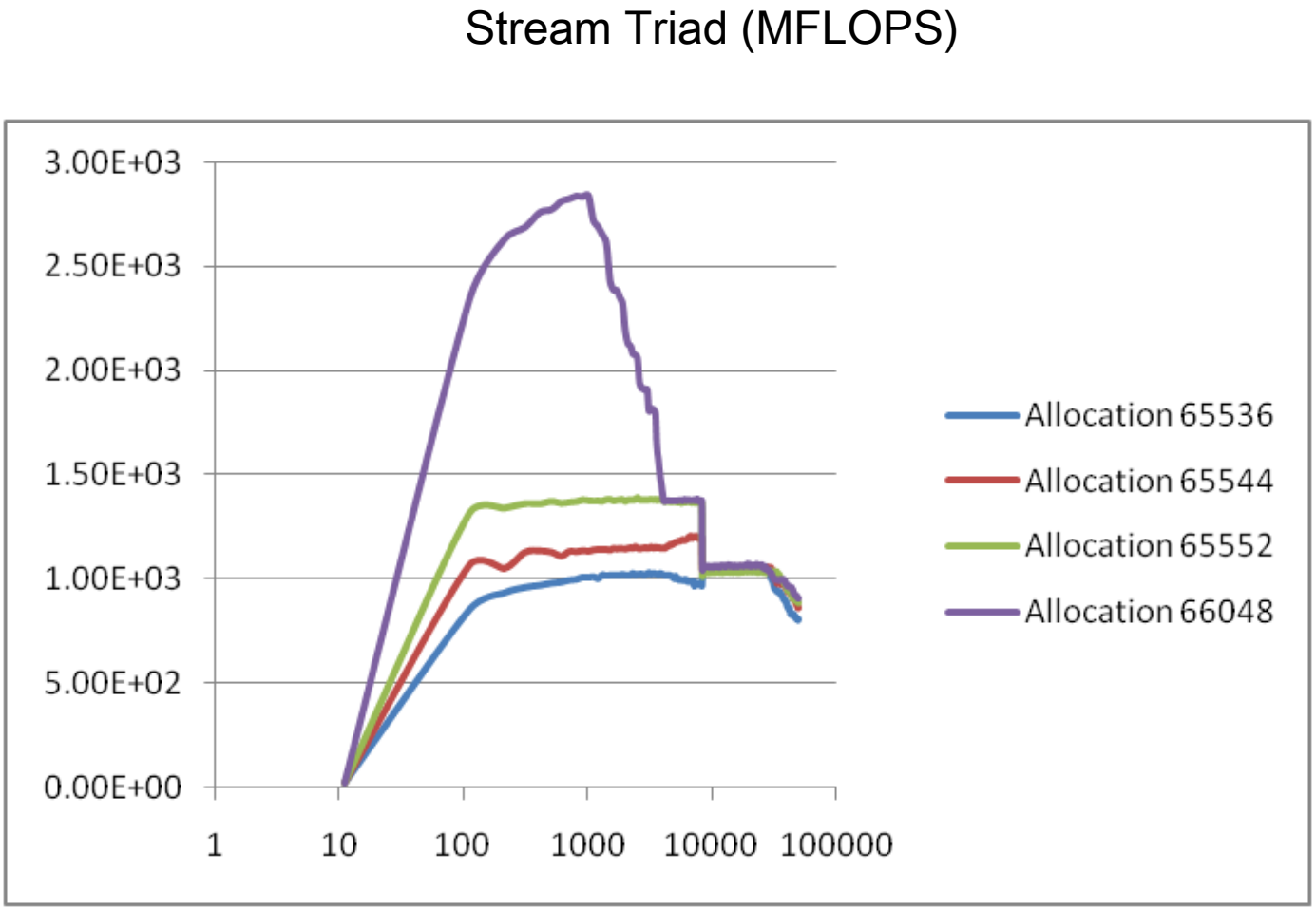
Good Cache Alignment

Time%		0.1%	
Time		0.000002	
Calls		1	
PAPI_L1_DCA	689.986M/sec	1333	ops
DC_L2_REFILL_MOESI	33.645M/sec	65	ops
DC_SYS_REFILL_MOESI		0	ops
BU_L2_REQ_DC	34.163M/sec	66	req
User time	0.000 secs	5023	cycles
Utilization rate		95.1%	
L1 Data cache misses	33.645M/sec	65	misses
LD & ST per D1 miss		20.51	ops/miss
D1 cache hit ratio		95.1%	
LD & ST per D2 miss		1333.00	ops/miss
D2 cache hit ratio		100.0%	
L2 cache hit ratio		100.0%	
Memory to D1 refill		0	lines
Memory to D1 bandwidth		0	bytes
L2 to Dcache bandwidth	2053.542MB/sec	4160	bytes

Performance = F(Cache Utilization)



Performance = F(Cache Utilization)



Compilers

PGI

- Recommended first compile/run
  - -fastsse –tp barcelona-64
- Get diagnostics
  - -Minfo –Mneginfo
- Inlining
  - **–Mipa=fast,inline**
- Recognize OpenMP directives
  - -mp=nonuma
- **Automatic parallelization**
  - **-Mconcur**

Pathscale

- Recommended first compile/run
  - Ftn –O3 –OPT:Ofast -march=barcelona
- Get Diagnostics
  - -LNO:simd_verbose=ON
- Inlining
  - -ipa
- Recognize OpenMP directives
  - -mp
- **Automatic parallelization**
  - **-apo**

PGI Basic Compiler Usage

- A compiler driver interprets options and invokes pre-processors, compilers, assembler, linker, etc.
- Options precedence: if options conflict, last option on command line takes precedence
- Use -Minfo to see a listing of optimizations and transformations performed by the compiler
- Use -help to list all options or see details on how to use a given option, e.g. pgf90 -Mvect -help
- Use man pages for more details on options, e.g. “man pgf90”
- Use –v to see under the hood

Flags to support language dialects

- Fortran
  - pgf77, pgf90, pgf95, pghpf tools
  - Suffixes .f, .F, .for, .fpp, .f90, .F90, .f95, .F95, .hpf, .HPF
  - -Mextend, -Mfixed, -Mfreeform
  - Type size –i2, -i4, -i8, -r4, -r8, etc.
  - -Mcray, -Mbyteswapio, -Mupcase, -Mnomain, -Mrecursive, etc.
- C/C++
  - pgcc, pgCC, aka pgcpp
  - Suffixes .c, .C, .cc, .cpp, .i
  - -B, -c89, -c9x, -Xa, -Xc, -Xs, -Xt
  - -Msignextend, -Mfcon, -Msingle, -Muchar, -Mgccbugs

Specifying the target architecture

- Use the “tp” switch. Don’t need for Dual Core
  - -tp k8-64 or –tp p7-64 or –tp core2-64 for 64-bit code.
  - -tp amd64e for AMD opteron rev E or later
  - -tp x64 for unified binary
  - -tp k8-32, k7, p7, piv, piii, p6, p5, px for 32 bit code
  - -tp barcelona-64

Flags for debugging aids

- -g generates symbolic debug information used by a debugger
- -gopt generates debug information in the presence of optimization
- -Mbounds adds array bounds checking
- -v gives verbose output, useful for debugging system or build problems
- -Mlist will generate a listing
- -Minfo provides feedback on optimizations made by the compiler
- -S or –Mkeepasm to see the exact assembly generated

Basic optimization switches

- Traditional optimization controlled through -O[<n>], n is 0 to 4.
- -fast switch combines common set into one simple switch, is equal to -O2 -Munroll=c:1 -Mnoframe -Mlre
  - For -Munroll, c specifies completely unroll loops with this loop count or less
  - -Munroll=n:<m> says unroll other loops m times
- -Mlre is loop-carried redundancy elimination

Basic optimization switches, cont.

- fastsse switch is commonly used, extends –fast to SSE hardware, and vectorization
- -fastsse is equal to -O2 -Munroll=c:1 -Mnoframe -Mlre (-fast) plus -Mvect=sse, -Mscalarsse -Mcache_align, -Mflushz
- -Mcache_align aligns top level arrays and objects on cache-line boundaries
- -Mflushz flushes SSE denormal numbers to zero

Node level tuning

- **Vectorization** – packed SSE instructions maximize performance
- **Interprocedural Analysis (IPA)** – use it! motivating examples
- **Function Inlining –** especially important for C and C++
- **Parallelization** – for Cray multi-core processors
- **Miscellaneous Optimizations** – hit or miss, but worth a try

Vectorizable F90 Array Syntax Data is REAL*4

```
350 !
351 !   Initialize vertex, similarity and coordinate arrays
352 !
353     Do Index = 1, NodeCount
354       IX = MOD (Index - 1, NodesX) + 1
355       IY = ((Index - 1) / NodesX) + 1
356       CoordX (IX, IY) = Position (1) + (IX - 1) * StepX
357       CoordY (IX, IY) = Position (2) + (IY - 1) * StepY
358       JetSim (Index)  = SUM (Graph (:, :, Index) * &
359     &                       GaborTrafo (:, :, CoordX(IX,IY), CoordY(IX,IY)))
360       VertexX (Index) = MOD (Params%Graph%RandomIndex (Index) - 1, NodesX) + 1
361       VertexY (Index) = ((Params%Graph%RandomIndex (Index) - 1) / NodesX) + 1
362     End Do
```

Inner “loop” at line 358 is vectorizable, can used packed SSE instructions

–fastsse to Enable SSE Vectorization
–Minfo to List Optimizations to stderr

```
% pgf95 -fastsse -Mipa=fast -Minfo -S graphRoutines.f90
…
localmove:
   334, Loop unrolled 1 times (completely unrolled)
   343, Loop unrolled 2 times (completely unrolled)
   358, Generated an alternate loop for the inner loop
            Generated vector sse code for inner loop
            Generated 2 prefetch instructions for this loop
            Generated vector sse code for inner loop
            Generated 2 prefetch instructions for this loop
   …
```

Scalar SSE:

```
.LB6_668:
# lineno: 358
        movss   -12(%rax),%xmm2
        movss   -4(%rax),%xmm3
        subl    $1,%edx
        mulss   -12(%rcx),%xmm2
        addss   %xmm0,%xmm2
        mulss   -4(%rcx),%xmm3
        movss   -8(%rax),%xmm0
        mulss   -8(%rcx),%xmm0
        addss   %xmm0,%xmm2
        movss   (%rax),%xmm0
        addq    $16,%rax
        addss   %xmm3,%xmm2
        mulss   (%rcx),%xmm0
        addq    $16,%rcx
        testl   %edx,%edx
        addss   %xmm0,%xmm2
        movaps  %xmm2,%xmm0
        jg      .LB6_625
```

Vector SSE:

```
.LB6_1245:
# lineno: 358
        movlps  (%rdx,%rcx),%xmm2
        subl    $8,%eax
        movlps  16(%rcx,%rdx),%xmm3
        prefetcht0  64(%rcx,%rsi)
        prefetcht0  64(%rcx,%rdx)
        movhps  8(%rcx,%rdx),%xmm2
        mulps   (%rsi,%rcx),%xmm2
        movhps  24(%rcx,%rdx),%xmm3
        addps   %xmm2,%xmm0
        mulps   16(%rcx,%rsi),%xmm3
        addq    $32,%rcx
        testl   %eax,%eax
        addps   %xmm3,%xmm0
        jg      .LB6_1245:
```

Facerec Scalar: 104.2 sec
Facerec Vector: 84.3 sec

Vectorizable C Code Fragment?

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222         u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224         float vdt = v[i] * dt;
225         u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Minfo functions.c
func4:
    221, Loop unrolled 4 times
    221, Loop not vectorized due to data dependency
    223, Loop not vectorized due to data dependency
```

Pointer Arguments Inhibit Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222         u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224         float vdt = v[i] * dt;
225         u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Unrolled inner loop 4 times
```

C Constant Inhibits Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222         u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224         float vdt = v[i] * dt;
225         u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Mfcon –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Generated vector SSE code for inner loop
         Generated 4 prefetch instructions for this loop
```

-Msafeptr Option and Pragma

–M[no]safeptr[=all | arg | auto | dummy | local | static | global]

all	All pointers are safe
arg	Argument pointers are safe
local	local pointers are safe
static	static local pointers are safe
global	global pointers are safe

#pragma [*scope*] [no]safeptr={arg | local | global | static | all},…

Where *scope* is *global*, *routine* or *loop*

Common Barriers to SSE Vectorization

- **Potential Dependencies & C Pointers –** Give compiler more info with –Msafeptr, pragmas, or restrict type qualifer
- **Function Calls –** Try inlining with –Minline or –Mipa=inline
- **Type conversions –** manually convert constants or use flags
- **Large Number of Statements –** Try –Mvect=nosizelimit
- **Too few iterations –** Usually better to unroll the loop
- **Real dependencies –** Must restructure loop, if possible

Barriers to Efficient Execution of Vector SSE Loops

- Not enough work – vectors are too short
- Vectors not aligned to a cache line boundary
- Non unity strides
- Code bloat if altcode is generated

What can Interprocedural Analysis and Optimization with –Mipa do for You?

- Interprocedural constant propagation
- Pointer disambiguation
- Alignment detection, Alignment propagation
- Global variable mod/ref detection
- F90 shape propagation
- Function inlining
- IPA optimization of libraries, including inlining

Effect of IPA on the WUPWISE Benchmark

PGF95 Compiler Options	Execution Time in Seconds
–fastsse	156.49
–fastsse –Mipa=fast	121.65
–fastsse –Mipa=fast,inline	91.72

- –Mipa=fast => constant propagation => compiler sees complex matrices are all 4x3 => completely unrolls loops
- –Mipa=fast,inline => small matrix multiplies are all inlined

Using Interprocedural Analysis

- Must be used at both compile time and link time
- Non-disruptive to development process – edit/build/run
- Speed-ups of 5% - 10% are common
- –Mipa=safe:<*name*> - safe to optimize functions which call or are called from unknown function/library *name*
- –Mipa=libopt – perform IPA optimizations on libraries
- –Mipa=libinline – perform IPA inlining from libraries

Explicit Function Inlining

–Minline[=[lib:]<inlib> | [name:]<func> | except:<func> | size:<n> | levels:<n>]

[lib:]<inlib>	Inline extracted functions from *inlib*
[name:]<func>	Inline function func
except:<func>	Do not inline function func
size:<n>	Inline only functions smaller than n statements (approximate)
levels:<n>	Inline n levels of functions

For C++ Codes, PGI Recommends IPA-based inlining or –Minline=levels:10!

Other C++ recommendations

- **Encapsulation, Data Hiding - ** small functions, inline!
- **Exception Handling –** use –no_exceptions until 7.0
- **Overloaded operators, overloaded functions -** okay
- **Pointer Chasing -** -Msafeptr, restrict qualifer, 32 bits?
- **Templates, Generic Programming –** now okay
- **Inheritance, polymorphism, virtual functions –** runtime lookup or check, no inlining, potential performance penalties

SMP Parallelization

- –Mconcur for auto-parallelization on multi-core
  - Compiler strives for parallel outer loops, vector SSE inner loops
  - –Mconcur=innermost forces a vector/parallel innermost loop
  - –Mconcur=cncall enables parallelization of loops with calls
- –mp to enable OpenMP 2.5 parallel programming model
  - See PGI User's Guide or OpenMP 2.5 standard
  - OpenMP programs compiled w/out –mp=nonuma
- –Mconcur and –mp can be used together!

EKOPath Basic Optimizations

PathScale™

`-g`	Generate debug (DWARF) information. Changes optimization level to -O0 unless explicitly overridden.
`-O0`	No optimization
`-O1`	Local optimization (straight line code)
`-O2`	Global scalar optimizations (-O2 is default)
`-O3`	Loop level transformations and vectorizations
`-ipa`	Inter-procedural optimizations (whole program). Can be used at any optimization level.
`-OPT:Ofast`	Generally safe but may impact floating point correctness. Maximizes performance. Equivalent to: `-OPT:ro=2:Olimit=0:div_split=ON:alias=typed`
`-Ofast`	Equivalent to `-O3 -ipa -OPT:Ofast -fno-math-errno`

Option Groups

PathScale™

- Options organized into groups by compiler phase or by class of feature
- General syntax:

```
-GROUPNAME:opt[=val]{:opt=[val]}
```

- Some GNU-style flags map to these options

```
-march -ffast-math -ffloat-store -fno-inline
```

- Group names:

`-LIST:`	User listing
`-OPT:`	Optimizations
`-TARG:`	Target machine
`-TENV:`	Target environment
`-INLINE:`	Back-end inlining
`-IPA:`	Inter-procedural analysis
`-LANG:`	Language features
`-CG:`	Code generation
`-WOPT:`	Global scalar optimization
`-LNO:`	Loop nest optimization

Alias options

PathScale™

Improving performance of generated code by allowing the compiler to make assumptions about aliasing

Mainly for C/C++ programs

-OPT:alias=typed

- Activate ANSI/ISO C standard
- Object not aliased if they have different base types
- Implied by –Ofast

-OPT:alias=restrict

- Regard all pointers as having the 'restrict' attribute

-OPT:alias=disjoint

- No two pointers ever point to the same object
- Many programs will not run correctly with this option

Controlling Floating-point Code

PathScale™

- Lower precision requirements to allow for faster code

- `OPT:roundoff=`
  - Specifies extent of roundoff error the compiler is allowed to introduce
    - `0` = no roundoff error (default at `-O0`, `-O1`, `-O2`)
    - `1` = limited roundoff error (default at `-O3`)
    - `2` = allow roundoff error due to re-associating expressions (default at `-Ofast`)
    - `3` = any roundoff error is allowed
- `OPT:IEEE_arithmetic=`
  - Specifies level of conformance to IEEE 754 floating-point roundoff and overflow behavior
    - 1 = string conformance to IEEE accuracy (default at `-O0`, `-O1`, `-O2`)
    - 2 = allow inexact results not conforming to IEEE 754 (default at `-O3`)
    - 3 = allow any mathematically valid transformations

`-OPT:IEEE_NaN_Inf=(on|off)`

- Controls conformance to IEEE for Not-a-Number and Infinity operands
- Default is `on`

Parallelization

PathScale™

- OpenMP
  - Option to enable directives:

    `-mp`
  - OpenMP 2.5 in Fortran, C, & C++
    - The C++ OpenMP support is limited and does not support OpenMP directives in C++ source that use exceptions, classes or templates. (We have found some codes with very simplistic use of classes may work.)

- Autoparallelization
  - Option to enable: `-apo`

Performance Tuning

PathScale™

`-Ofast` is the most aggressive optimization option

Equivalent to `-O3 -ipa -OPT:Ofast -fno-math-errno`

`-OPT:Ofast` is equivalent to

`-OPT:ro=2:Olimit=0:div_split=ON:alias=typed`

– A large number of other options are related to performance tuning

- Phase specific options:

`-CG`
`-INLINE`
`-IPA`
`-WOPT`
`-LNO`

– PathOpt2 allows automatic search of best flag combinations

Tuning for AMD "Barcelona"

PathScale™

- Tuned Prefetch for smaller L2 cache
  - option LNO: stream_prefetch=1
  - option CG:use_prefetch_nta (non temporal)
- SIMD unaligned loads
  - improves vectorization
- FP instruction tuning
  - aligned packed double (movapd)
    - replaces scalar double (movsd)
    - removes register dependency
  - movsd replaces movlpd (for loads)

Above changes account for >10% performance improvement

Options to Help Expose User Errors

PathScale™

Programs may run incorrectly only at higher optimization levels

- Causes include compiler bugs or bad coding practices

To help diagnose bad coding practices

`-OPT:alias=no_parm`

- Fortran compiler does NOT assume Fortran no-alias rule for parameters

`-LANG:rw_const=on`

- For cases where callee modifies constant argument

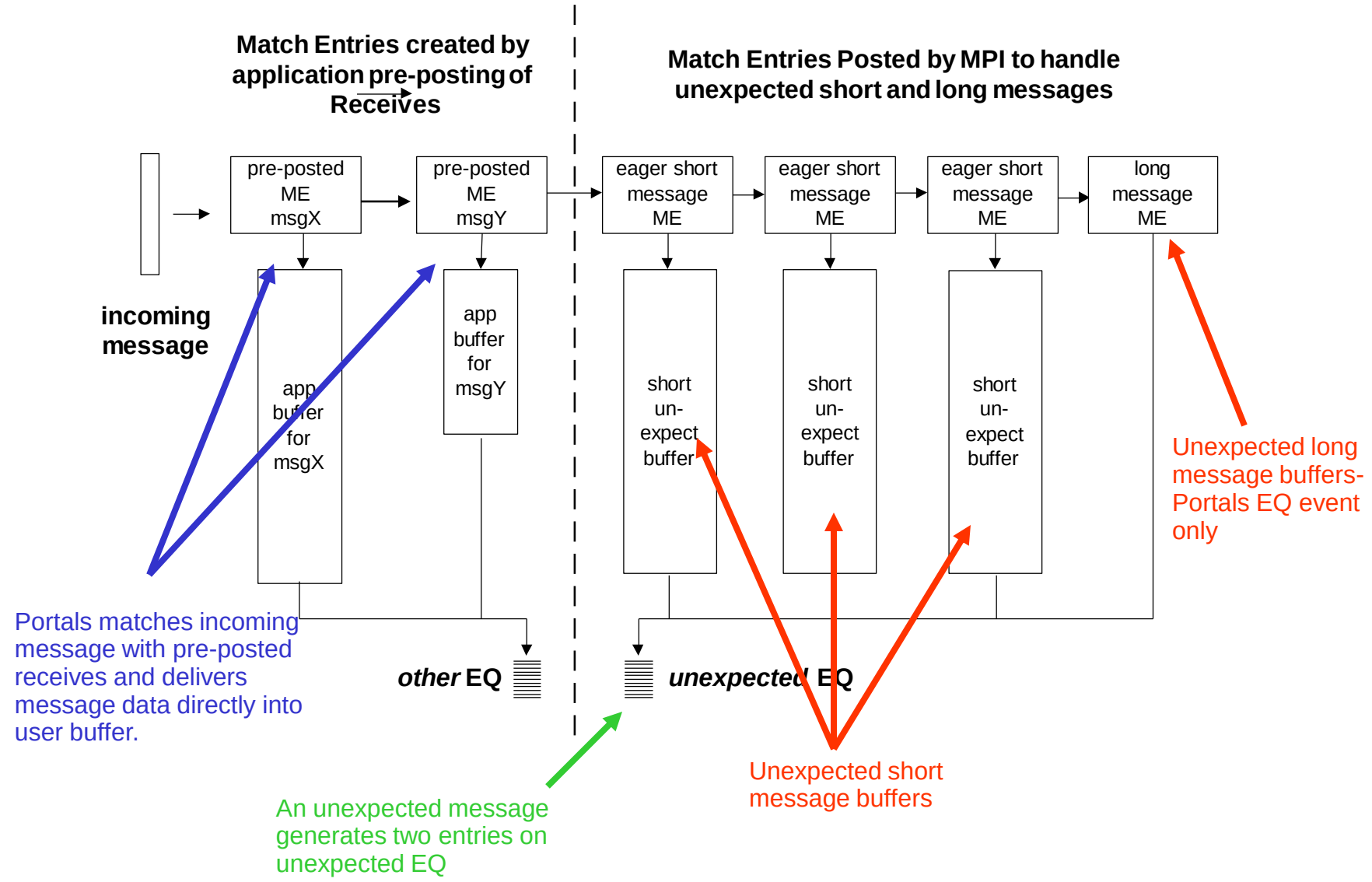
`-trapuv`

- Initializes variables with NaN. If program uses the uninitialized variable, it will crash instead of generating incorrect results

`-zerouv`

- Initializes variables to 0
- Good for programs that incorrectly assume memory is always initialized to zero.

XT MPI – Receive Side



ninept_4 original

```
do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
       XOUT(i,j) = (9 pt. weighted sum)
    end do
end do

! fill buffers and send east-west
! boundary info
do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      buffer_east_snd(i)= ...
      buffer_west_snd(i)= ...
   end do
end do

call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...

! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_RECV(buffer_west_rcv, ...
call MPI_RECV(buffer_east_rcv, ...

call MPI_WAITALL(2, …

do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      XOUT(n,j)          = ...
      XOUT(iphys_e+n,j) = ...
   end do
end do

! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...

! receive north-south boundary info
call MPI_RECV(XOUT(...
call MPI_RECV(XOUT(...
call MPI_WAITALL(2, …
```

Oak Ridge National Laboratory

Office of Science U.S. Department of Energy

ninept_4 modified

```
! Prepost receive requests
call MPI_IRECV(buffer_west_rcv, ...
call MPI_IRECV(buffer_east_rcv, ...
call MPI_IRECV(XOUT(...
call MPI_IRECV(XOUT(...

do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
       XOUT(i,j) = (9 pt. weighted sum)
    end do
end do

! fill buffers and send east-west
! boundary info
do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      buffer_east_snd(i)= ...
      buffer_west_snd(i)= ...
   end do
end do
```

```
call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...

! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_WAITALL(2, …

do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
      XOUT(n,j)         = ...
      XOUT(iphys_e+n,j) = ...
   end do
end do

! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...

! receive north-south bddy info
call MPI_WAITALL(6, …
```

Optimization

Getting ready for Quad Core

- Bytes/flops will decrease
  - XT3 – 5 GB/sec/2.6 GHZ* 2Flops/clock
    - 1 Byte/flop
  - XT4 (dual) – 6.25GB/sec/2.6 GHZ* 2Flops/clock/2 processors
    - ½ Byte/flop
  - XT4 (quad) – 8 GB/sec/2.2GHZ*4Flops/clock/4 processors
    - ¼ Byte/flop
- Interconnect Bytes/flop will decrease
  - XT3 – 2 GB/sec/2.6 GHZ* 2Flops/clock
    - 1/3 Bytes/flop
  - XT4 (dual) – 6 GB/sec/2.6 GHZ* 2Flops/clock/2 processors
    - 1/2 Bytes/flop
  - XT4 (quad) – 6 GB/sec/2.2GHZ*4Flops/clock/4 processors
    - 1/7 Byte/flop

What can be done?

- MPI is optimized for intra-node communication; however, messages off the node will contend for bandwidth requirements off the node
  - Number of messages going through the NIC could become a problem
- OpenMP across the cores on the node will help
  - Shared Cache is designed to help OpenMP reduce the applications memory requirements
  - Reduces the message traffic off the node

What about those SSE instructions

- The Quad core is capable of generating 4 flops/clock in 64 bit mode and 8 flops/clock for 32 bit mode
  - Assembler must contain SSE instructions
  - Compilers only generate SSE instructions when they vectorize the DO loops
- Operands should be aligned on 128 bit boundaries
  - Operand alignment can be performed; however, it degrades the performance.
- Watch out for Libraries – are they Quad core enabled?

Caution when timing Kernels

- The worse case timings will be shown in the following examples. None of the operands will be cache resident. This is assured by calling a routine called FLUSH prior to each example.

Flush Routine

```
SUBROUTINE FLUSH
 common/fl/ A(896896),x
 real*8 A,x
 do i=1,896896
 x=x+a(i)
 enddo
 end
```

Notice, we are replacing everything that is in cache with read Data. If we stored into A, the contents of cache would have to Be written to memory before using the cache for other data.

When calling FLUSH

```
      REAL*8 A,X
      common/fl/ A(896896),x
C
        X=0
        A=ranf()
        CALL LP41000
        print *,x
```

These compilers can recognize that x in the COMMON block is not used anywhere, so we print it. Also we initialize A

Compiler Options for Quad Core

- Pathscale

  Ftn –O3 –OPT:Ofast -march=barcelona -LNO:simd_verbose=ON

- PGI

  Ftn –fastsse –r8 –Minfo –Mneginfo –tp barcelona-64

Indirect Addressing

```
(  300) C     FIVE OPERATIONS - TWO OPERANDS     RATIO = 5/2
(  301)
(  302)       DO 41012 I = 1, N
(  303)        Y(IY(I)) =  c0 + X(IX(I)) * (C1 + X(IX(I))
(  304)      *           * (C2 + X(IX(I))                 ))
(  305) 41012 CONTINUE
```

302, Loop unrolled 2 times

Contiguous Addressing

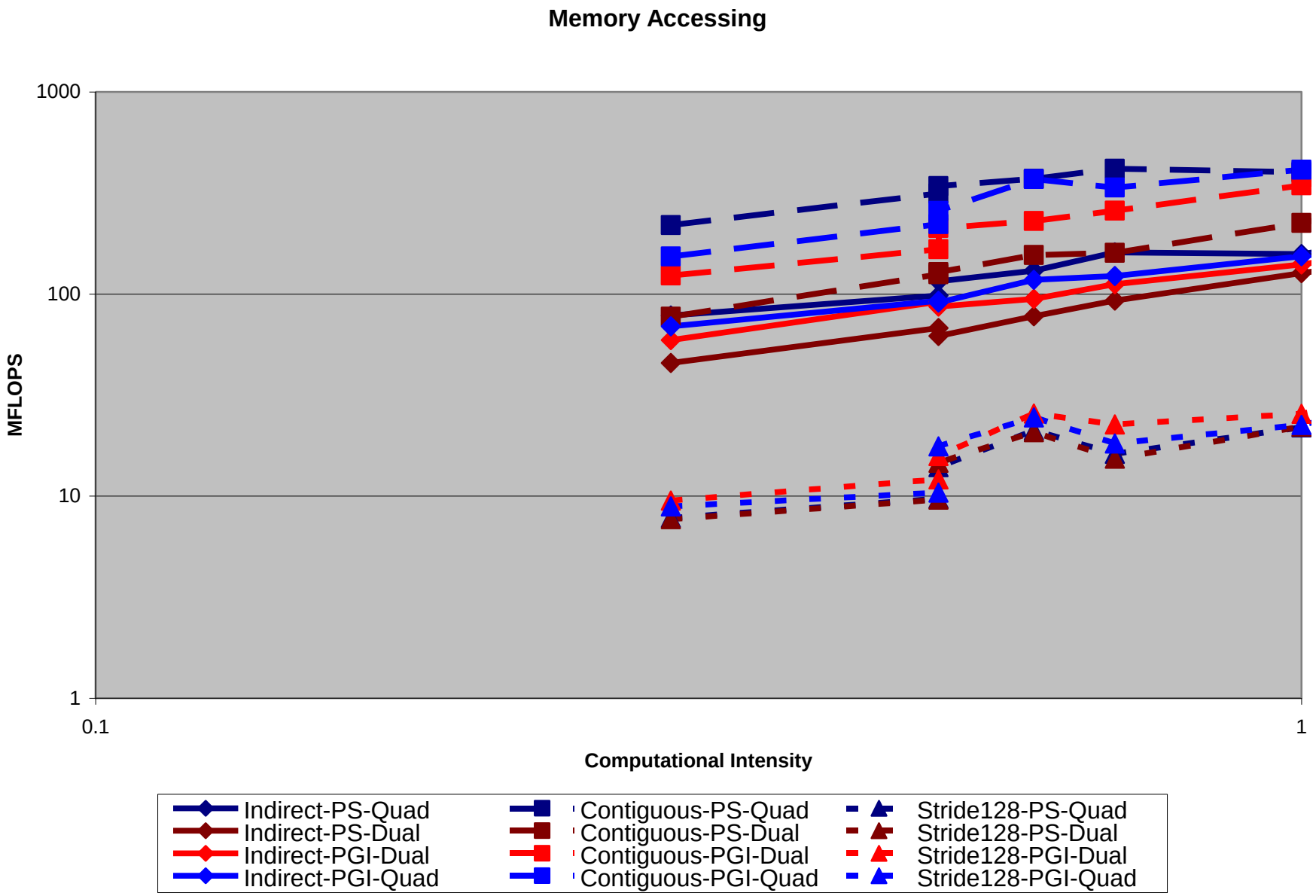
```
(  799)        DO 41033 I = 1, N
(  800)         Y(I) = c0 + X(I) * (C1 + X(I) * (C2 + X(I)
(  801)      *                   * (C3 + X(I)              )))
(  802) 41033 CONTINUE
```

799, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop

Bad Stride Addressing

```
( 1239)        II=1
( 1240)
( 1241)        DO 41072 I = 1, N
( 1242)         Y(II) = c0 + X(II) * (C1 + X(II) * (C2 + X(II) ))
( 1243)         II = II + ISTRIDE
( 1244) 41072 CONTINUE
```

1241, Loop unrolled 1 times


Memory Accessing
1000
100
10
1
MFLOPS
0.1
1
Computational Intensity
Indirect-PS-Quad
Indirect-PS-Dual
Indirect-PGI-Dual
Indirect-PGI-Quad
Contiguous-PS-Quad
Contiguous-PS-Dual
Contiguous-PGI-Dual
Contiguous-PGI-Quad
Stride128-PS-Quad
Stride128-PS-Dual
Stride128-PGI-Dual
Stride128-PGI-Quad

Bad Striding

```
(   47) C     DIMENSION A(128,N)
(   48)
(   49)       DO 41080  I = 1,N
(   50)        A( 1,I) = C1*A(13,I) + C2* A(12,I) + C3*A(11,I) +
(   51)      *           C4*A(10,I) + C5* A( 9,I) + C6*A( 8,I) +
(   52)      *           C7*A( 7,I) + C0*(A( 5,I) + A( 6,I) )  + A( 3,I)
(   53) 41080 CONTINUE
```

PGI
49, Generated vector sse code for inner loop
Pathscale
(lp41080.f:49) Non-contiguous array "A(_BLNK__.0.0)" reference exists.
Loop was not vectorized.

Rewrite

```
(    74) C     DIMENSION B(129,N)
(    75)
(    76)       DO 41081 I = 1,N
(    77)        B( 1,I) = C1*B(13,I) + C2* B(12,I) + C3*B(11,I) +
(    78)      *           C4*B(10,I) + C5* B( 9,I) + C6*B( 8,I) +
(    79)      *           C7*B( 7,I) + C0*(B( 5,I) + B( 6,I) )  + B( 3,I)
(    80) 41081 CONTINUE
```

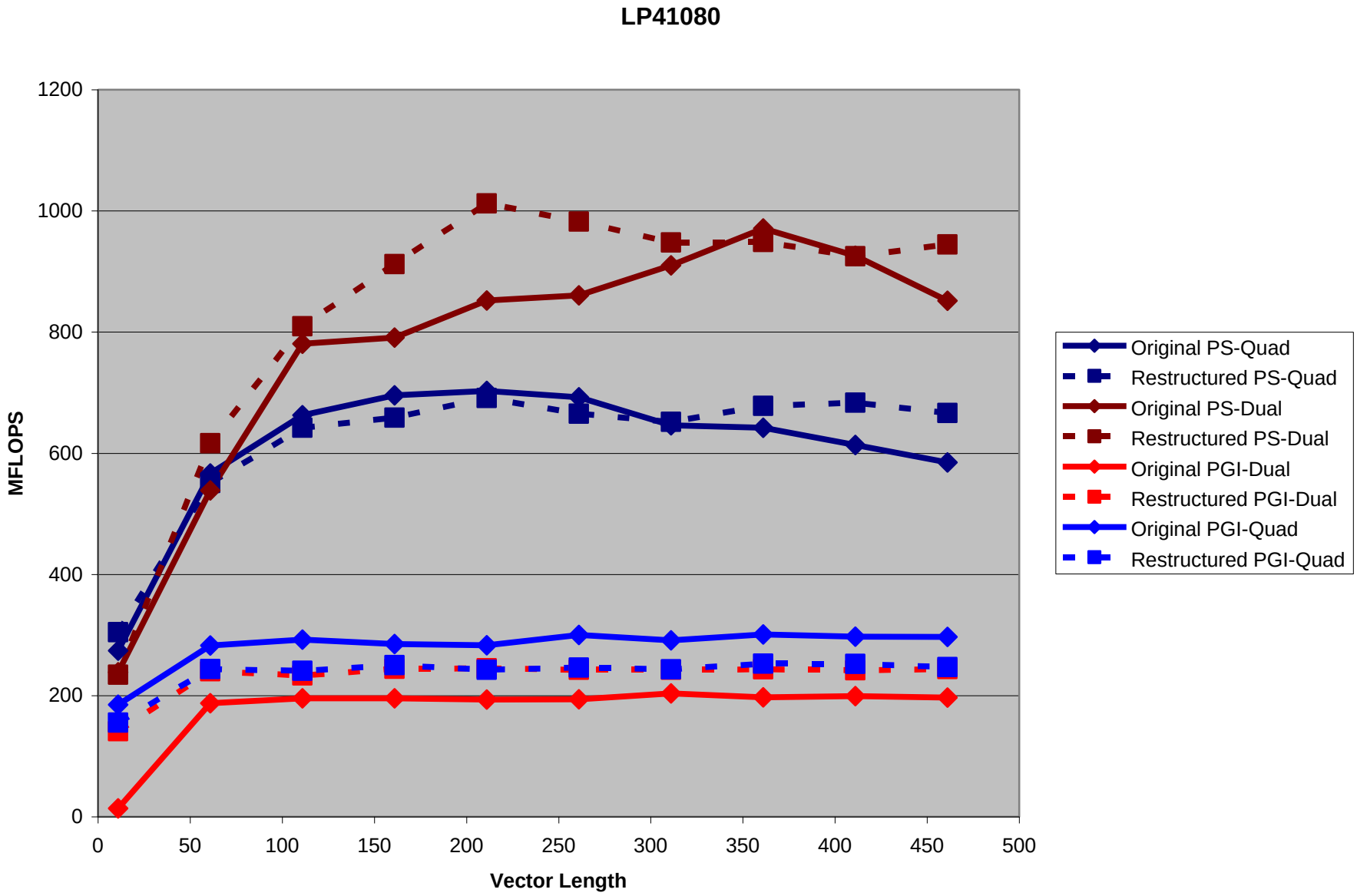
PGI

76, Generated vector sse code for inner loop

Pathscale

(lp41080.f:76) Non-contiguous array "B(_BLNK__.512000.0)" reference exists.

Loop was not vectorized.


LP41080
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Bad Striding

```
(    5)        COMMON A(8,8,IIDIM,8),B(8,8,iidim,8)


(   59)        DO 41090 K = KA, KE, -1
(   60)          DO 41090 J = JA, JE
(   61)            DO 41090 I = IA, IE
(   62)              A(K,L,I,J) = A(K,L,I,J) - B(J,1,i,k)*A(K+1,L,I,1)
(   63)       *    - B(J,2,i,k)*A(K+1,L,I,2) - B(J,3,i,k)*A(K+1,L,I,3)
(   64)       *    - B(J,4,i,k)*A(K+1,L,I,4) - B(J,5,i,k)*A(K+1,L,I,5)
(   65) 41090 CONTINUE
(   66)
```

PGI

59, Loop not vectorized: loop count too small
60, Interchange produces reordered loop nest: 61, 60
Loop unrolled 5 times (completely unrolled)
61, Generated vector sse code for inner loop

Pathscale

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

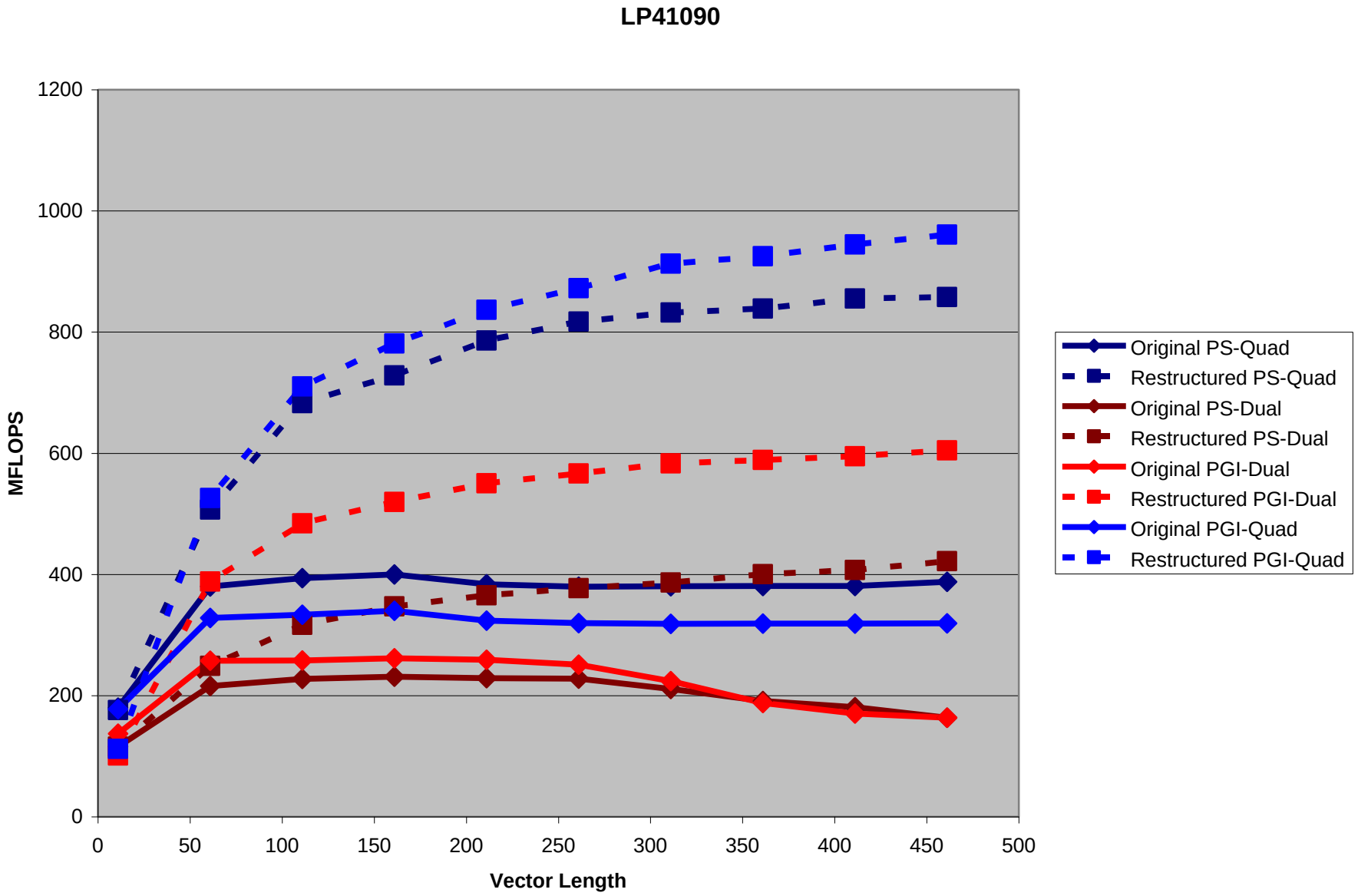
```
(     6)        COMMON AA(IIDIM,8,8,8),BB(IIDIM,8,8,8)


(    95)        DO 41091 K = KA, KE, -1
(    96)          DO 41091 J = JA, JE
(    97)            DO 41091 I = IA, IE
(    98)              AA(I,K,L,J) = AA(I,K,L,J) - BB(I,J,1,K)*AA(I,K+1,L,1)
(    99)       *      - BB(I,J,2,K)*AA(I,K+1,L,2) - BB(I,J,3,K)*AA(I,K+1,L,3)
(   100)       *      - BB(I,J,4,K)*AA(I,K+1,L,4) - BB(I,J,5,K)*AA(I,K+1,L,5)
(   101) 41091 CONTINUE
```

PGI

95, Loop not vectorized: loop count too small
96, Outer loop unrolled 5 times (completely unrolled)
97, Generated 3 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop

Pathscale

(lp41090.f:99) LOOP WAS VECTORIZED.


LP41090
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Scalars

```
(   59) C       THE ORIGINAL
(   60)
(   61)       DO 42010 KK = 1, N
(   62)        T000       = A(KK,K000)
(   63)        T001       = A(KK,K001)
(   64)        T010       = A(KK,K010)
(   65)        T011       = A(KK,K011)
(   66)        T100       = A(KK,K100)
(   67)        T101       = A(KK,K101)
(   68)        T110       = A(KK,K110)
(   69)        T111       = A(KK,K111)
(   70)        B1         = B(KK,K000)
(   71)        B2         = B(KK,K001)
(   72)        B3         = B(KK,K010)
(   73)        B4         = B(KK,K011)
(   74)        R1         = T100 * C1 + T110 * C2
(   75)        S1         = T101 * C1 - T111 * C2
(   76)        RS         = T000 + R1
(   77)        SS         = T001 + S1
(   78)        RU         = T010 - R1
(   79)        SU         = T011 - S1
(   80)        B(KK,K000) = B1 + RS
(   81)        B(KK,K001) = B2 + RU
(   82)        B(KK,K010) = B3 + SS
(   83)        B(KK,K011) = B4 - SU
(   84) 42010 CONTINUE
(   85)
```

PGI
61, Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Pathscale
(lp42010.f:61) LOOP WAS VECTORIZED.

```
(  106) C       THE RESTRUCTURED
(  107)
(  108)         DO 42011 KK = 1,N
(  109)           B(KK,K000) = B(KK,K000)         + A(KK,K000)
(  110)      *               + (A(KK,K100) * C1 + A(KK,K110) * C2)
(  111)           B(KK,K001) = B(KK,K001)         + A(KK,K010)
(  112)      *               - (A(KK,K100) * C1 + A(KK,K110) * C2)
(  113)           B(KK,K010) = B(KK,K010)         + A(KK,K001)
(  114)      *               + (A(KK,K101) * C1 - A(KK,K111) * C2)
(  115)           B(KK,K011) = B(KK,K011)         - A(KK,K011)
(  116)      *               + (A(KK,K101) * C1 - A(KK,K111) * C2)
(  117)  42011 CONTINUE
(  118)
```

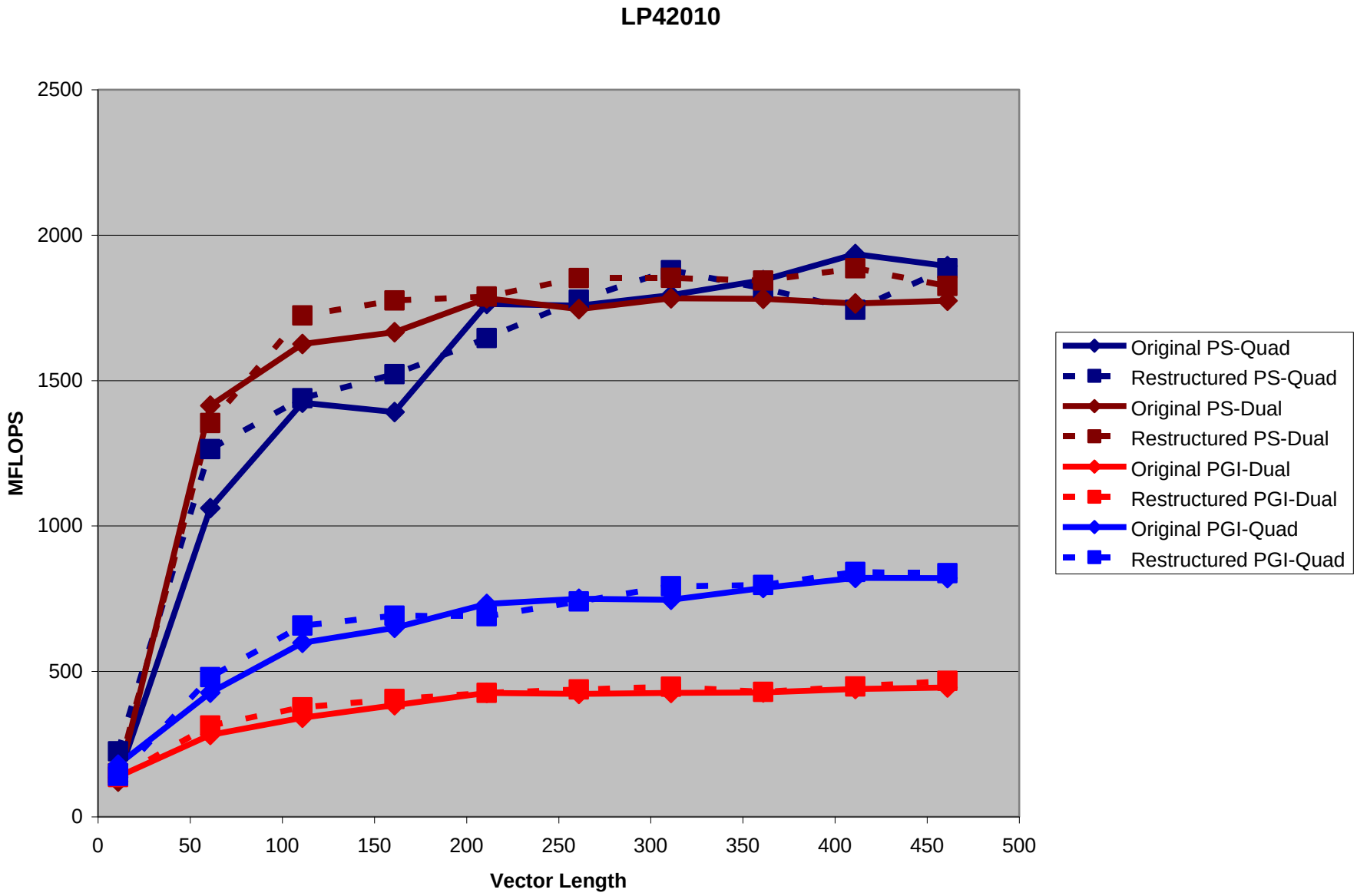
PGI

108, Generated vector sse code for inner loop

Generated 8 prefetch instructions for this loop

Pathscale

(lp42010.f:108) LOOP WAS VECTORIZED.


LP42010
MFLOPS
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
400
450
500
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

```
(   35) C     NON-RECURSIVE DO LOOP FOR TIMING COMPARISON
(   36)
(   37)       DO 43010 I = 2, N
(   38)        A(I) = A(I+1) * B(I) + C(I)
(   39) 43010 CONTINUE
(   40)
```

PGI

37, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop

Pathscale

(lp43010.f:37) LOOP WAS VECTORIZED.

```
(   52) C       RECURSIVE DO LOOP
(   53)
(   54)       DO 43011 I = 2, N
(   55)        A(I) = A(I-1) * B(I) + C(I)
(   56) 43011 CONTINUE
(   57)
```

PGI

54, Loop not vectorized: data dependency
Loop unrolled 2 times

Pathscale

(lp43010.f:54) Loop has dependencies. Loop was not vectorized.

FOLR - Unrolled

```
(   71) C      UNROLLED TO DEPTH FOUR
(   72)
(   73)       DO 43012 I = 2, N-3, 4
(   74)        A(I)   = A(I-1) * B(I)   + C(I)
(   75)        A(I+1) = A(I)   * B(I+1) + C(I+1)
(   76)        A(I+2) = A(I+1) * B(I+2) + C(I+2)
(   77)        A(I+3) = A(I+2) * B(I+3) + C(I+3)
(   78) 43012 CONTINUE
(   79)
(   80) C      CLEANUP LOOP FOR DEPTH FOUR UNROLLING
(   81)
(   82)       DO 43013 J = I,N
(   83)        A(J) = A(J-1) * B(J) + C(J)
(   84) 43013 CONTINUE
(   85)
```

PGI

73, Loop not vectorized: data dependency

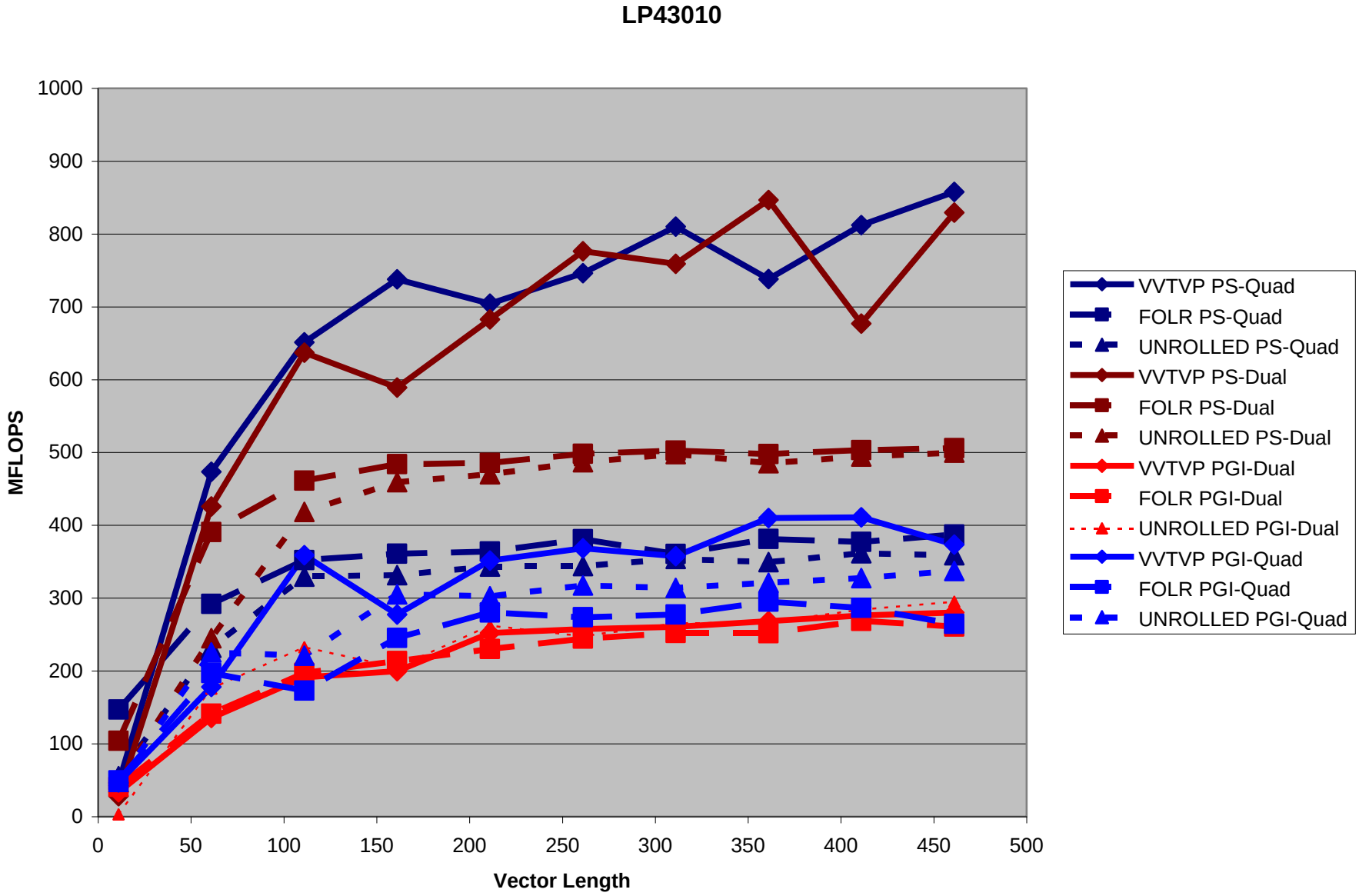
82, Loop not vectorized: data dependency

Loop unrolled 2 times

Pathscale

(lp43010.f:73) Non-contiguous array "C(_BLNK__.8000.0)" reference exists. Loop was not vectorized.

(lp43010.f:82) Loop has dependencies. Loop was not vectorized.


LP43010
MFLOPS
Vector Length
0
100
200
300
400
500
600
700
800
900
1000
0
50
100
150
200
250
300
350
400
450
500
VVTVP PS-Quad
FOLR PS-Quad
UNROLLED PS-Quad
VVTVP PS-Dual
FOLR PS-Dual
UNROLLED PS-Dual
VVTVP PGI-Dual
FOLR PGI-Dual
UNROLLED PGI-Dual
VVTVP PGI-Quad
FOLR PGI-Quad
UNROLLED PGI-Quad

Potential Recursion

```
(    42) C  GAUSS ELIMINATION
(    43)
(    44)        DO 43020 I = 1, MATDIM
(    45)         A(I,I) = 1. / A(I,I)
(    46)         DO 43020 J = I+1, MATDIM
(    47)          A(J,I) = A(J,I) * A(I,I)
(    48)          DO 43020 K = I+1, MATDIM
(    49)           A(J,K) = A(J,K) - A(J,I) * A(I,K)
(    50) 43020 CONTINUE
(    51)
```

Pathscale

(lp43020.f:46) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:48) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:48) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:48) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

PGI

```
46, Distributed loop; 2 new loops
    Interchange produces reordered loop nest: 48, 46
    Generated 2 alternate loops for the inner loop
    Unrolled inner loop 4 times
    Generated 1 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Generated 2 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 1 stores
    Generated 1 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 1 stores
    Generated 1 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 1 stores
    Generated 2 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 1 stores
    Generated 2 prefetch instructions for this loop
```

Rewrite

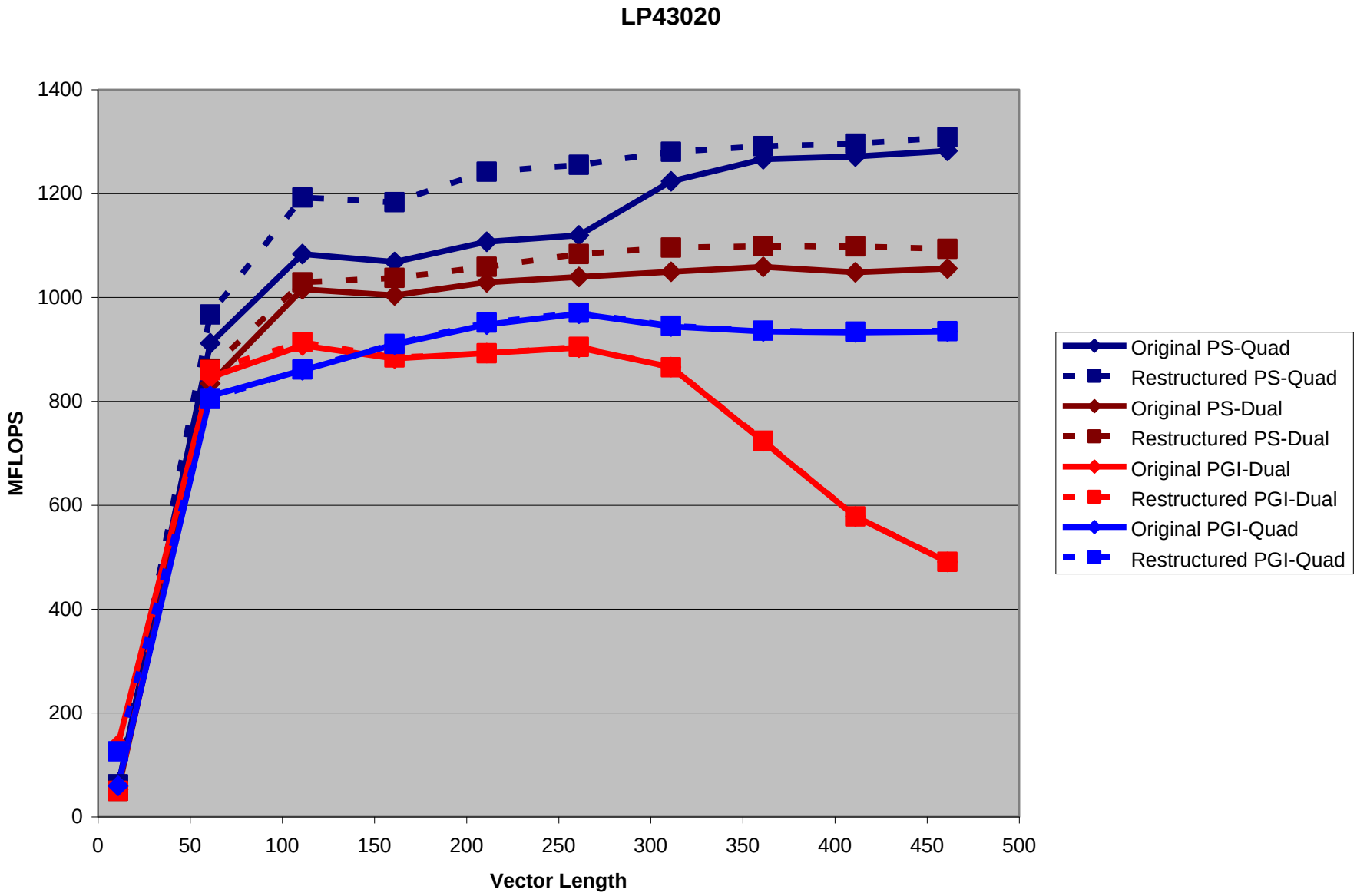
```
(    80) C  GAUSS ELIMINATION
(    81)
(    82)       DO 43021 I = 1, MATDIM
(    83)        A(I,I) = 1. / A(I,I)
(    84)        DO 43021 J = I+1, MATDIM
(    85)         A(J,I) = A(J,I) * A(I,I)
(    86) CVD$ NODEPCHK
(    87) CDIR$ IVDEP
(    88) *VDIR NODEP
(    89)          DO 43021 K = I+1, MATDIM
(    90)           A(J,K) = A(J,K) - A(J,I) * A(I,K)
(    91) 43021 CONTINUE
```

Pathscale

(lp43020.f:84) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:89) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:89) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.
(lp43020.f:89) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

PGI

```
 84, Distributed loop; 2 new loops
      Interchange produces reordered loop nest: 89, 84
      Generated 2 alternate loops for the inner loop
      Unrolled inner loop 4 times
      Generated 1 prefetch instructions for this loop
      Unrolled inner loop 4 times
      Generated 2 prefetch instructions for this loop
      Unrolled inner loop 4 times
      Used combined stores for 1 stores
      Generated 1 prefetch instructions for this loop
      Unrolled inner loop 4 times
      Used combined stores for 1 stores
      Generated 1 prefetch instructions for this loop
      Unrolled inner loop 4 times
      Used combined stores for 1 stores
      Generated 2 prefetch instructions for this loop
      Unrolled inner loop 4 times
      Used combined stores for 1 stores
```


LP43020
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Potential Recursion

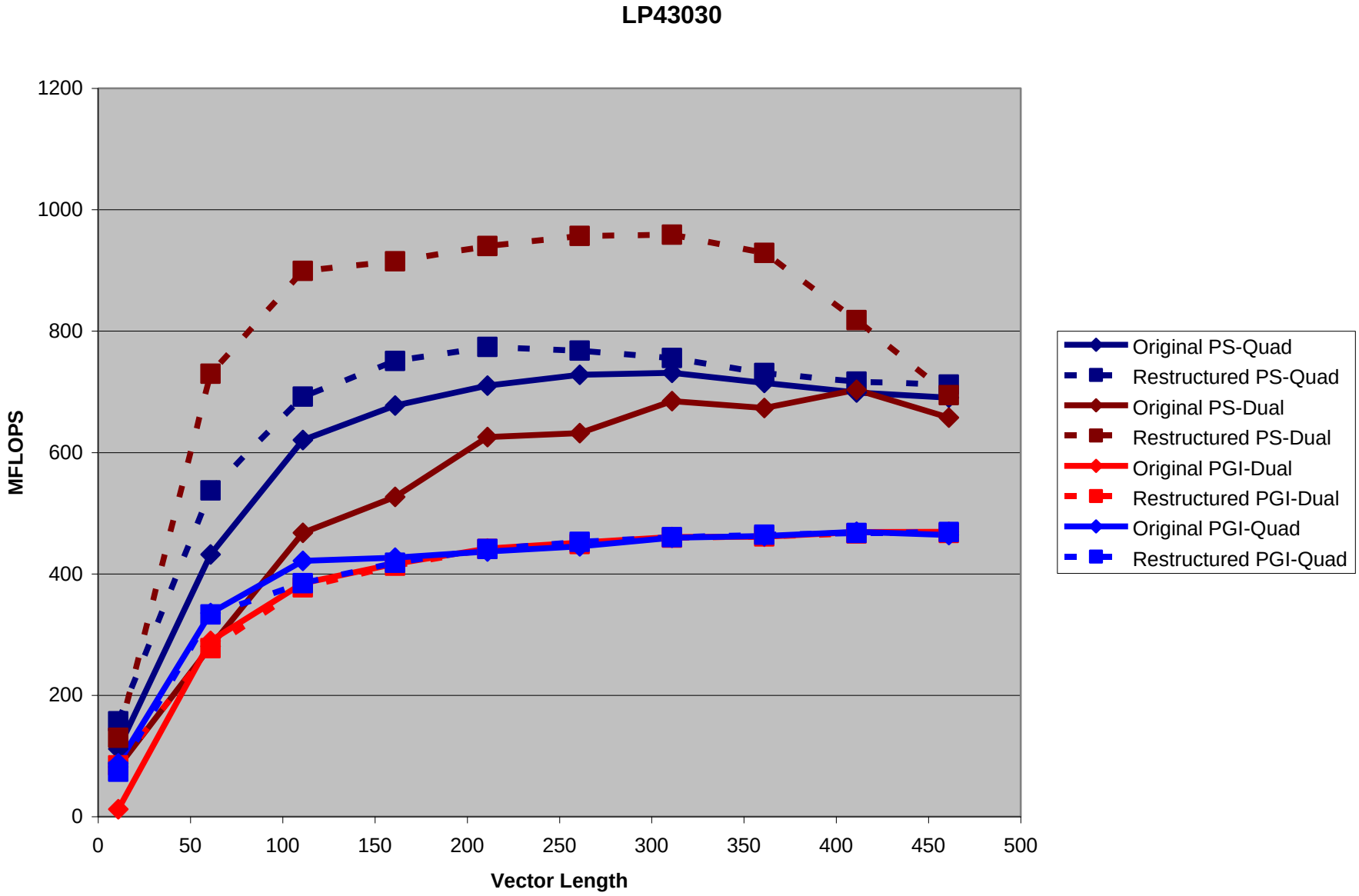
```
(   39) C      THE ORIGINAL
(   40)
(   41)       DO  43030  I = 2, N
(   42)        DO  43030  K = 1, I-1
(   43)         A(I)= A(I)  + B(I,K) * A(I-K)
(   44) 43030 CONTINUE
```

PGI
42, Generated vector sse code for inner loop
Pathscale
(lp43030.f:42) Non-contiguous array "B(_BLNK__.4000.0)" reference exists. Loop was not vectorized.

Rewrite

```
(    67) C      THE RESTRUCTURED
(    68)
(    69)       DO 43031 I = 2, N
(    70) CVD$ NODEPCHK
(    71) CDIR$ IVDEP
(    72) *VDIR NODEP
(    73)        DO 43031 K = 1, I-1
(    74)         A(I) = A(I)  + B(I,K) * A(I-K)
(    75) 43031 CONTINUE
(    76)
```

PGI
73, Generated vector sse code for inner loop
Pathscale
(lp43030.f:73) Non-contiguous array "B(_BLNK__.4000.0)" reference exists.
Loop was not vectorized.


LP43030
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Potential Recursion

```
(   45)       DO 43040 J = 2, 8
(   46)        N1 = J
(   47)        N2 = J - 1
(   48)        DO 43040 I = 2, N
(   49)         A(I,N1) = A(I-1,N2) * B(I,J) + C(I)
(   50) 43040 CONTINUE
(   51)
```

PGI

48, Loop not vectorized: data dependency
Loop unrolled 2 times

Pathscale

(lp43040.f:48) LOOP WAS VECTORIZED.

Rewrite

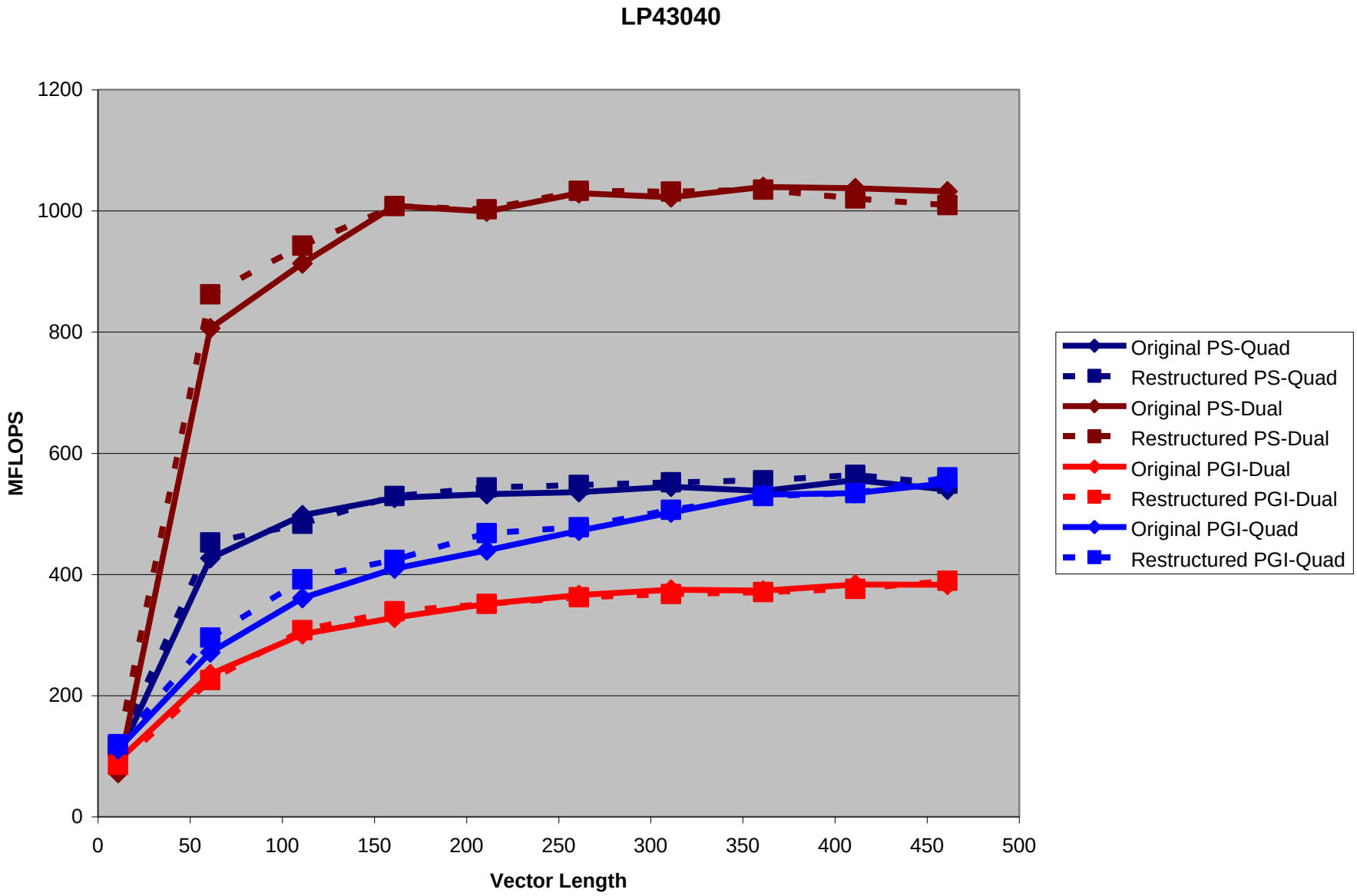
```
(    75) C       THE RESTRUCTURED
(    76)
(    77)        DO 43041 J = 2, 8
(    78)         N1 = J
(    79)         N2 = J - 1
(    80) CVD$ NODEPCHK
(    81) CDIR$ IVDEP
(    82) *VDIR NODEP
(    83)         DO 43041 I = 2, N
(    84)          A(I,N1) = A(I-1,N2) * B(I,J) + C(I)
(    85) 43041 CONTINUE
(    86)
```

PGI

83, Loop not vectorized: data dependency
Loop unrolled 2 times

Pathscale

(lp43040.f:83) LOOP WAS VECTORIZED.


LP43040
MFLOPS
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
400
450
500
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Potential Recursion

```
(   40) C      THE ORIGINAL
(   41)
(   42)       DO 43050 I = 1, N
(   43)        A(I) = A(I+N2) * A(I+N3) + A(I+N4)
(   44) 43050 CONTINUE
```

PGI

```
  42, Generated an alternate loop for the inner loop
      Generated vector sse code for inner loop
      Generated 3 prefetch instructions for this loop
      Generated vector sse code for inner loop
      Generated 3 prefetch instructions for this loop
```

Pathscale

(lp43050.f:42) LOOP WAS VECTORIZED.

Rewrite

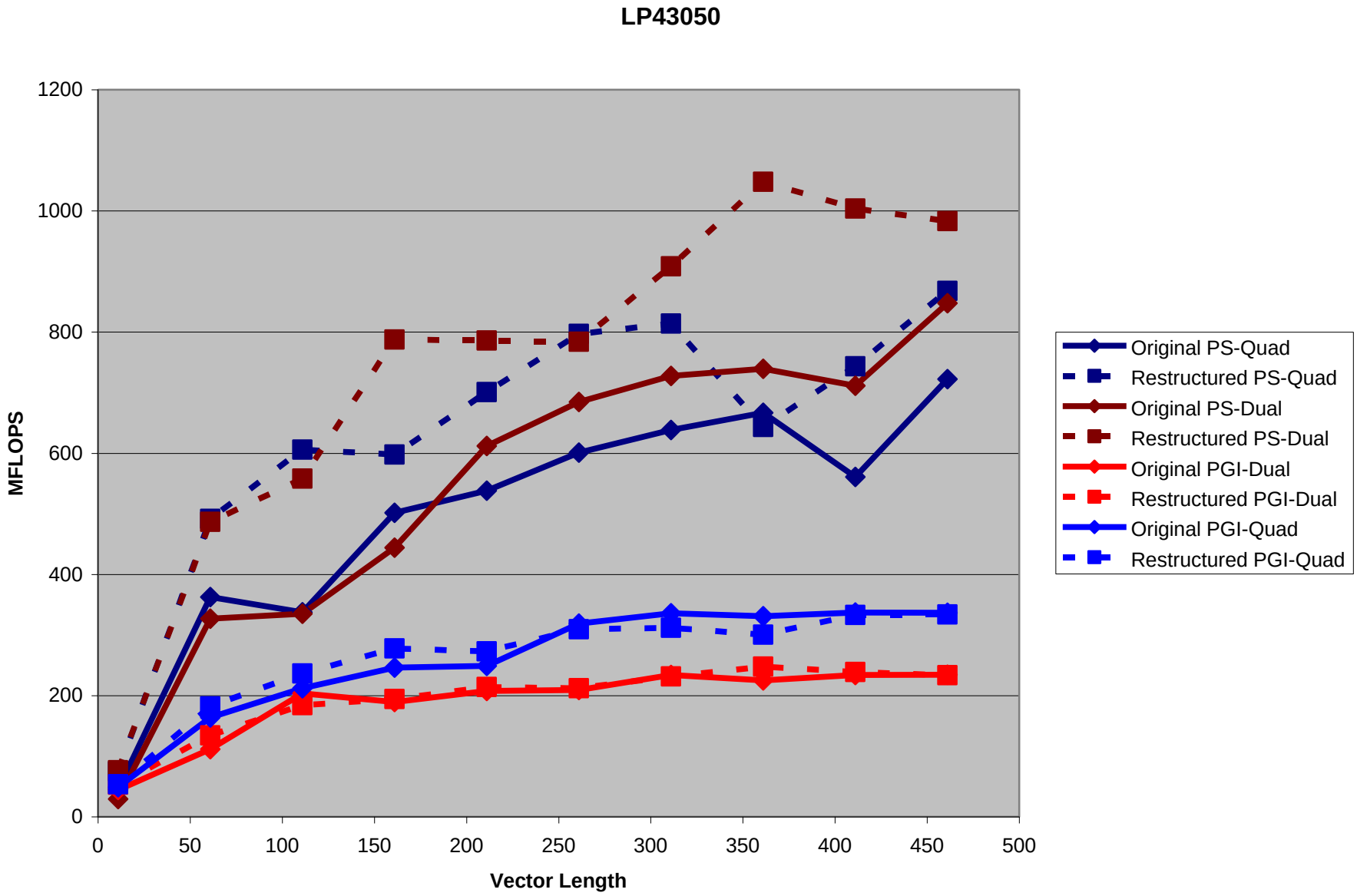
```
(    63) C      THE RESTRUCTURED
(    64)
(    65) CVD$ NODEPCHK
(    66) CDIR$ IVDEP
(    67) *VDIR NODEP
(    68)       DO 43051 I = 2, N
(    69)        A(I) = A(I+N2) * A(I+N3) + A(I+N4)
(    70) 43051 CONTINUE
(    71)
```

PGI

68, Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop

Pathscale

(lp43050.f:68) LOOP WAS VECTORIZED.


LP43050
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Potential Recursion

```
(   72) C      THE ORIGINAL
(   73)
(   74)       DO 43060 KX = 2, 3
(   75)        DO 43060 KY = 2, N
(   76)          D(KY) = A(KX,KY+1,NL12) - A(KX,KY-1,NL12)
(   77)          E(KY) = B(KX,KY+1,NL22) - B(KX,KY-1,NL22)
(   78)          F(KY) = C(KX,KY+1,NL32) - C(KX,KY-1,NL32)
(   79)          A(KX,KY,NL11) = A(KX,KY,NL11)
(   80)      *   + C1*D(KY)            + C2*E(KY)         + C3*F(KY)
(   81)      *   + C0*(A(KX+1,KY,NL1) - 2.*A(KX,KY,NL1) + A(KX-1,KY,NL1))
(   82)          B(KX,KY,NL21) = B(KX,KY,NL21)
(   83)      *   + C4*D(KY)            + C5*E(KY)         + C6*F(KY)
(   84)      *   + C0*(B(KX+1,KY,NL1) - 2.*B(KX,KY,NL1) + B(KX-1,KY,NL1))
(   85)          C(KX,KY,NL31) = C(KX,KY,NL31)
(   86)      *   + C7*D(KY)            + C8*E(KY)         + C9*F(KY)
(   87)      *   + C0*(C(KX+1,KY,NL1) - 2.*C(KX,KY,NL1) + C(KX-1,KY,NL1))
(   88) 43060 CONTINUE
```

PGI

74, Loop not vectorized: loop count too small
Outer loop unrolled 2 times (completely unrolled)
75, Generated vector sse code for inner loop

Pathscale

(lp43060.f:75) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

```
(  121)        DO  43061 KX = 2, 3
(  122)
(  123) CVD$ NODEPCHK
(  124) CDIR$ IVDEP
(  125) *VDIR NODEP
(  126)
(  127)         DO  43061 KY = 2, N
(  128)          D(KY) = A(KX,KY+1,NL12) - A(KX,KY-1,NL12)
(  129)          E(KY) = B(KX,KY+1,NL22) - B(KX,KY-1,NL22)
(  130)          F(KY) = C(KX,KY+1,NL32) - C(KX,KY-1,NL32)
(  131)          A(KX,KY,NL11) = A(KX,KY,NL11)
(  132)       *   + C1*D(KY)            + C2*E(KY)         + C3*F(KY)
(  133)       *   + C0*(A(KX+1,KY,NL1) - 2.*A(KX,KY,NL1) + A(KX-1,KY,NL1))
(  134)          B(KX,KY,NL21) = B(KX,KY,NL21)
(  135)       *   + C4*D(KY)            + C5*E(KY)         + C6*F(KY)
(  136)       *   + C0*(B(KX+1,KY,NL1) - 2.*B(KX,KY,NL1) + B(KX-1,KY,NL1))
(  137)          C(KX,KY,NL31) = C(KX,KY,NL31)
(  138)       *   + C7*D(KY)            + C8*E(KY)         + C9*F(KY)
(  139)       *   + C0*(C(KX+1,KY,NL1) - 2.*C(KX,KY,NL1) + C(KX-1,KY,NL1))
(  140) 43061 CONTINUE
(  141)
```

PGI

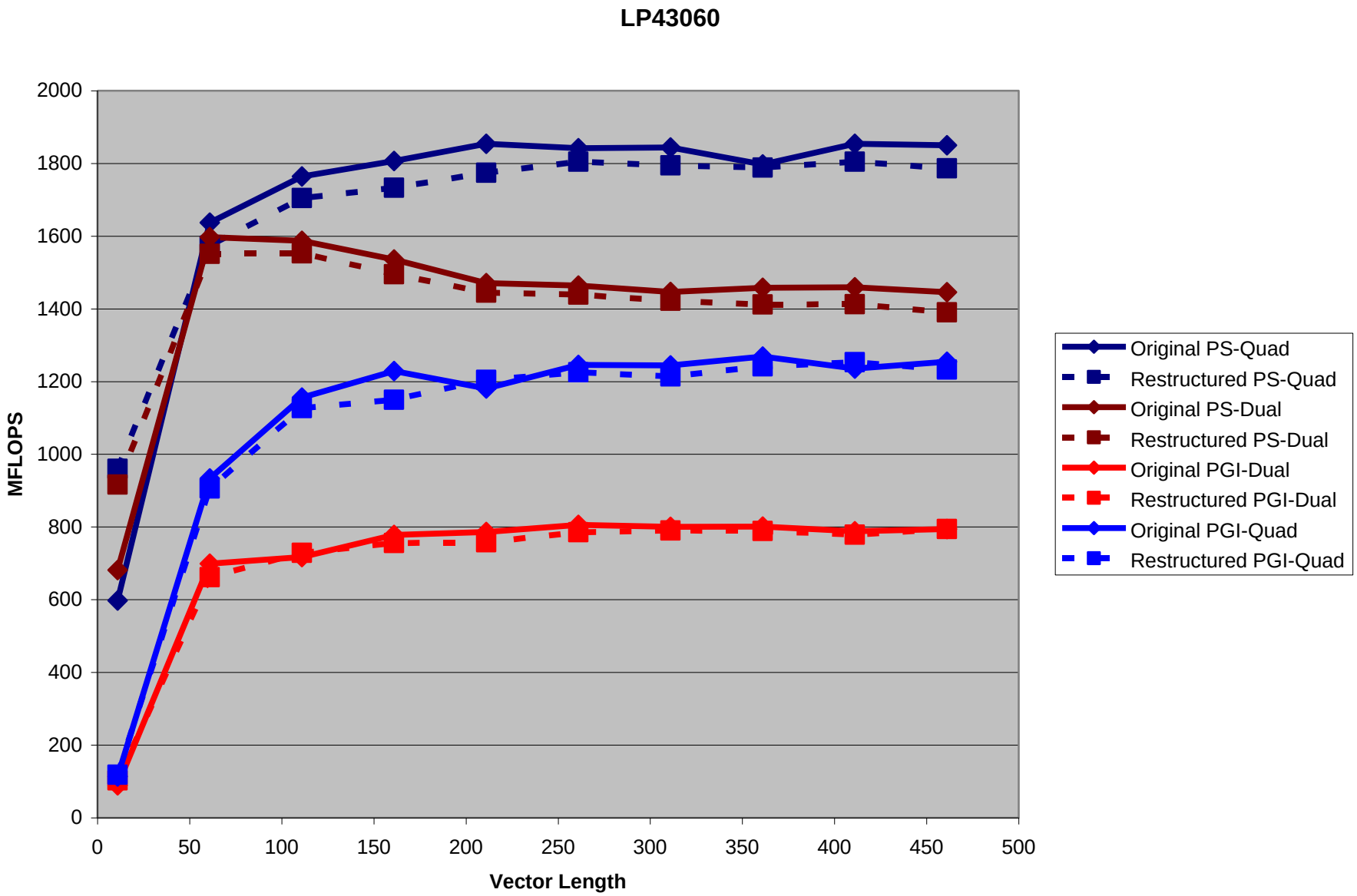
121, Loop not vectorized: loop count too small

Outer loop unrolled 2 times (completely unrolled)

127, Generated vector sse code for inner loop

Pathscale

(lp43060.f:127) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.


LP43060
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Potential Recursion

```
(    55) C      THE ORIGINAL
(    56)
(    57)       DO 43070 I = 1, N
(    58)        A(IA(I)) = A(IA(I)) + C0 * B(I)
(    59) 43070 CONTINUE
(    60)
```

PGI

57, Loop not vectorized: data dependency

Loop unrolled 4 times

Pathscale

(lp43070.f:57) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

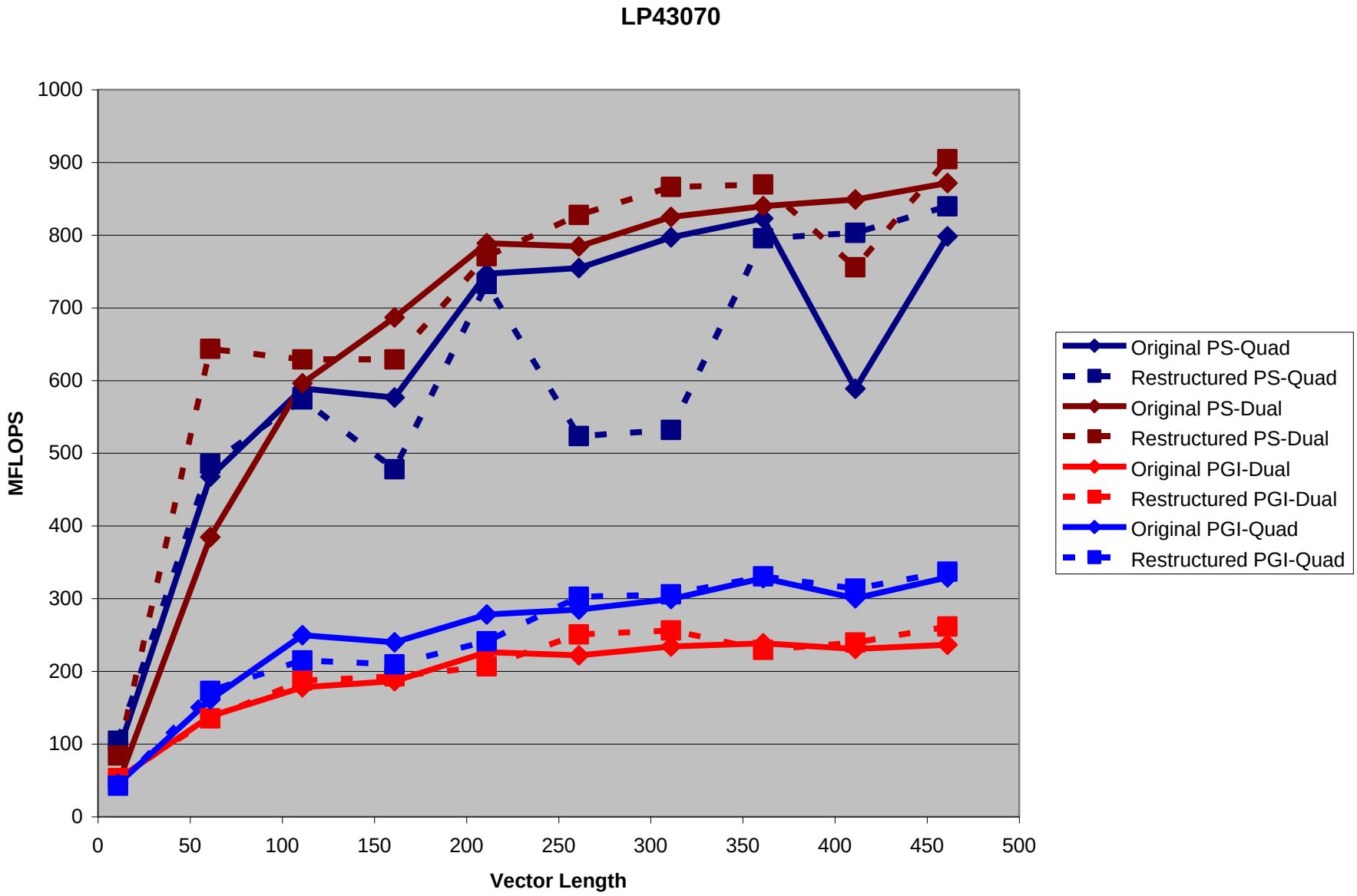
```
(    87) CDIR$ IVDEP
(    88) CVD$ NODEPCHK
(    89) *VDIR NODEP
(    90)       DO 43071 I = 1, N
(    91)        A(IA(I)) = A(IA(I)) + C0 * B(I)
(    92) 43071 CONTINUE
(    93)
```

PGI

90, Loop unrolled 4 times

Pathscale

(lp43070.f:90) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.


LP43070
MFLOPS
Vector Length
0
100
200
300
400
500
600
700
800
900
1000
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Wrap Around Scalar

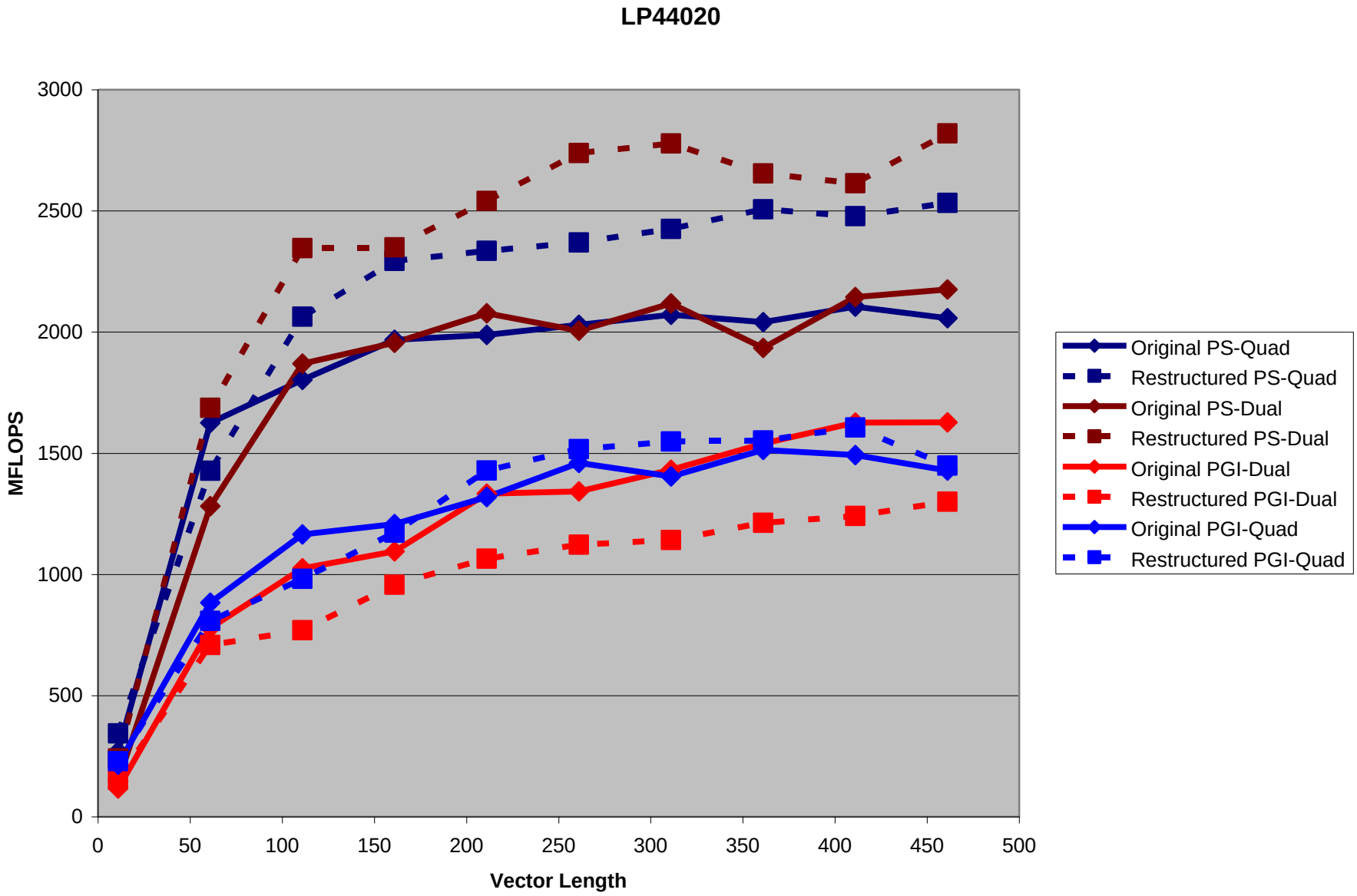
```
(    41)        BR =0.0
(    42)        DO 44020 I = 1, N
(    43)         BL = BR
(    44)         BR = (I-1) * DELB
(    45)         A(I) = (BR - BL) * C(I) + (BR**2 - BL**2) * C(I)**2
(    46)  44020 CONTINUE
```

42, Loop not vectorized: mixed data types
Generated an alternate loop for the inner loop
Loop not vectorized: mixed data types
Unrolled inner loop 4 times
Used combined stores for 1 stores
Generated 1 prefetch instructions for this loop
Loop not vectorized: mixed data types
Unrolled inner loop 4 times
Used combined stores for 1 stores
Generated 1 prefetch instructions for this loop

Rewrite

```
(    67)        BSQ(1) = 0.0
(    68)        A(1)   = 0.0
(    69)        B = 0.0
(    70)        DO 44022 I = 2, N
(    71)        B   = B   + DELB
(    72)        BSQ(I) = B ** 2
(    73)         A(I)   = C(I) * ( DELB + C(I) * (BSQ(I) - BSQ(I-1)))
(    74)  44022 CONTINUE
```

70, Generated 2 alternate loops for the inner loop
Unrolled inner loop 4 times
Generated 2 prefetch instructions for this loop
Unrolled inner loop 4 times
Used combined stores for 1 stores
Generated 2 prefetch instructions for this loop
Unrolled inner loop 4 times
Used combined stores for 1 stores
Generated 2 prefetch instructions for this loop


LP44020
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Maximum within Loop

```
(   61)        DO 44040 I = 2, N
(   62)         RR          = 1. / A(I,1)
(   63)         U           = A(I,2) * RR
(   64)         V           = A(I,3) * RR
(   65)         W           = A(I,4) * RR
(   66)         SNDSP       = SQRT (GD * (A(I,5) * RR + .5* (U*U + V*V + W*W)))
(   67)         SIGA        = ABS (XT + U*B(I) + V*C(I) + W*D(I))
(   68)       *               + SNDSP * SQRT (B(I)**2 + C(I)**2 + D(I)**2)
(   69)         SIGB        = ABS (YT + U*E(I) + V*F(I) + W*G(I))
(   70)       *               + SNDSP * SQRT (E(I)**2 + F(I)**2 + G(I)**2)
(   71)         SIGC        = ABS (ZT + U*H(I) + V*R(I) + W*S(I))
(   72)       *               + SNDSP * SQRT (H(I)**2 + R(I)**2 + S(I)**2)
(   73)         SIGABC      = AMAX1 (SIGA, SIGB, SIGC)
(   74)         IF (SIGABC.GT.SIGMAX) THEN
(   75)         IMAX        = I
(   76)         SIGMAX      = SIGABC
(   77)         ENDIF
(   78) 44040 CONTINUE
```

PGI

61, Generated an alternate loop for the inner loop
    Generated vector sse code for inner loop
    Generated 8 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 8 prefetch instructions for this loop

Pathscale

(lp44040.f:62) Expression rooted at op "OPC_IF"(line 63) is not vectorizable. Loop was not vectorized.

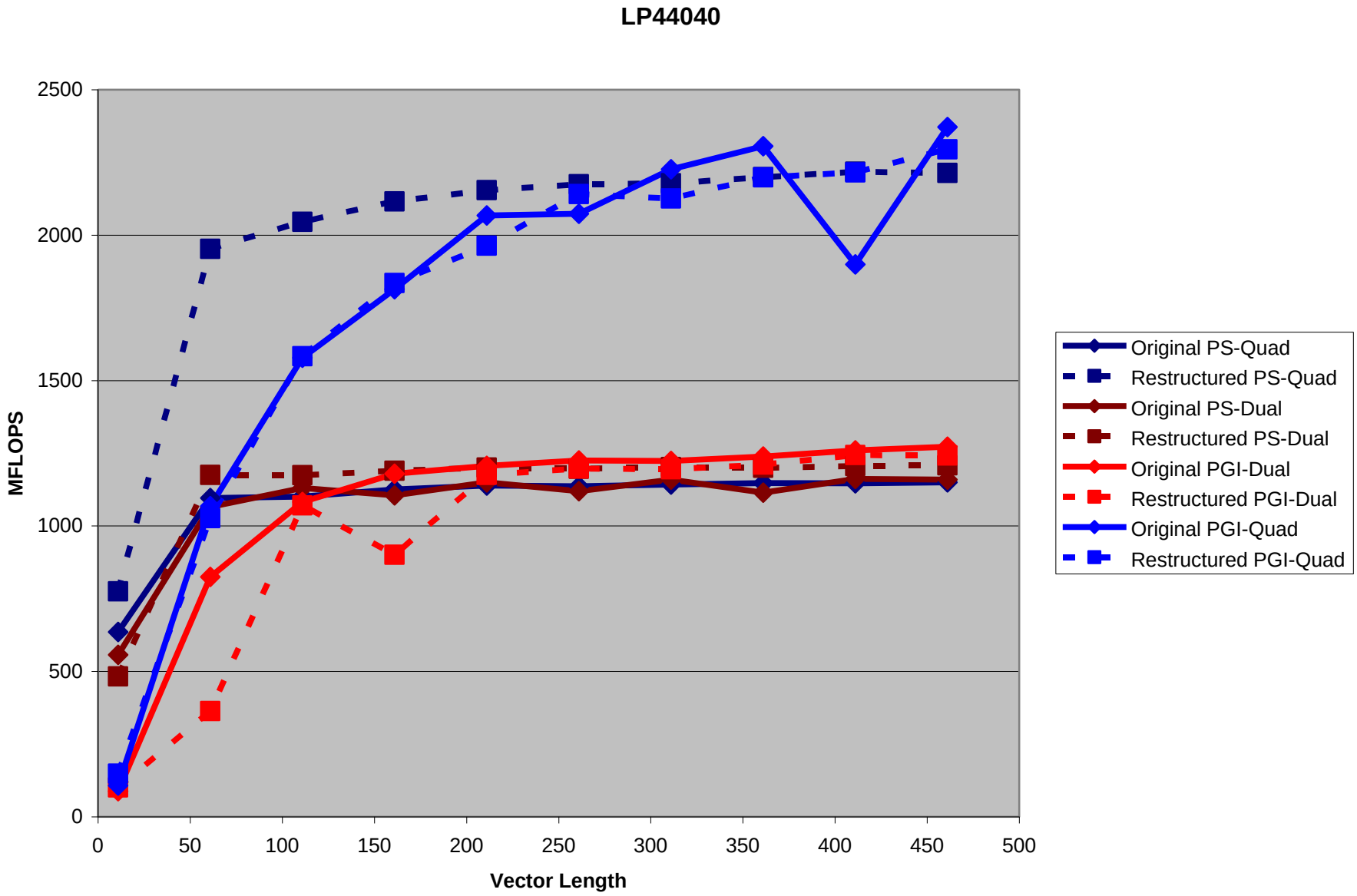
```
(   98)        DO 44041 I = 2, N
(   99)         RR          = 1. / A(I,1)
(  100)         U           = A(I,2) * RR
(  101)         V           = A(I,3) * RR
(  102)         W           = A(I,4) * RR
(  103)         SNDSP       = SQRT (GD * (A(I,5) * RR + .5* (U*U + V*V + W*W)))
(  104)         SIGA        = ABS (XT + U*B(I) + V*C(I) + W*D(I))
(  105)       *               + SNDSP * SQRT (B(I)**2 + C(I)**2 + D(I)**2)
(  106)         SIGB        = ABS (YT + U*E(I) + V*F(I) + W*G(I))
(  107)       *               + SNDSP * SQRT (E(I)**2 + F(I)**2 + G(I)**2)
(  108)         SIGC        = ABS (ZT + U*H(I) + V*R(I) + W*S(I))
(  109)       *               + SNDSP * SQRT (H(I)**2 + R(I)**2 + S(I)**2)
(  110)         VSIGABC(I) = AMAX1 (SIGA, SIGB, SIGC)
(  111) 44041 CONTINUE
(  112)
(  113)        DO 44042 I = 2, N
(  114)         IF (VSIGABC(I) .GT. SIGMAX) THEN
(  115)          IMAX       = I
(  116)          SIGMAX     = VSIGABC(I)
(  117)         ENDIF
(  118) 44042 CONTINUE
(  119)
```

PGI

98, Generated 2 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
113, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop

Pathscale

(lp44040.f:100) LOOP WAS VECTORIZED.
(lp44040.f:115) Expression rooted at op "OPC_IF"(line 116) is not vectorizable. Loop was not vectorized.


LP44040
MFLOPS
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
400
450
500
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Matrix Multiply

```
(    44) C      THE ORIGINAL
(    45)
(    46)       DO 44050 I = 1, N
(    47)        DO 44050 J = 1, N
(    48)         A(I,J) = 0.0
(    49)         DO 44050 K = 1, N
(    50)          A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    51) 44050 CONTINUE
(    52)
```

PGI

49, Generated 2 alternate loops for the inner loop
    Generated vector sse code for inner loop
    Generated 1 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 1 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 1 prefetch instructions for this loop

Pathscale

(lp44050.f:46) Loop has too many loop invariants. Loop was not vectorized.
(lp44050.f:46) LOOP WAS VECTORIZED.
(lp44050.f:46) LOOP WAS VECTORIZED.
(lp44050.f:46) LOOP WAS VECTORIZED.

Rewritten

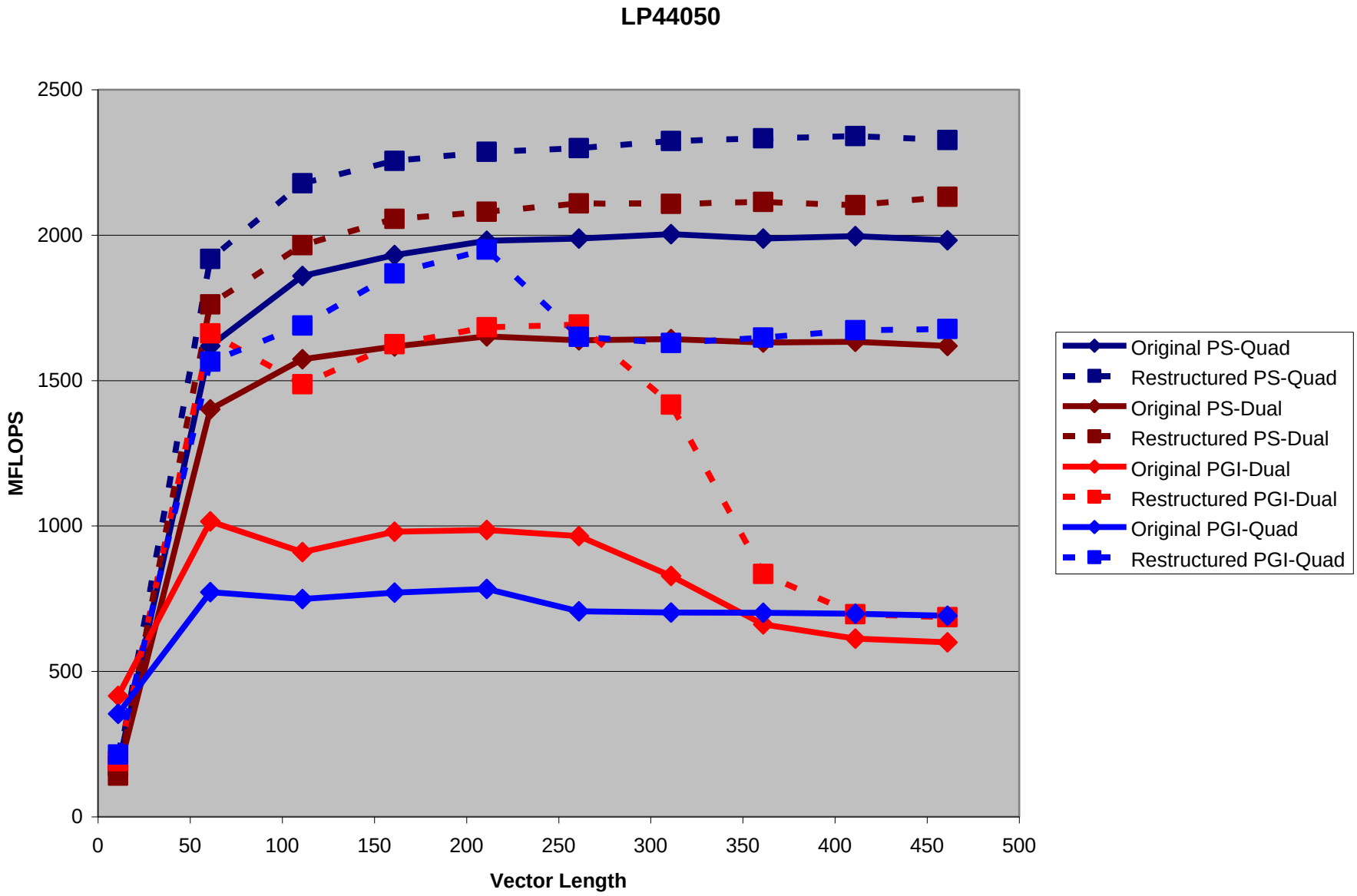
```
(    77) C      THE RESTRUCTURED
(    78)
(    79)       DO 44051 J = 1, N
(    80)        DO 44051 I = 1, N
(    81)         A(I,J) = 0.0
(    82) 44051 CONTINUE
(    83)
(    84)       DO 44052 K = 1, N
(    85)        DO 44052 J = 1, N
(    86)         DO 44052 I = 1, N
(    87)          A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    88) 44052 CONTINUE
(    89) C
```

PGI

79, Loop not vectorized: contains call
80, Memory zero idiom, loop replaced by memzero call
84, Interchange produces reordered loop nest: 85, 84, 86
86, Generated 3 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 2 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 2 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 2 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 2 prefetch instructions for this loop

Pathscale

(lp44050.f:80) LOOP WAS VECTORIZED.
(lp44050.f:80) LOOP WAS VECTORIZED.
(lp44050.f:86) Loop has too many loop invariants. Loop was not vectorized.
(lp44050.f:86) LOOP WAS VECTORIZED.
(lp44050.f:86) LOOP WAS VECTORIZED.
(lp44050.f:86) LOOP WAS VECTORIZED.


LP44050
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Nested Loops

```
(    47)         DO 45020 I = 1, N
(    48)          F(I) = A(I) + .5
(    49)          DO 45020 J = 1, 10
(    50)           D(I,J) = B(J) * F(I)
(    51)           DO 45020 K = 1, 5
(    52)            C(K,I,J) = D(I,J) * E(K)
(    53)  45020 CONTINUE
```

PGI

49, Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop
Loop unrolled 2 times (completely unrolled)

Pathscale

(lp45020.f:48) LOOP WAS VECTORIZED.
(lp45020.f:48) Non-contiguous array "C(_BLNK__.0.0)" reference exists. Loop was not vectorized.

Rewrite

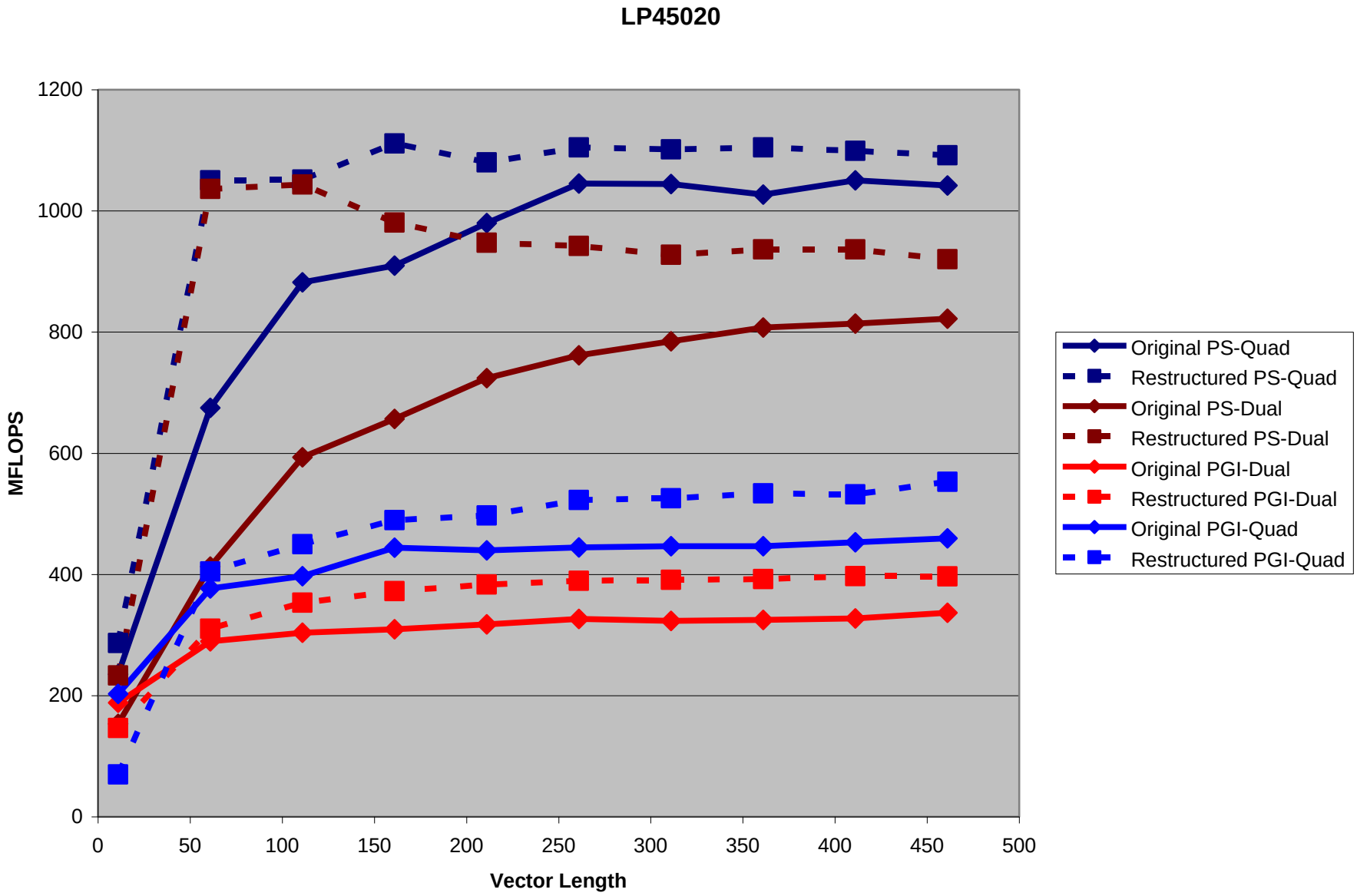
```
(   71)       DO 45021 I = 1,N
(   72)        F(I) = A(I) + .5
(   73) 45021 CONTINUE
(   74)
(   75)       DO 45022 J = 1, 10
(   76)        DO 45022 I = 1, N
(   77)         D(I,J) = B(J) * F(I)
(   78) 45022 CONTINUE
(   79)
(   80)       DO 45023 K = 1, 5
(   81)        DO 45023 J = 1, 10
(   82)         DO 45023 I = 1, N
(   83)          C(K,I,J) = D(I,J) * E(K)
(   84) 45023 CONTINUE
```

PGI

```
     73, Generated an alternate loop for the inner loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
     78, Generated 2 alternate loops for the inner loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
     82, Interchange produces reordered loop nest: 83, 84, 82
         Loop unrolled 5 times (completely unrolled)
     84, Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
```

Pathscale

```
(lp45020.f:73) LOOP WAS VECTORIZED.
(lp45020.f:78) LOOP WAS VECTORIZED.
(lp45020.f:78) LOOP WAS VECTORIZED.
(lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference
exists. Loop was not vectorized.
(lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference
exists. Loop was not vectorized.
```


LP45020
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Nx4 Matmul

```
(    45)         DO 46020 I = 1,N
(    46)          DO 46020 J = 1,4
(    47)           A(I,J) = 0.
(    48)           DO 46020 K = 1,4
(    49)            A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    50)  46020 CONTINUE
```

PGI

46, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
47, Loop unrolled 4 times (completely unrolled)
49, Loop not vectorized: loop count too small
Loop unrolled 4 times (completely unrolled)

Pathscale

(lp46020.f:46) Loop has too many loop invariants. Loop was not vectorized.

Rewrite

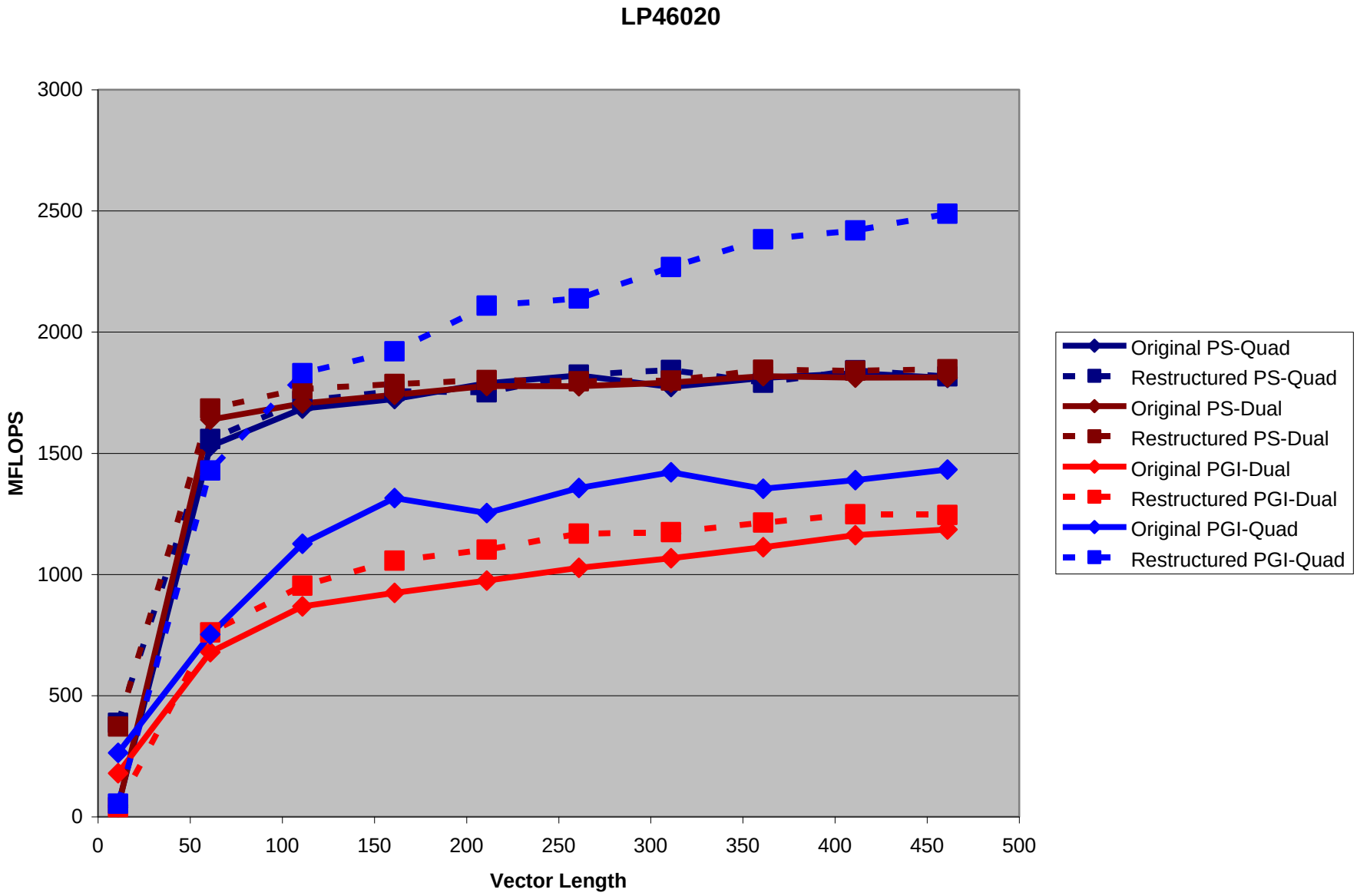
```
(    68) C        THE RESTRUCTURED
(    69)
(    70)         DO 46021 I = 1, N
(    71)           A(I,1) = B(I,1) * C(1,1) + B(I,2) * C(2,1)
(    72)       *          + B(I,3) * C(3,1) + B(I,4) * C(4,1)
(    73)           A(I,2) = B(I,1) * C(1,2) + B(I,2) * C(2,2)
(    74)       *          + B(I,3) * C(3,2) + B(I,4) * C(4,2)
(    75)           A(I,3) = B(I,1) * C(1,3) + B(I,2) * C(2,3)
(    76)       *          + B(I,3) * C(3,3) + B(I,4) * C(4,3)
(    77)           A(I,4) = B(I,1) * C(1,4) + B(I,2) * C(2,4)
(    78)       *          + B(I,3) * C(3,4) + B(I,4) * C(4,4)
(    79) 46021 CONTINUE
(    80)
```

PGI

70, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop

Pathscale

(lp46020.f:70) Loop has too many loop invariants. Loop was not vectorized.


LP46020
MFLOPS
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
400
450
500
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Traditional MATMUL

```
(    41) C       THE ORIGINAL
(    42)
(    43)         DO 46030 J  = 1, N
(    44)          DO 46030 I = 1, N
(    45)           A(I,J) = 0.
(    46) 46030 CONTINUE
(    47)
(    48)         DO 46031   K = 1, N
(    49)          DO 46031  J = 1, N
(    50)           DO 46031 I = 1, N
(    51)            A(I,J) = A(I,J) + B(I,K) * C(K,J)
(    52) 46031 CONTINUE
(    53)
```

PGI

```
   43, Loop not vectorized: contains call
    44, Memory zero idiom, loop replaced by memzero call
    48, Interchange produces reordered loop nest: 49, 48, 50
    50, Generated 3 alternate loops for the inner loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
```

Pathscale

```
(lp46030.f:44) LOOP WAS VECTORIZED.
(lp46030.f:44) LOOP WAS VECTORIZED.
(lp46030.f:50) Loop has too many loop invariants. Loop was not
vectorized.
(lp46030.f:50) LOOP WAS VECTORIZED.
(lp46030.f:50) LOOP WAS VECTORIZED.
(lp46030.f:50) LOOP WAS VECTORIZED.
```

Rewrite

```
(   69) C       THE RESTRUCTURED
(   70)
(   71)       DO 46032  J = 1, N
(   72)        DO 46032 I = 1, N
(   73)         A(I,J)=0.
(   74) 46032 CONTINUE
(   75) C
(   76)       DO 46033   K = 1, N-5, 6
(   77)        DO 46033  J = 1, N
(   78)         DO 46033 I = 1, N
(   79)          A(I,J) = A(I,J) + B(I,K  ) * C(K  ,J)
(   80)      *                   + B(I,K+1) * C(K+1,J)
(   81)      *                   + B(I,K+2) * C(K+2,J)
(   82)      *                   + B(I,K+3) * C(K+3,J)
(   83)      *                   + B(I,K+4) * C(K+4,J)
(   84)      *                   + B(I,K+5) * C(K+5,J)
(   85) 46033 CONTINUE
(   86) C
(   87)       DO 46034  KK = K, N
(   88)        DO 46034  J = 1, N
(   89)         DO 46034 I = 1, N
(   90)          A(I,J) = A(I,J) + B(I,KK) * C(KK ,J)
(   91) 46034 CONTINUE
(   92)
```

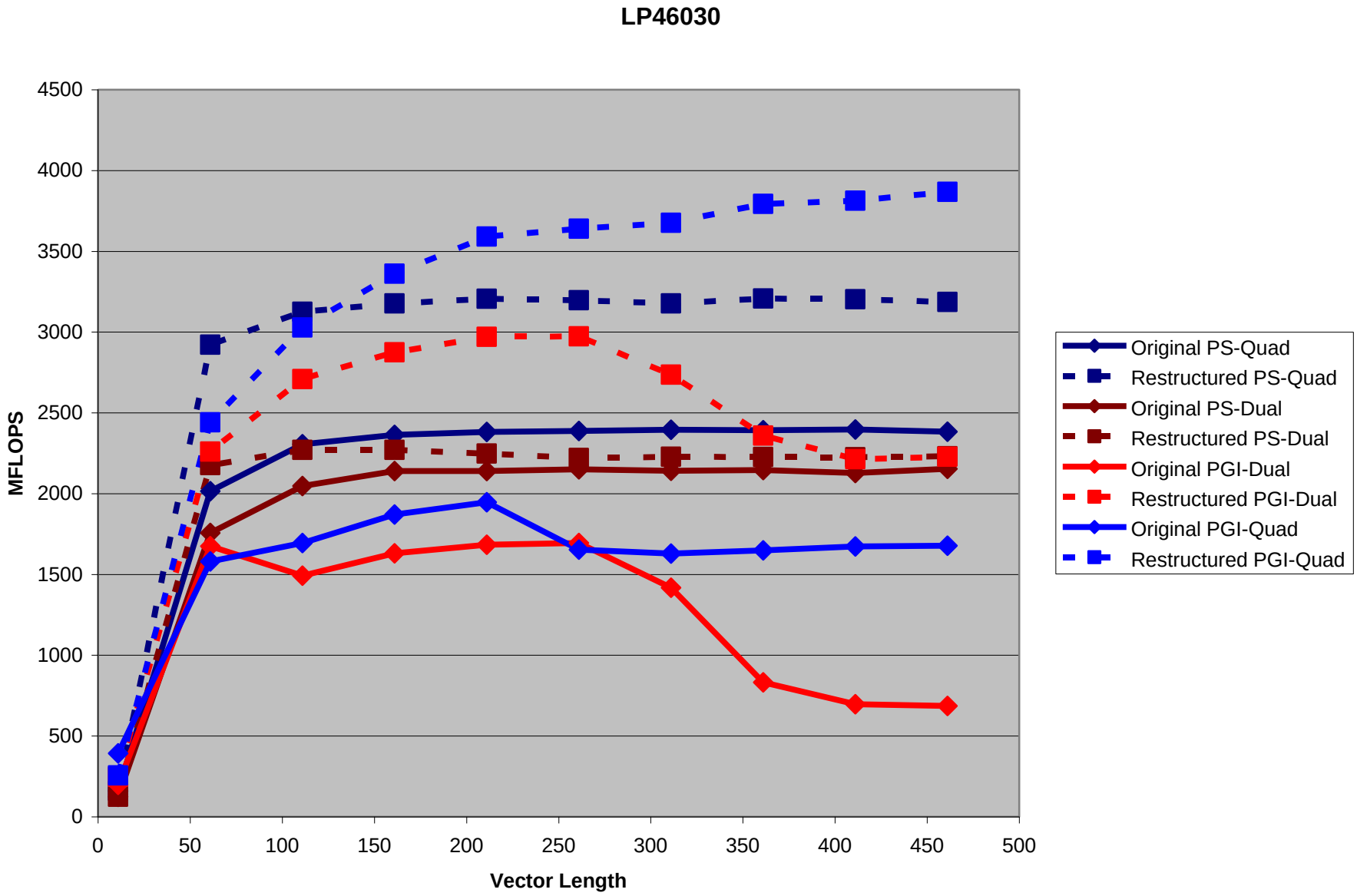
Rewrite

PGI

```
71, Loop not vectorized: contains call
72, Memory zero idiom, loop replaced by memzero call
78, Generated 3 alternate loops for the inner loop
    Generated vector sse code for inner loop
    Generated 7 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 7 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 7 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 7 prefetch instructions for this loop
87, Interchange produces reordered loop nest: 88, 87, 89
89, Generated 3 alternate loops for the inner loop
    Generated vector sse code for inner loop
    Generated 2 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 2 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 2 prefetch instructions for this loop
    Generated vector sse code for inner loop
    Generated 2 prefetch instructions for this loop
```

Rewrite

Pathscale
(lp46030.f:72) LOOP WAS VECTORIZED.
(lp46030.f:72) LOOP WAS VECTORIZED.
(lp46030.f:78) LOOP WAS VECTORIZED.
(lp46030.f:78) LOOP WAS VECTORIZED.
(lp46030.f:89) Loop has too many loop invariants. Loop was not vectorized.
(lp46030.f:89) LOOP WAS VECTORIZED.
(lp46030.f:89) LOOP WAS VECTORIZED.
(lp46030.f:89) LOOP WAS VECTORIZED.


LP46030
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Big Loop

```
(   52) C      THE ORIGINAL
(   53)
(   54)       DO 47020   J = 1, JMAX
(   55)        DO 47020  K = 1, KMAX
(   56)         DO 47020 I = 1, IMAX
(   57)           JP         = J + 1
(   58)           JR         = J - 1
(   59)           KP         = K + 1
(   60)           KR         = K - 1
(   61)           IP         = I + 1
(   62)           IR         = I - 1
(   63)           IF (J .EQ. 1)     GO TO 50
(   64)            IF( J .EQ. JMAX) GO TO 51
(   65)             XJ = ( A(I,JP,K) - A(I,JR,K) ) * DA2
(   66)             YJ = ( B(I,JP,K) - B(I,JR,K) ) * DA2
(   67)             ZJ = ( C(I,JP,K) - C(I,JR,K) ) * DA2
(   68)             GO TO 70
(   69)    50     J1 = J + 1
(   70)           J2 = J + 2
(   71)           XJ = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
(   72)           YJ = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
(   73)           ZJ = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
(   74)           GO TO 70
(   75)    51     J1 = J - 1
(   76)           J2 = J - 2
(   77)           XJ = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
(   78)           YJ = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
(   79)           ZJ = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
(   80)    70     CONTINUE
(   81)           IF (K .EQ. 1)     GO TO 52
(   82)            IF (K .EQ. KMAX) GO TO 53
(   83)             XK = ( A(I,J,KP) - A(I,J,KR) ) * DB2
(   84)             YK = ( B(I,J,KP) - B(I,J,KR) ) * DB2
(   85)             ZK = ( C(I,J,KP) - C(I,J,KR) ) * DB2
(   86)             GO TO 71
```

Big Loop

```
(   87)     52    K1 = K + 1
(   88)           K2 = K + 2
(   89)           XK = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
(   90)           YK = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
(   91)           ZK = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
(   92)           GO TO 71
(   93)     53    K1 = K - 1
(   94)           K2 = K - 2
(   95)           XK = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
(   96)           YK = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
(   97)           ZK = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
(   98)     71    CONTINUE
(   99)           IF (I .EQ. 1)     GO TO 54
(  100)            IF (I .EQ. IMAX) GO TO 55
(  101)              XI = ( A(IP,J,K) - A(IR,J,K) ) * DC2
(  102)              YI = ( B(IP,J,K) - B(IR,J,K) ) * DC2
(  103)              ZI = ( C(IP,J,K) - C(IR,J,K) ) * DC2
(  104)              GO TO 60
(  105)     54    I1 = I + 1
(  106)           I2 = I + 2
(  107)           XI = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(  108)           YI = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(  109)           ZI = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
(  110)           GO TO 60
(  111)     55    I1 = I - 1
(  112)           I2 = I - 2
(  113)           XI = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
(  114)           YI = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
(  115)           ZI = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
(  116)     60    CONTINUE
(  117)           DINV     = XJ * YK * ZI  +  YJ * ZK * XI  +  ZJ * XK * YI
(  118)       *              - XJ * ZK * YI  -  YJ * XK * ZI  -  ZJ * YK * XI
(  119)           D(I,J,K) = 1. / (DINV + 1.E-20)
(  120) 47020 CONTINUE
(  121)
```

PGI

55, Invariant if transformation

Loop not vectorized: loop count too small

56, Invariant if transformation

Pathscale

Nothing

Re-Write

```
(  141) C      THE RESTRUCTURED
(  142)
(  143)       DO 47029 J = 1, JMAX
(  144)        DO 47029 K = 1, KMAX
(  145)
(  146)         IF(J.EQ.1)THEN
(  147)
(  148)         J1         = 2
(  149)         J2         = 3
(  150)         DO 47021 I = 1, IMAX
(  151)          VAJ(I) = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
(  152)          VBJ(I) = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
(  153)          VCJ(I) = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
(  154) 47021   CONTINUE
(  155)
(  156)         ELSE IF(J.NE.JMAX) THEN
(  157)
(  158)         JP         = J+1
(  159)         JR         = J-1
(  160)         DO 47022 I = 1, IMAX
(  161)          VAJ(I) = ( A(I,JP,K) - A(I,JR,K) ) * DA2
(  162)          VBJ(I) = ( B(I,JP,K) - B(I,JR,K) ) * DA2
(  163)          VCJ(I) = ( C(I,JP,K) - C(I,JR,K) ) * DA2
(  164) 47022   CONTINUE
(  165)
(  166)         ELSE
(  167)
(  168)         J1         = JMAX-1
(  169)         J2         = JMAX-2
(  170)         DO 47023 I = 1, IMAX
(  171)          VAJ(I) = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
(  172)          VBJ(I) = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
(  173)          VCJ(I) = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
(  174) 47023   CONTINUE
(  175)
(  176)         ENDIF
```

Re-Write

```
(  178)          IF(K.EQ.1) THEN
(  179)
(  180)          K1          = 2
(  181)          K2          = 3
(  182)          DO 47024 I = 1, IMAX
(  183)           VAK(I) = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
(  184)           VBK(I) = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
(  185)           VCK(I) = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
(  186) 47024    CONTINUE
(  187)
(  188)          ELSE IF(K.NE.KMAX)THEN
(  189)
(  190)          KP          = K + 1
(  191)          KR          = K - 1
(  192)          DO 47025 I = 1, IMAX
(  193)           VAK(I) = ( A(I,J,KP) - A(I,J,KR) ) * DB2
(  194)           VBK(I) = ( B(I,J,KP) - B(I,J,KR) ) * DB2
(  195)           VCK(I) = ( C(I,J,KP) - C(I,J,KR) ) * DB2
(  196) 47025    CONTINUE
(  197)
(  198)          ELSE
(  199)
(  200)          K1          = KMAX - 1
(  201)          K2          = KMAX - 2
(  202)          DO 47026 I = 1, IMAX
(  203)           VAK(I) = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
(  204)           VBK(I) = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
(  205)           VCK(I) = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
(  206) 47026    CONTINUE
(  207)          ENDIF
(  208)
```

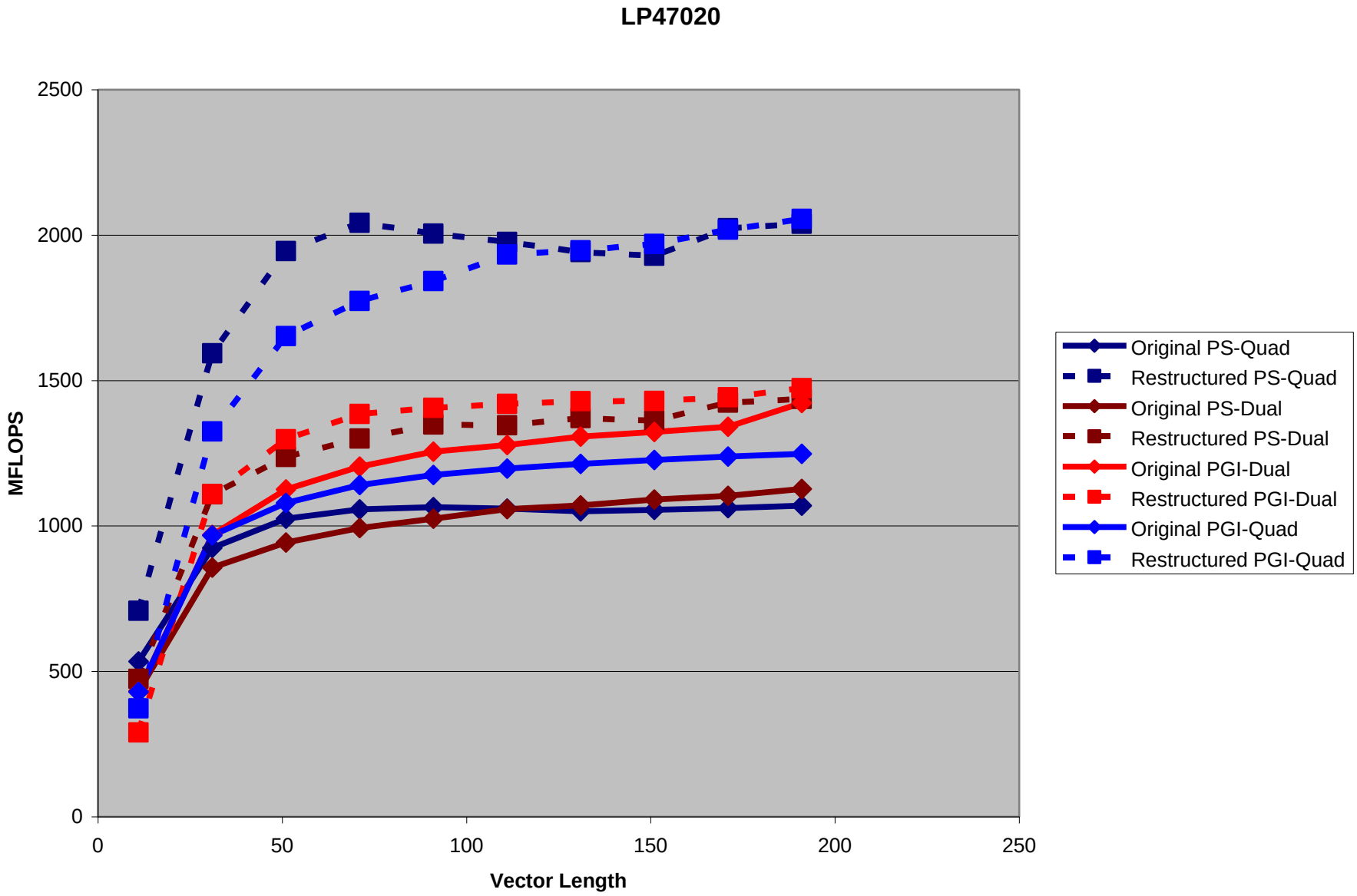
Re-Write

```
(  209)         I = 1
(  210)         I1         = 2
(  211)         I2         = 3
(  212)         VAI(I) = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(  213)         VBI(I) = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(  214)         VCI(I) = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
(  215)
(  216)         DO 47027 I = 2, IMAX-1
(  217)          IP         = I + 1
(  218)          IR         = I - 1
(  219)             VAI(I) = ( A(IP,J,K) - A(IR,J,K) ) * DC2
(  220)             VBI(I) = ( B(IP,J,K) - B(IR,J,K) ) * DC2
(  221)             VCI(I) = ( C(IP,J,K) - C(IR,J,K) ) * DC2
(  222) 47027    CONTINUE
(  223)
(  224)          I = IMAX
(  225)          I1          = IMAX - 1
(  226)          I2          = IMAX - 2
(  227)          VAI(I) = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
(  228)          VBI(I) = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
(  229)          VCI(I) = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
(  230)
(  231)          DO 47028 I = 1, IMAX
(  232)           DINV = VAJ(I) * VBK(I) * VCI(I) + VBJ(I) * VCK(I) * VAI(I)
(  233)       1         + VCJ(I) * VAK(I) * VBI(I) - VAJ(I) * VCK(I) * VBI(I)
(  234)       2         - VBJ(I) * VAK(I) * VCI(I) - VCJ(I) * VBK(I) * VAI(I)
(  235)             D(I,J,K) = 1. / (DINV + 1.E-20)
(  236) 47028    CONTINUE
(  237) 47029 CONTINUE

(  238)
```

PGI

144, Invariant if transformation
Loop not vectorized: loop count too small
150, Generated 3 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 8 prefetch instructions for this loop
160, Generated 4 alternate loops for the inner loop
Generated vector sse code for inner loop
Generated 6 prefetch instructions for this loop
Generated vector sse code for inner loop
o o o

Pathscale
(lp47020.f:132) LOOP WAS VECTORIZED.
(lp47020.f:150) LOOP WAS VECTORIZED.
(lp47020.f:160) LOOP WAS VECTORIZED.
(lp47020.f:170) LOOP WAS VECTORIZED.
(lp47020.f:182) LOOP WAS VECTORIZED.
(lp47020.f:192) LOOP WAS VECTORIZED.
(lp47020.f:202) LOOP WAS VECTORIZED.
(lp47020.f:216) LOOP WAS VECTORIZED.
(lp47020.f:231) LOOP WAS VECTORIZED.
(lp47020.f:248) LOOP WAS VECTORIZED.


LP47020
MFLOPS
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   48) C      THE ORIGINAL
(   49)
(   50)       DO 47030 I = 1, N
(   51)        A(I) = PROD * B(1,I) * A(I)
(   52)        IF (A(I) .LT. 0.0) A(I) = -A(I)
(   53)        IF (XL .LT. 0.0) A(I) = -A(I)
(   54)        IF (GAMMA) 47030, 47030, 100
(   55) 100     XL = -XL
(   56) 47030 CONTINUE
```

PGI
Nothing
Pathscale
(lp47030.f:50) Non-contiguous array "B(_BLNK__.4000.0)" reference exists. Loop was not vectorized.

```
(   77) C      THE RESTRUCTURED
(   78)
(   79)       DO 47031 I = 1, N
(   80)        A(I) = PROD * B(1,I) * A(I)
(   81)        A(I) = ABS (A(I))
(   82) 47031 CONTINUE
(   83)
(   84)       IF (GAMMA .LE. 0.) THEN
(   85)
(   86)        IF (XL .LT. 0.0) THEN
(   87)         DO 47032 I = 1, N
(   88)          A(I) = -A(I)
(   89) 47032   CONTINUE
(   90)        ENDIF
(   91)
(   92)       ELSE
(   93)
(   94)        IF (XL .LT. 0.0) THEN
(   95)         DO 47033 I = 1, N, 2
(   96)          A(I) = -A(I)
(   97) 47033   CONTINUE
(   98)        ENDIF
(   99)
(  100)        IF (XL .GT. 0.0) THEN
(  101)         DO 47034 I = 2, N, 2
(  102)          A(I) = -A(I)
(  103) 47034   CONTINUE
(  104)        ENDIF
(  105)
(  106)       ENDIF
(  107)
```

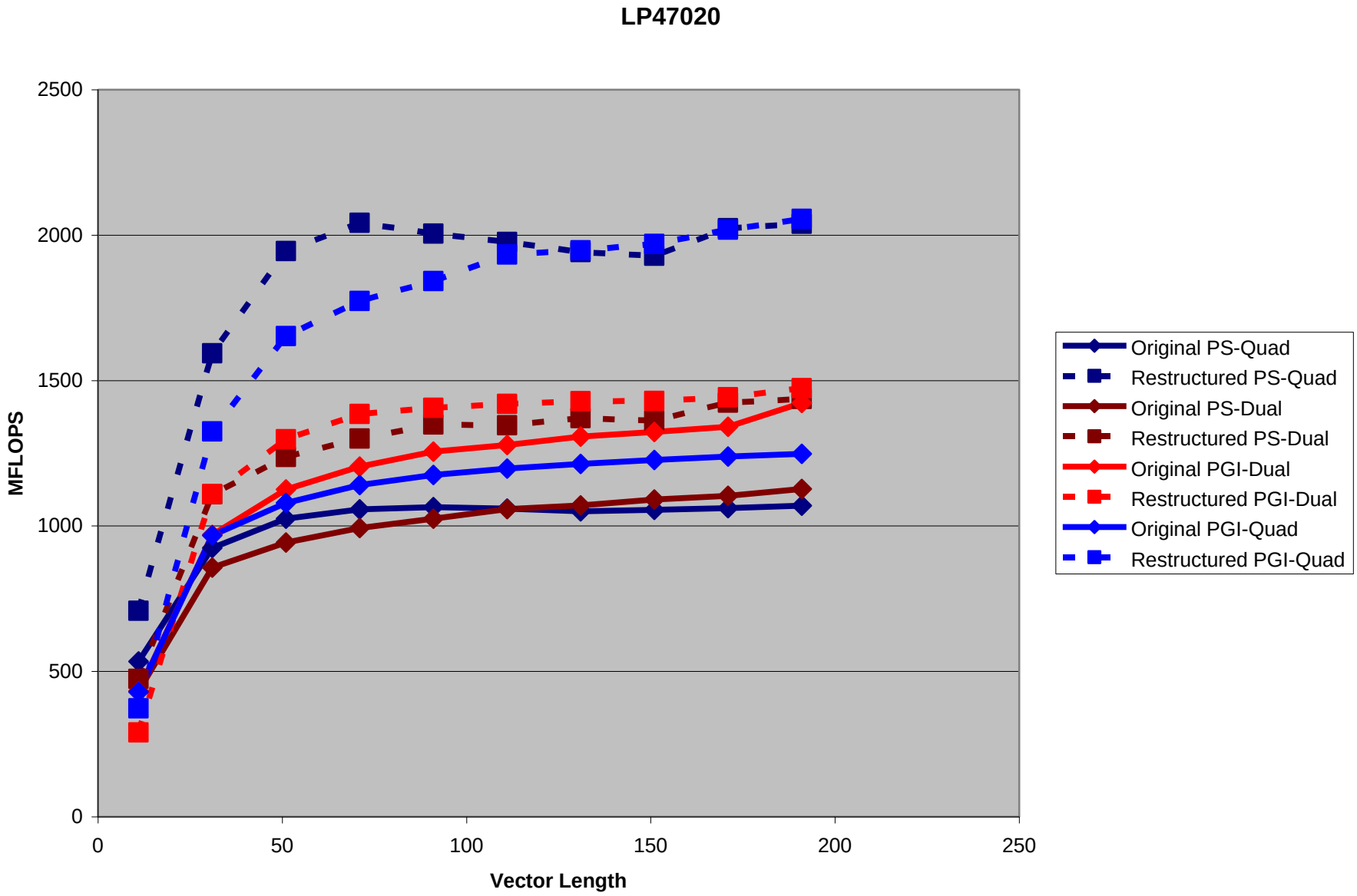
Re-Write

PGI

79, Generated vector sse code for inner loop
Generated 2 prefetch instructions for this loop
95, Generated vector sse code for inner loop
Generated 1 prefetch instructions for this loop

Pathscale

(lp47030.f:79) Non-contiguous array "B(_BLNK__.4000.0)" reference exists. Loop was not vectorized.
(lp47030.f:95) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.


LP47020
MFLOPS
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

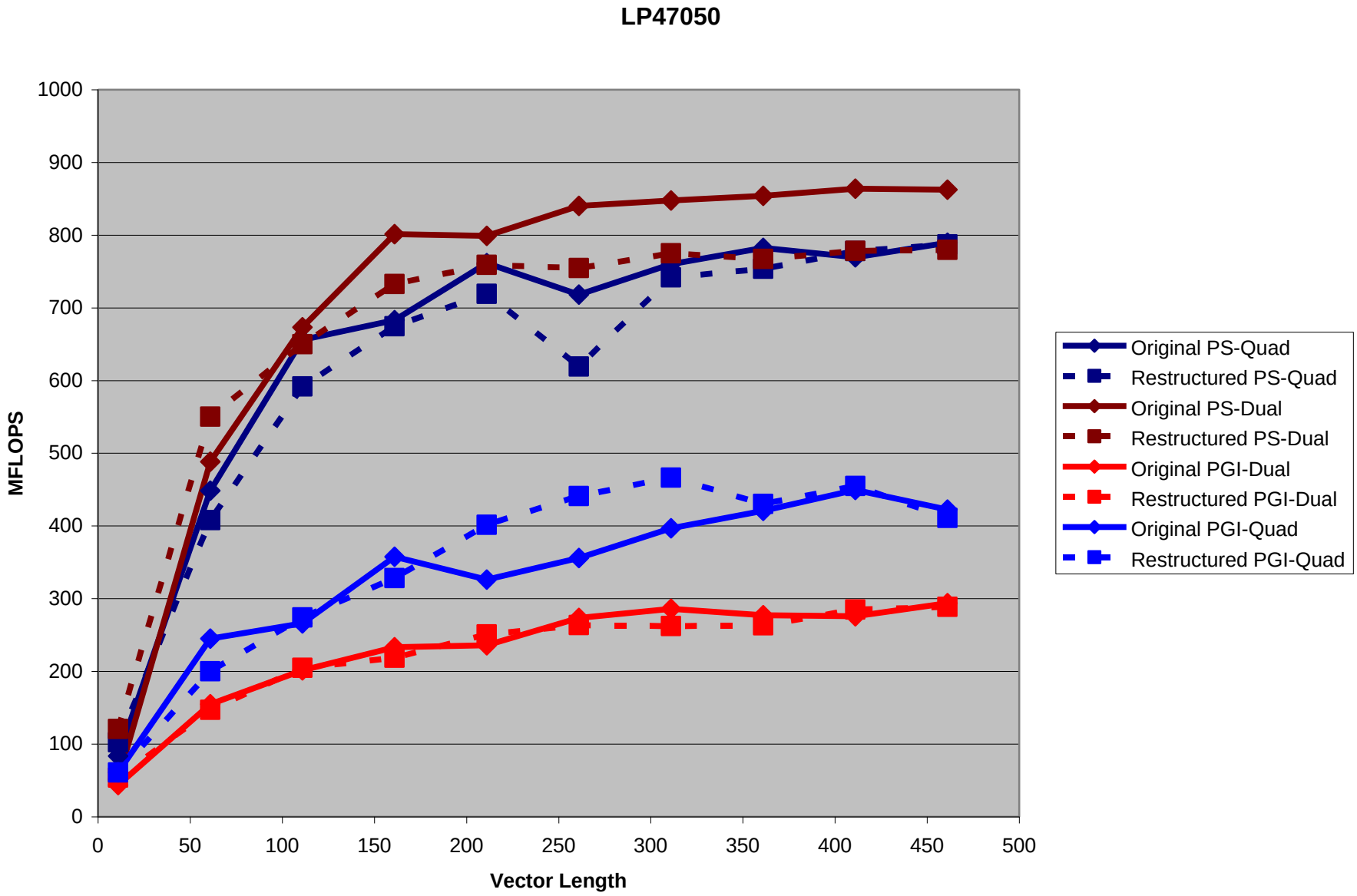
```
(   42) C      THE ORIGINAL
(   43)
(   44)       DO 47050 I = 1, N
(   45)        IIA = IA(I)
(   46)        GO TO (110, 120) IIA
(   47) 110    D(I) = B(I)
(   48)        A(I) = D(I) + 1.7
(   49)        GO TO 47050
(   50) 120    D(I) = C(I)
(   51)        A(I) = D(I) + 1.1
(   52) 47050 CONTINUE
(   53)
```

PGI
Nothing
Pathscale
Nothing

Restructured

```
(   71) C      THE RESTRUCTURED
(   72)
(   73)       DO 47051 I = 1, N
(   74)        IF(IA(I) .NE. 2) THEN
(   75)         D(I) = B(I)
(   76)         A(I) = D(I) + 1.7
(   77)        ELSE
(   78)         D(I) = C(I)
(   79)         A(I) = D(I) + 1.1
(   80)        ENDIF
(   81) 47051 CONTINUE
```

PGI
Nothing
Pathscale
(lp47050.f:73) Expression rooted at op "OPC_IF"(line 74) is not vectorizable.
Loop was not vectorized.


LP47050
MFLOPS
Vector Length
0
100
200
300
400
500
600
700
800
900
1000
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

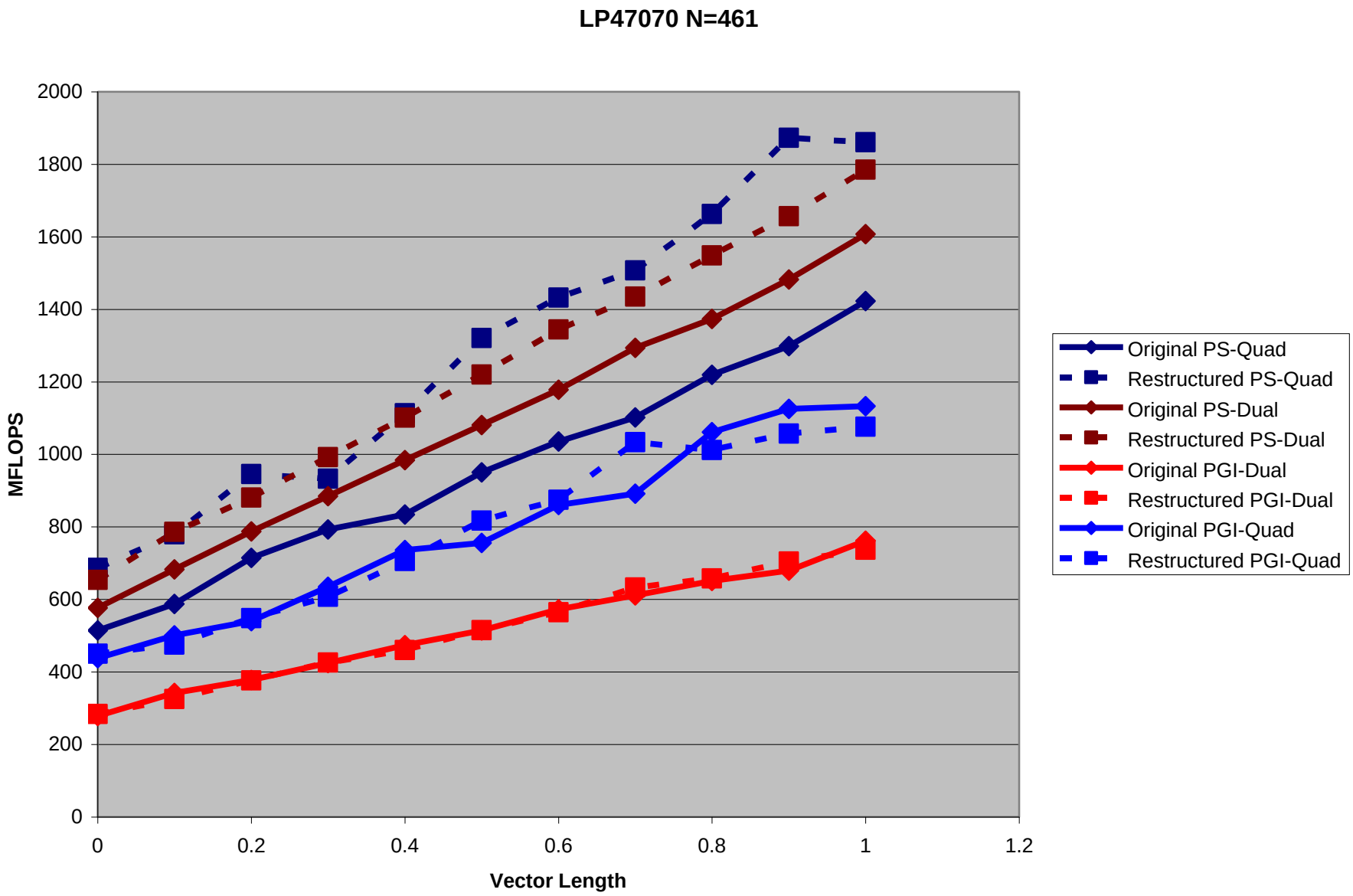
```
(   45)
(   46)       DO 47070 I = 1, N
(   47)        A(I) = B(I) * C(I)
(   48)        IF (A(I) .NE. 0.) GO TO 110
(   49)         C0   = B(I)**2 + C(I)**2
(   50)        A(I) = D(I) * E(I) + C0
(   51)         B(I) = 1.
(   52) 110    CONTINUE
(   53)       F(I) = A(I) + B(I)
(   54) 47070 CONTINUE
(   55)
```

PGI
Nothing
Pathscale
Nothing

Restructured

```
(   74) C      THE RESTRUCTURED
(   75)
(   76)       DO 47071 I = 1, N
(   77)        A(I) = B(I) * C(I)
(   78)        IF (A(I) .EQ. 0.) THEN
(   79)         A(I) = D(I) * E(I) + B(I)**2 + C(I)**2
(   80)         B(I) = 1.
(   81)        ENDIF
(   82)        F(I) = A(I) + B(I)
(   83) 47071 CONTINUE
(   84)
```

PGI
Nothing
Pathscale
(lp47070.f:76) Expression rooted at op "OPC_IF"(line 77) is not vectorizable.
Loop was not vectorized.


LP47070 N=461
MFLOPS
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
0.2
0.4
0.6
0.8
1
1.2
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   45)
(   46) C       THE ORIGINAL
(   47)
(   48)        DO 47101 I = 1, N
(   49)         U1 = X2(I)
(   50)
(   51)         DO 47100 LT = 1, NTAB
(   52)          IF (U1 .GT. X1(LT)) GO TO 47100
(   53)            IL = LT
(   54)            GO TO 121
(   55) 47100   CONTINUE
(   56)
(   57)         IL = NTAB - 1
(   58) 121     Y2(I) = Y1(IL) + ( Y1(IL+1) - Y1(IL) ) /
(   59)       *                  ( X1(IL+1) - X1(IL) ) *
(   60)       *                     ( X2(I) - X1(IL) )
(   61) 47101 CONTINUE
(   62)
```

PGI

51, Loop not vectorized: multiple exits

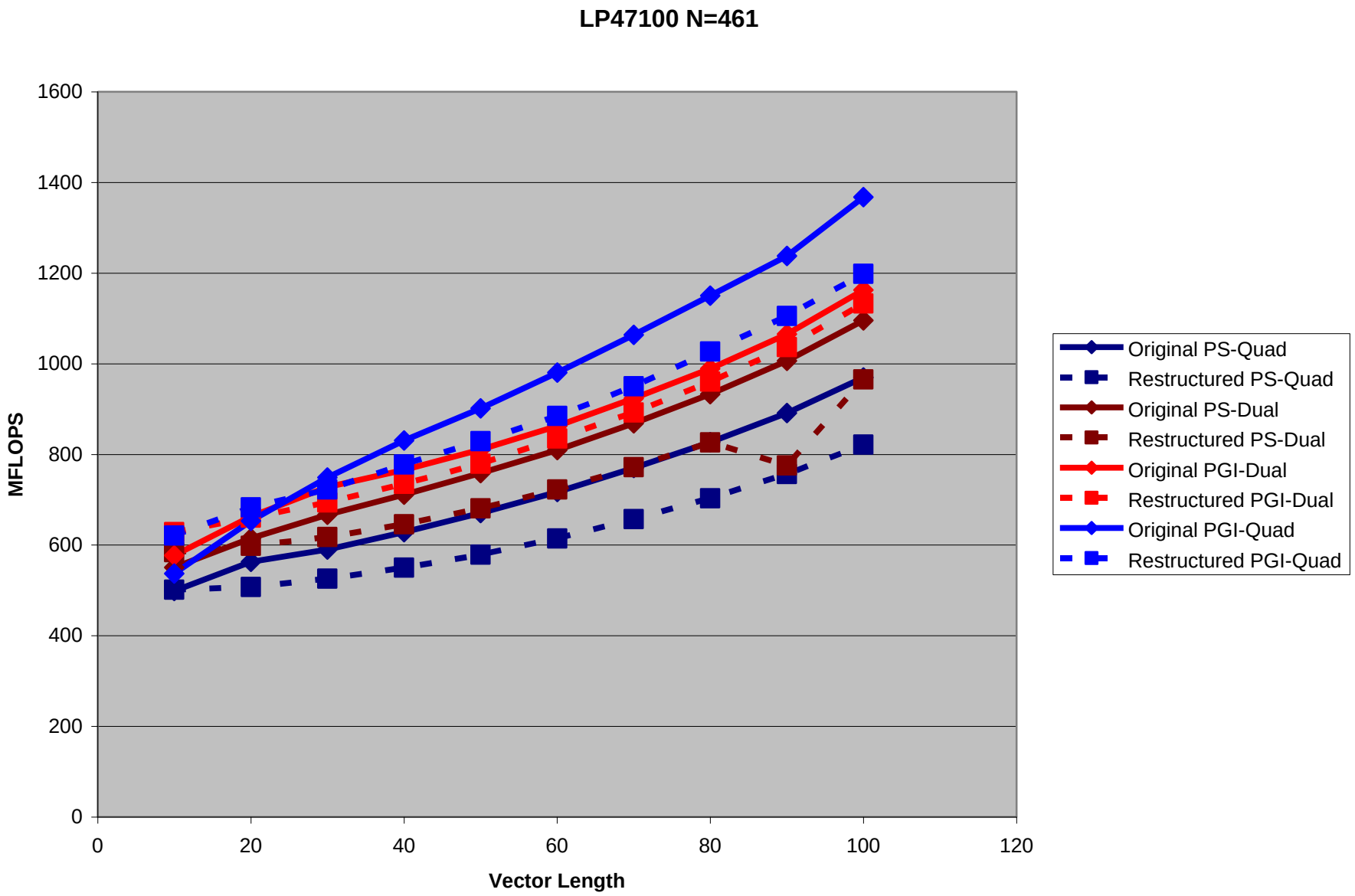
Pathscale

Nothing

Restructured

```
(   80) C      THE RESTRUCTURED
(   81)
(   82)       DO 47103 I = 1, N
(   83)        U1 = X2(I)
(   84)
(   85)        DO 47102 LT = 1, NTAB
(   86)         IF (U1 .GT. X1(LT)) GO TO 47102
(   87)          IV(I) = LT
(   88)          GO TO 47103
(   89) 47102  CONTINUE
(   90)
(   91)        IV(I) = NTAB - 1
(   92) 47103 CONTINUE
(   93)
(   94)       DO 47104 I  =  1, N
(   95)        Y2(I) = Y1(IV(I)) + ( Y1(IV(I)+1) - Y1(IV(I)) ) /
(   96)      *                     ( X1(IV(I)+1) - X1(IV(I)) ) *
(   97)      *                           ( X2(I) - X1(IV(I)) )
(   98) 47104 CONTINUE
(   99)
```

PGI
 85, Loop not vectorized: multiple exits
Pathscale
(lp47100.f:94) Non-contiguous array "Y1(_BLNK__.8808.0)" reference exists.
Loop was not vectorized.


LP47100 N=461
MFLOPS
1600
1400
1200
1000
800
600
400
200
0
0
20
40
60
80
100
120
Vector Length
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   42) C       THE ORIGINAL
(   43)
(   44)        I = 0
(   45) 47120 CONTINUE
(   46)         I = I + 1
(   47)         A(I) = B(I)**2 + .5 * C(I) * D(I) / E(I)
(   48)        IF (I .LT. N) GO TO 47120
(   49)
```

PGI
 Nothing
Pathscale
Nothing

Restructured

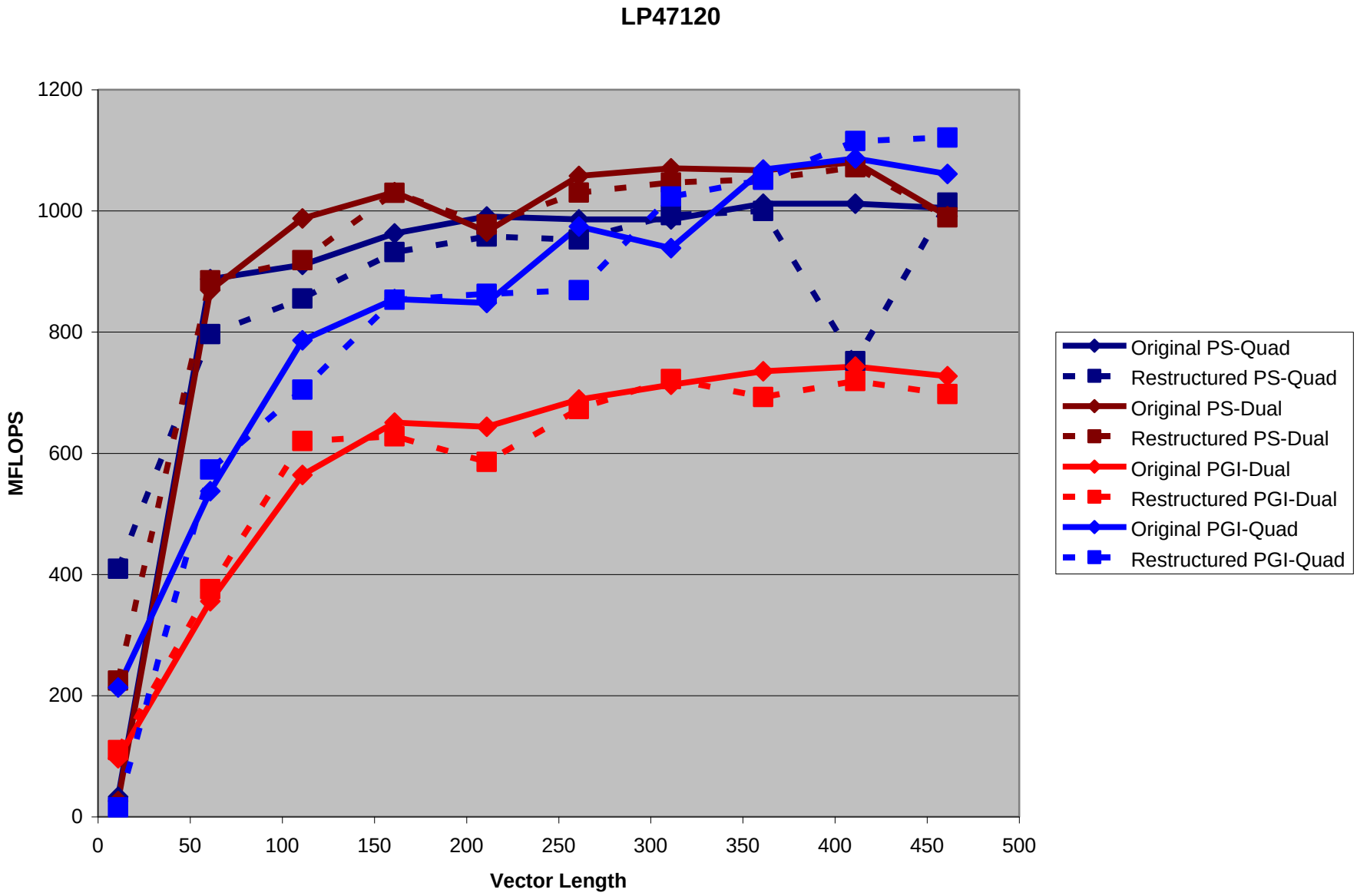
```
(    67) C       THE RESTRUCTURED
(    68)
(    69)        DO 47121 I = 1, N
(    70)         A(I) = B(I)**2 + .5 * C(I) * D(I) / E(I)
(    71) 47121 CONTINUE
(    72)
```

PGI

69, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop

Pathscale

(lp47120.f:69) Expression rooted at op "OPC_F8RECIP"(line 70) is not vectorizable. Loop was not vectorized.


LP47120
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   39) C      THE ORIGINAL
(   40)
(   41)       DO 48010 I = 1, N
(   42)        A(I) = B(I) * C(I)
(   43)        D(I) = FRED (A(I)**2 + 2.0)
(   44)        E(I) = D(I) / B(I) + A(I)
(   45) 48010 CONTINUE(   49)
```

PGI
 41, Loop not vectorized: contains call
Pathscale
Nothing

Restructured

```
(   65) C      THE RESTRUCTURED
(   66)
(   67)       DO 48011 I = 1,N
(   68)        A(I) = B(I) * C(I)
(   69)        D(I) = A(I)**2 + 2.0
(   70) 48011 CONTINUE
(   71)
(   72)       DO 48012 I = 1,N
(   73)        D(I) = FRED (D(I))
(   74) 48012 CONTINUE
(   75)
(   76)       DO 48013 I = 1,N
(   77)        E(I) = D(I) / B(I) + A(I)
(   78) 48013 CONTINUE
(   79)
```

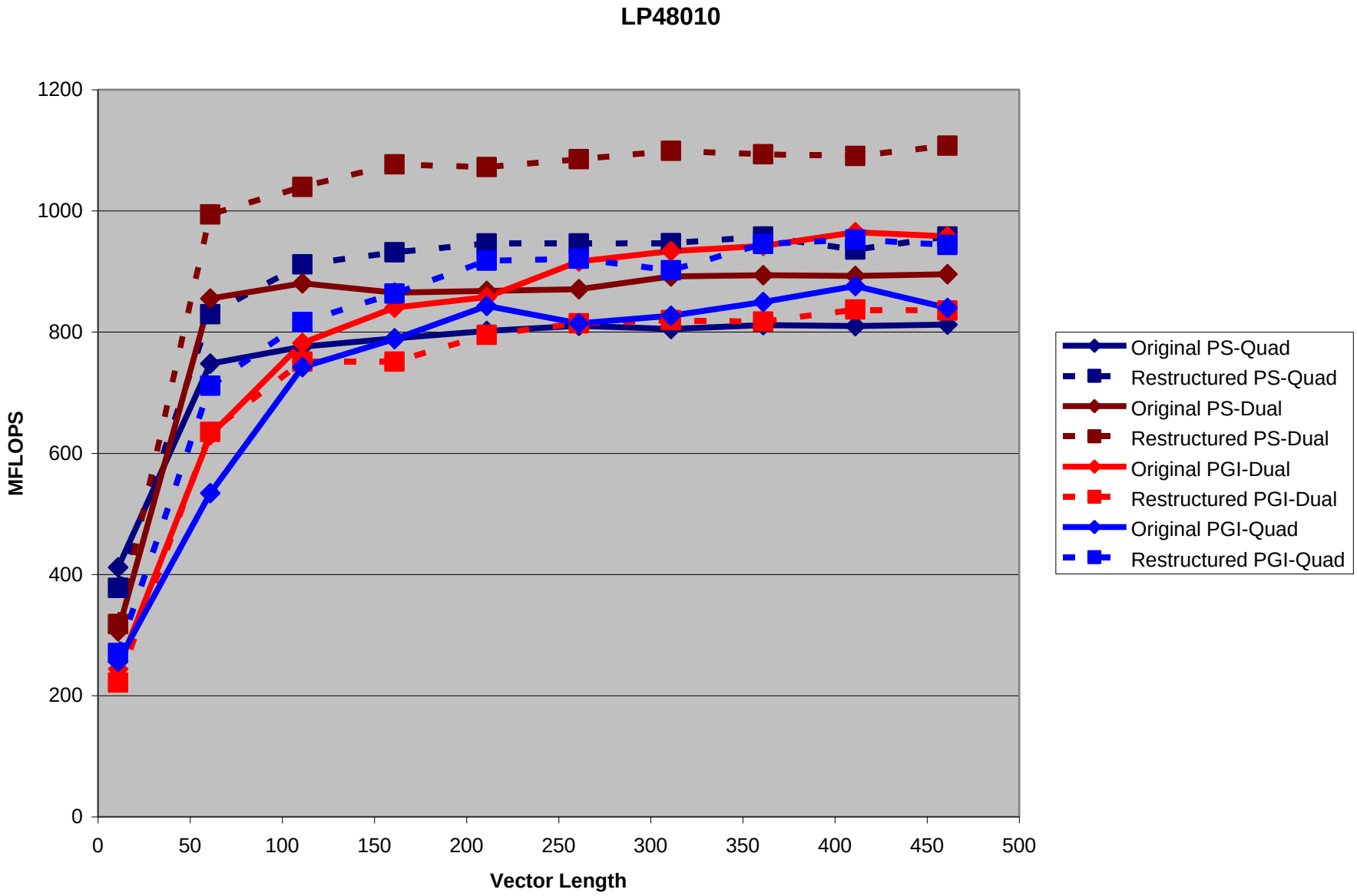
PGI

```
  67, Generated an alternate loop for the inner loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
    72, Loop not vectorized: contains call
    76, Generated an alternate loop for the inner loop
        Generated vector sse code for inner loop
        Generated 3 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 3 prefetch instructions for this loop
```

Pathscale

(lp48010.f:67) LOOP WAS VECTORIZED.

(lp48010.f:76) Expression rooted at op "OPC_F8RECIP"(line 77) is not vectorizable. Loop was not vectorized.


LP48010
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   39) C       THE ORIGINAL
(   40)
(   41)         DO 48020 I = 1, N
(   42)          A(I) = B(I) * FUNC (D(I)) + C(I)
(   43) 48020 CONTINUE
(   44)
```

PGI
 41, Loop not vectorized: contains call
Pathscale
Nothing

Restructured

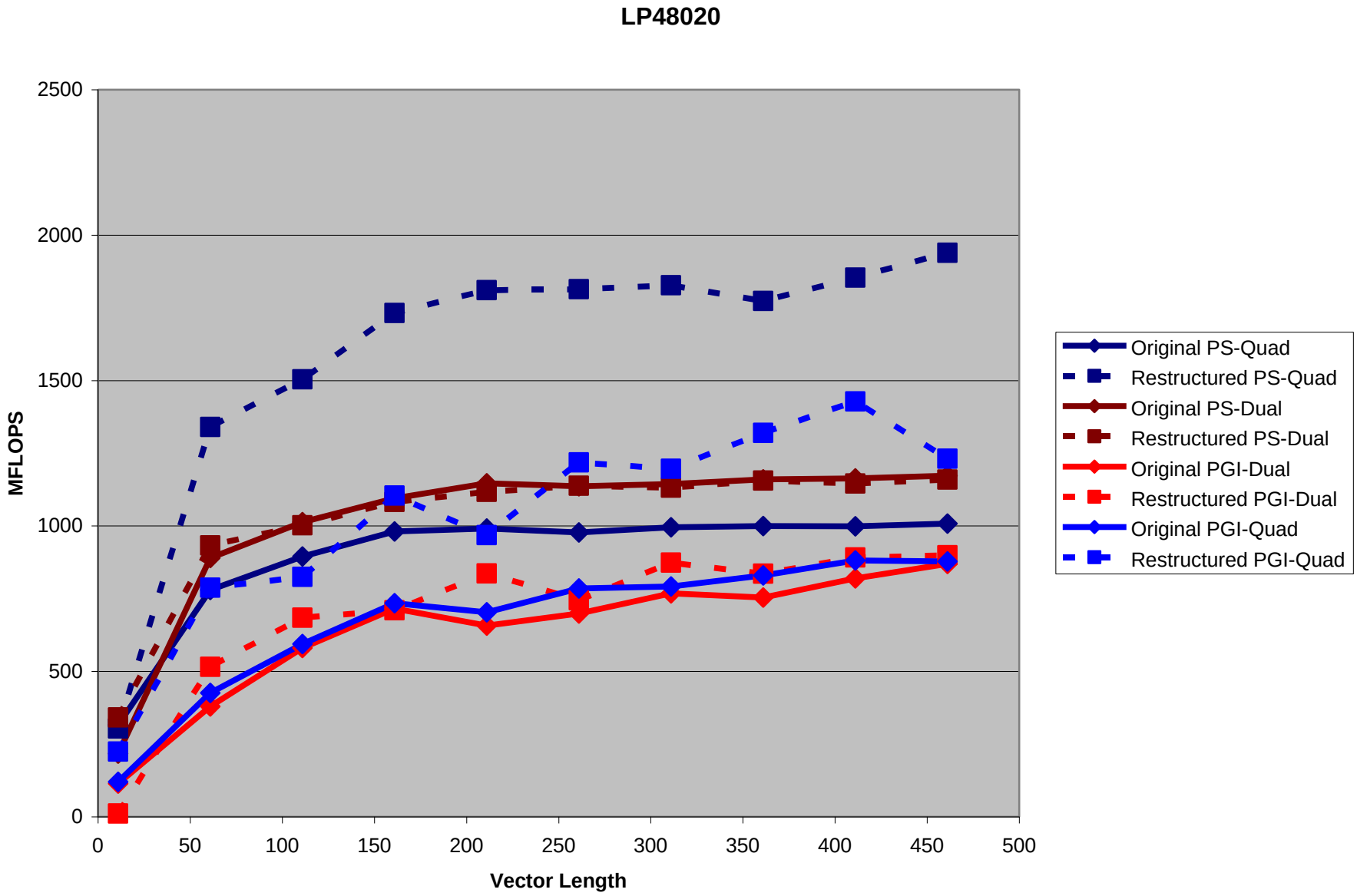
```
(  10)      FUNCX (X) = X**2 + 2.0 / X
(    62)
(    63)         DO 48021 I = 1, N
(    64)          A(I) = B(I) * FUNCX (D(I)) + C(I)
(    65)  48021 CONTINUE
(    66)
```

PGI

63, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 3 prefetch instructions for this loop

Pathscale

(lp48020.f:63) LOOP WAS VECTORIZED.


LP48020
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   42) C       THE ORIGINAL
(   43)
(   44)        DO 48060 I = 1, N
(   45)         AOLD = A(I)
(   46)         A(I) = UFUN (AOLD, B(I), SCA)
(   47)         C(I) = (A(I) + AOLD) * .5
(   48) 48060 CONTINUE
(   49)
```

PGI

41, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

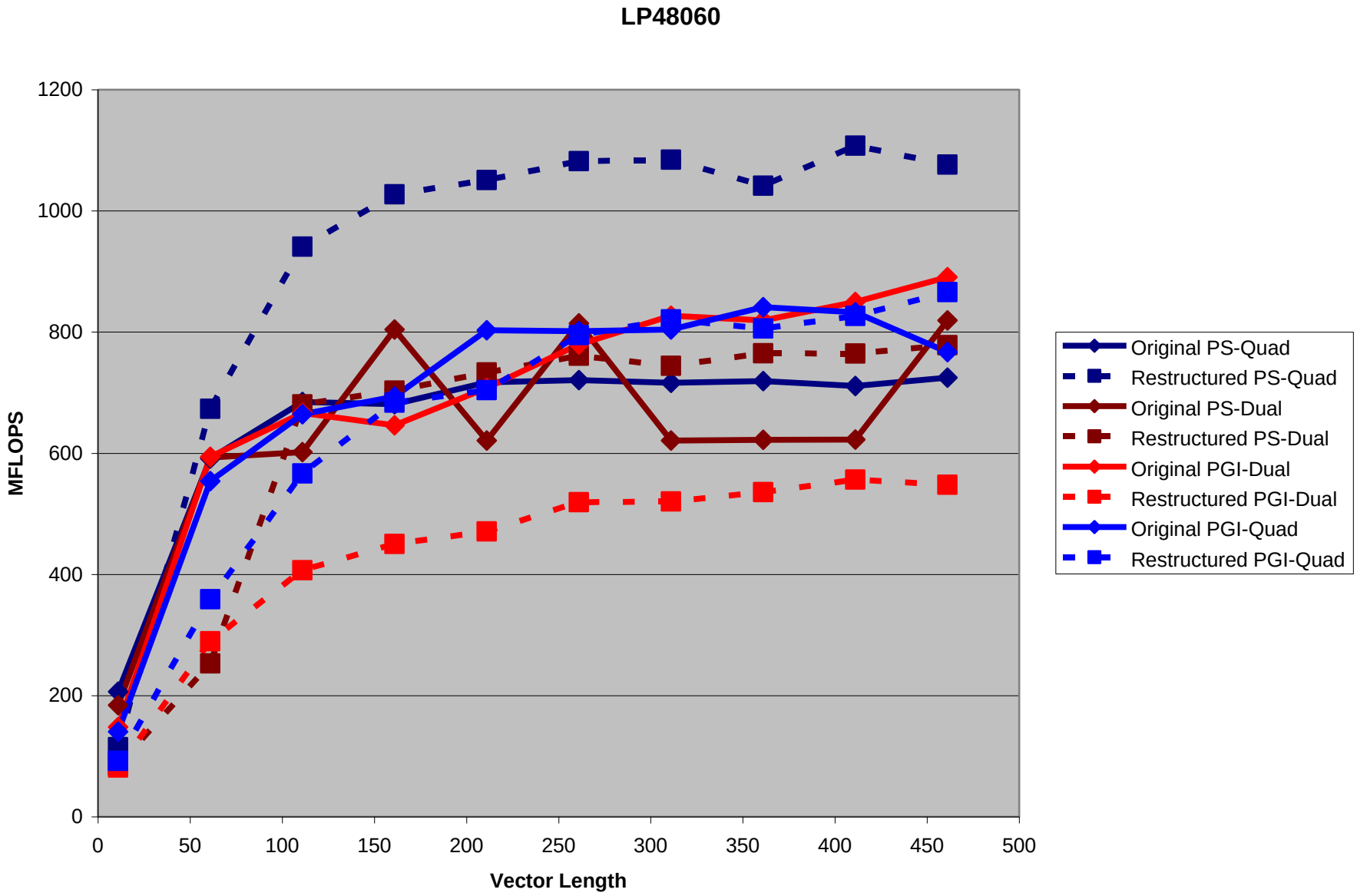
```
(   71) C      THE RESTRUCTURED
(   72)
(   73)       DO 48061 I = 1, N
(   74)        VAOLD(I) = A(I)
(   75) 48061 CONTINUE
(   76)
(   77)       CALL VUFUN (N, VAOLD, B, SCA, A)
(   78)
(   79)       DO 48062 I = 1, N
(   80)        C(I) = (A(I) + VAOLD(I)) * .5
(   81) 48062 CONTINUE
(   82)
```

PGI

```
  73, Memory copy idiom, loop replaced by memcopy call
    79, Generated an alternate loop for the inner loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
        Generated vector sse code for inner loop
        Generated 2 prefetch instructions for this loop
    91, Generated vector sse code for inner loop
```

Pathscale

```
(lp48060.f:73) LOOP WAS VECTORIZED.
(lp48060.f:79) LOOP WAS VECTORIZED.
(lp48060.f:91) LOOP WAS VECTORIZED.
```


LP48060
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   42) C      THE ORIGINAL
(   43)
(   44)       DO 48070 I = 1, N
(   45)        A(I) =  (B(I)**2 + C(I)**2)
(   46)        CT   = PI * A(I) + (A(I))**2
(   47)        CALL SSUB (A(I), CT, D(I), E(I))
(   48)        F(I) = (ABS (E(I)))
(   49) 48070 CONTINUE
(   50)
```

PGI

44, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

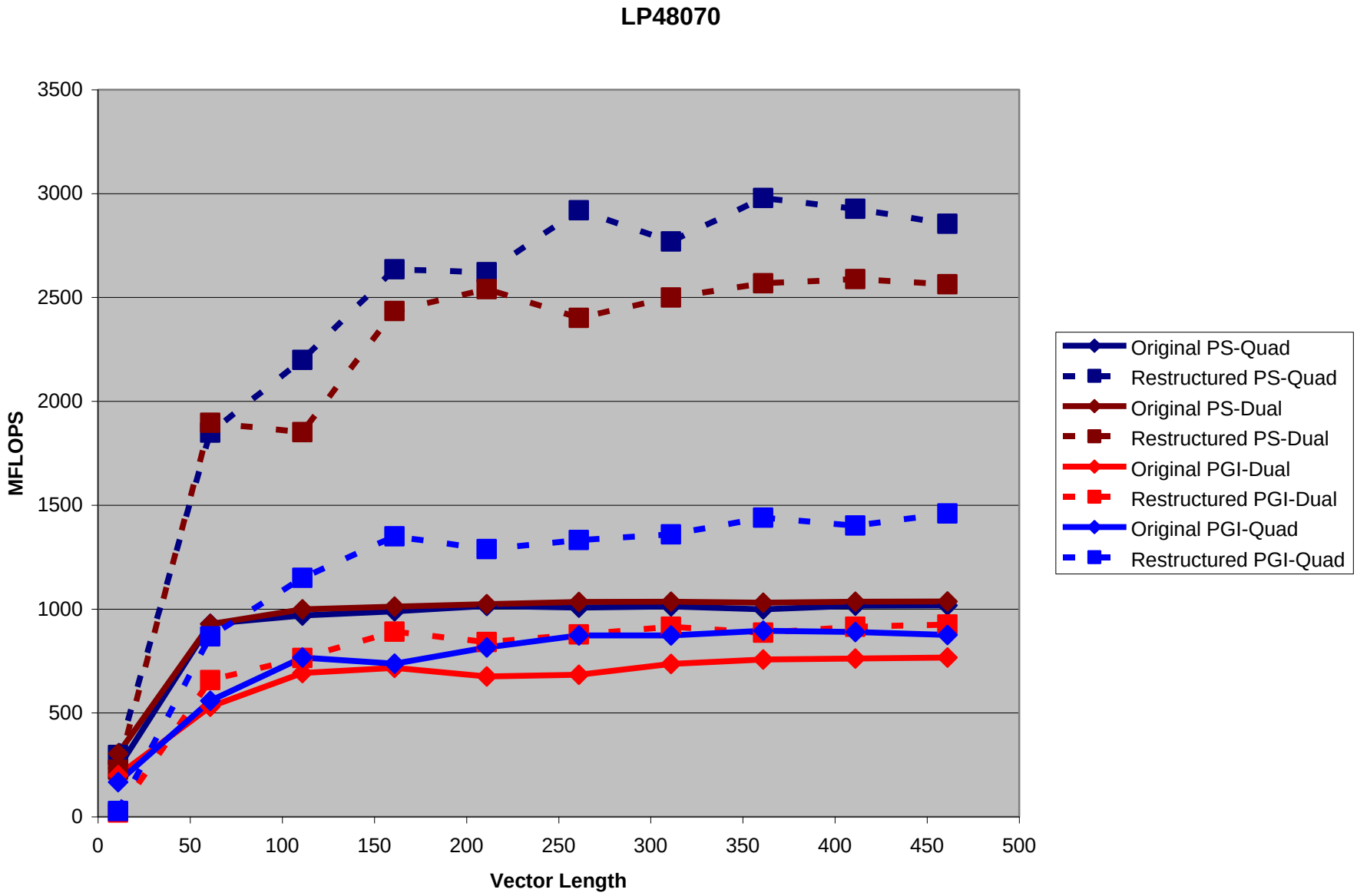
```
(   69) C      THE RESTRUCTURED
(   70)
(   71)       DO 48071 I = 1, N
(   72)        A(I) = (B(I)**2 + C(I)**2)
(   73)        CT   = PI * A(I) + (A(I))**2
(   74)        E(I) = A(I)**2 + (ABS (A(I) + CT)) * (CT * ABS (A(I) - CT))
(   75)        D(I) = A(I) + CT
(   76)        F(I) = (ABS (E(I)))
(   77) 48071 CONTINUE
(   78)
```

PGI

71, Generated an alternate loop for the inner loop
    Unrolled inner loop 4 times
    Used combined stores for 2 stores
    Generated 2 prefetch instructions for this loop
    Unrolled inner loop 4 times
    Used combined stores for 2 stores
    Generated 2 prefetch instructions for this loop

Pathscale

(lp48070.f:71) LOOP WAS VECTORIZED.


LP48070
MFLOPS
Vector Length
0
500
1000
1500
2000
2500
3000
3500
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(    41) C       THE ORIGINAL
(    42)
(    43)         DO 48080 i = 1 , n
(    44)         a(i)=sqrt(b(i)**2+c(i)**2)
(    45)         sca=a(i)**2+b(i)**2
(    46)         scalr=sca*2
(    47)         CALL sub2(sca)
(    48)         d(i)=sqrt(abs(a(i)+sca))
(    49) 48080 CONTINUE
(    50)
```

PGI

43, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

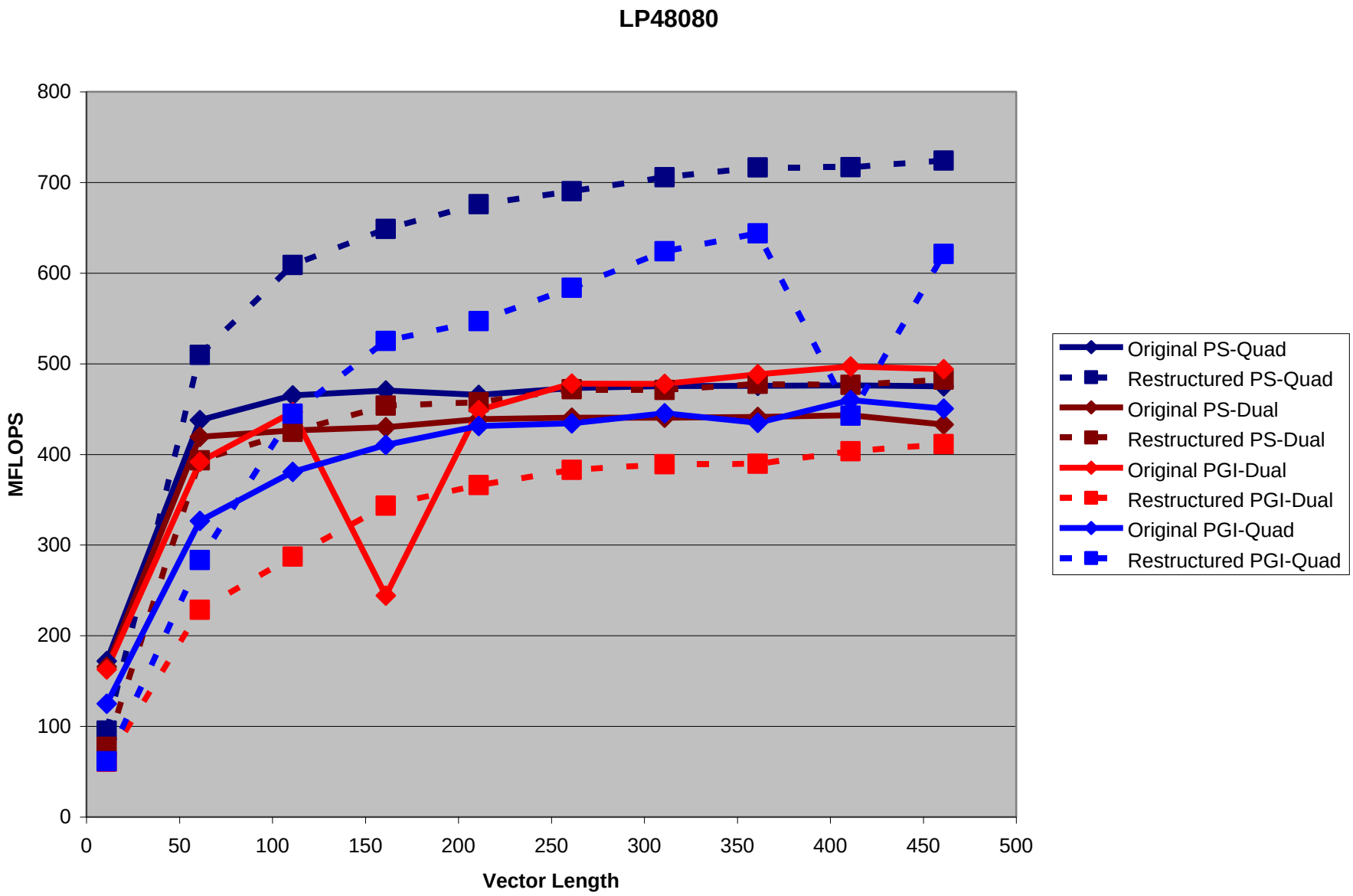
```
(   69) C      THE RESTRUCTURED
(   70)
(   71)        DO 48081 i = 1 , n
(   72)        a(i)=sqrt(b(i)**2+c(i)**2)
(   73) 48081 CONTINUE
(   74)
(   75)        CALL vsub1(n,a,b,vsca,vscalr)
(   76)
(   77)        CALL vsub2(n,vsca,vscalr)
(   78)
(   79)        DO 48082 i = 1 , n
(   80)        d(i)=sqrt(abs(a(i)+vsca(i)))
(   81) 48082 CONTINUE
(   82)
```

PGI

```
  71, Generated an alternate loop for the inner loop
      Generated vector sse code for inner loop
      Generated 2 prefetch instructions for this loop
      Generated vector sse code for inner loop
      Generated 2 prefetch instructions for this loop
  79, Generated an alternate loop for the inner loop
      Generated vector sse code for inner loop
      Generated 2 prefetch instructions for this loop
      Generated vector sse code for inner loop
      Generated 2 prefetch instructions for this loop
```

Pathscale

```
(lp48080.f:71) LOOP WAS VECTORIZED.
(lp48080.f:79) LOOP WAS VECTORIZED.
```


LP48080
MFLOPS
Vector Length
0
100
200
300
400
500
600
700
800
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

Original

```
(   43) C      THE ORIGINAL
(   44)
(   45)       ET = 0.0
(   46)       DO 48090 I = 1, N
(   47)        B(I) = SQRT (F(I)**2 + E(I)**2) + ET
(   48)        CALL SSSUB (B(I), ET, C(I), D(I), PI)
(   49)        A(I) = SQRT (ABS (D(I) ) )
(   50) 48090 CONTINUE
(   51)
```

PGI
 46, Loop not vectorized: contains call
Pathscale
Nothing

Restructured

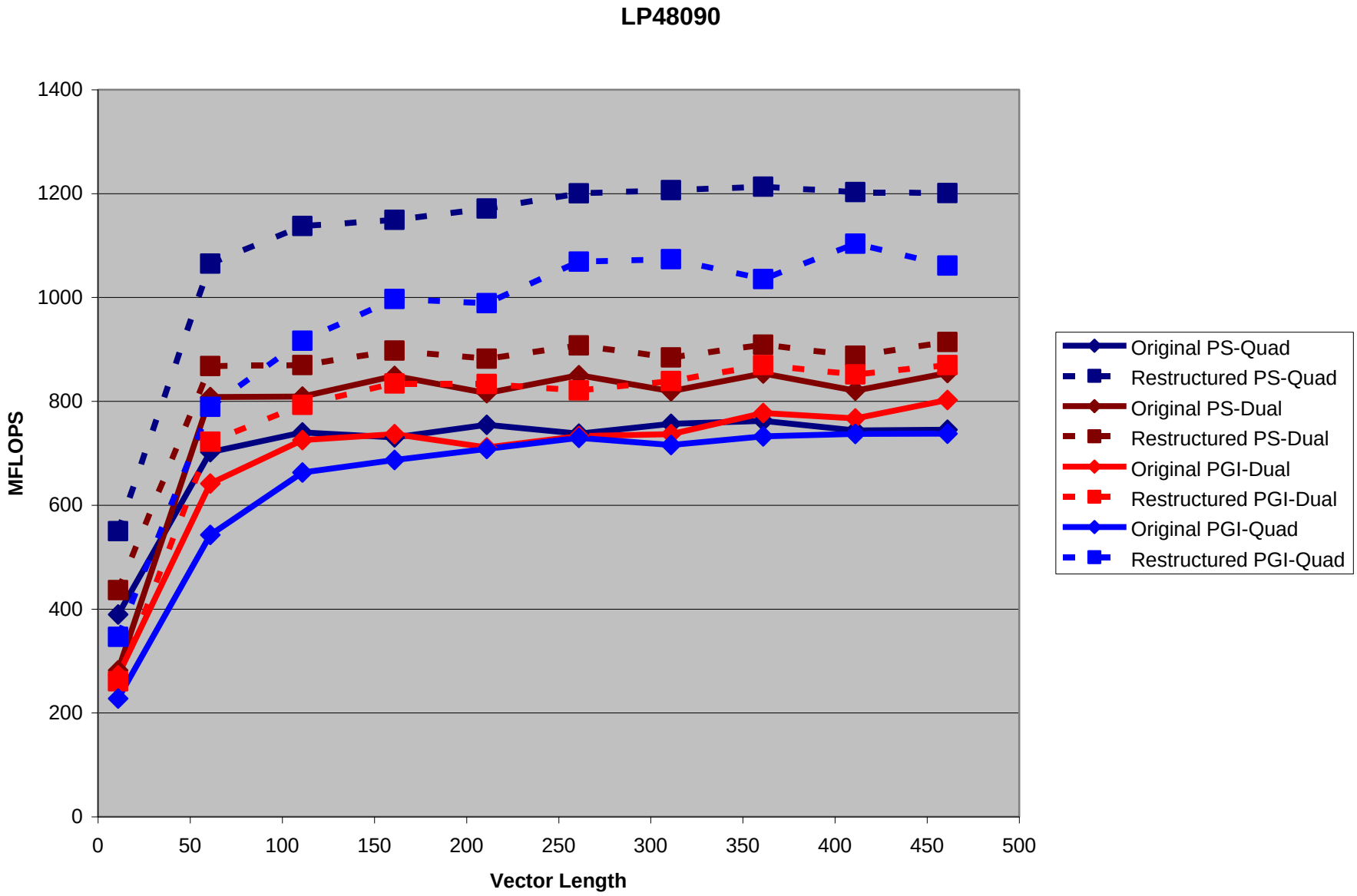
```
(   70) C      THE RESTRUCTURED
(   71)
(   72)       VET(1)=0.0
(   73)       DO 48091 I = 1, N
(   74)        VET(I+1) = PI * C(I) + C(I)
(   75)        B(I) = SQRT (F(I)**2 + E(I)**2) + VET(I)
(   76)        D(I) = B(I)**2 + C(I)**2 * SQRT (ABS (B(I) + C(I) ) )
(   77)        D(I) = VET(I+1) + D(I)
(   78)        A(I) = SQRT (ABS (D(I) ) )
(   79) 48091 CONTINUE
(   80)
```

PGI

73, Generated an alternate loop for the inner loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop
Generated vector sse code for inner loop
Generated 4 prefetch instructions for this loop

Pathscale

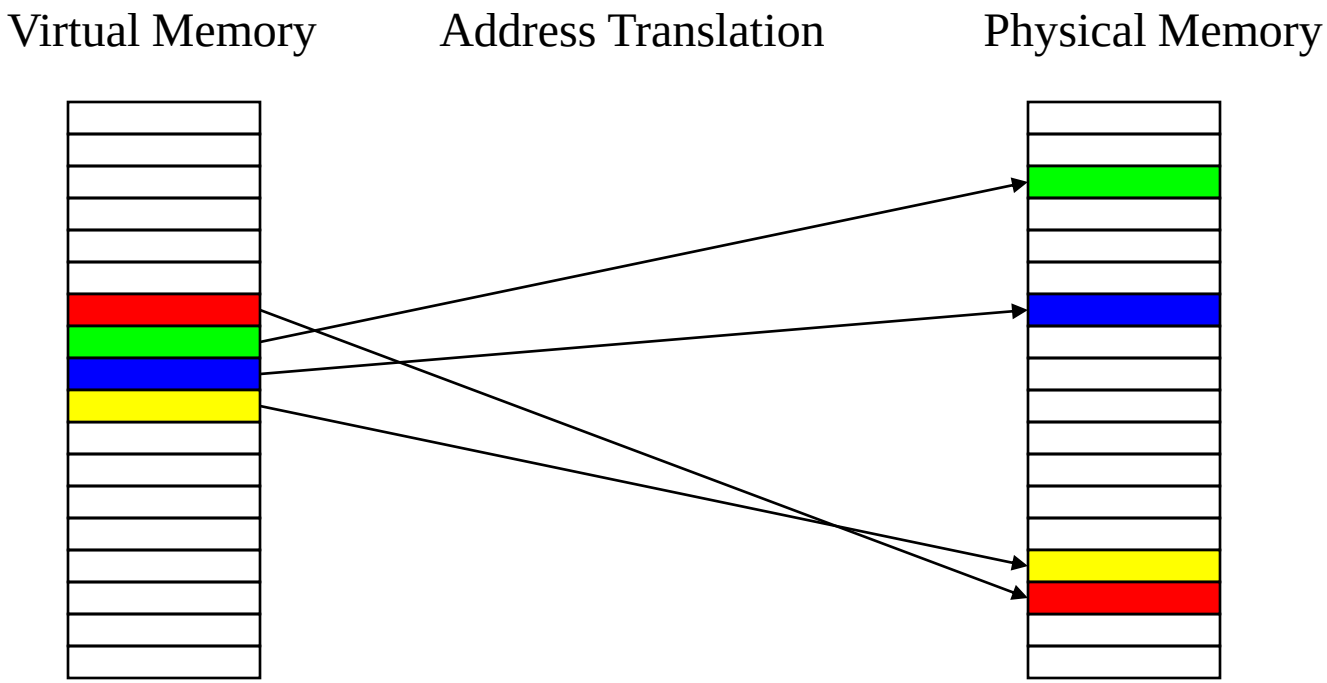
(lp48090.f:73) LOOP WAS VECTORIZED.


LP48090
MFLOPS
Vector Length
0
200
400
600
800
1000
1200
1400
0
50
100
150
200
250
300
350
400
450
500
Original PS-Quad
Restructured PS-Quad
Original PS-Dual
Restructured PS-Dual
Original PGI-Dual
Restructured PGI-Dual
Original PGI-Quad
Restructured PGI-Quad

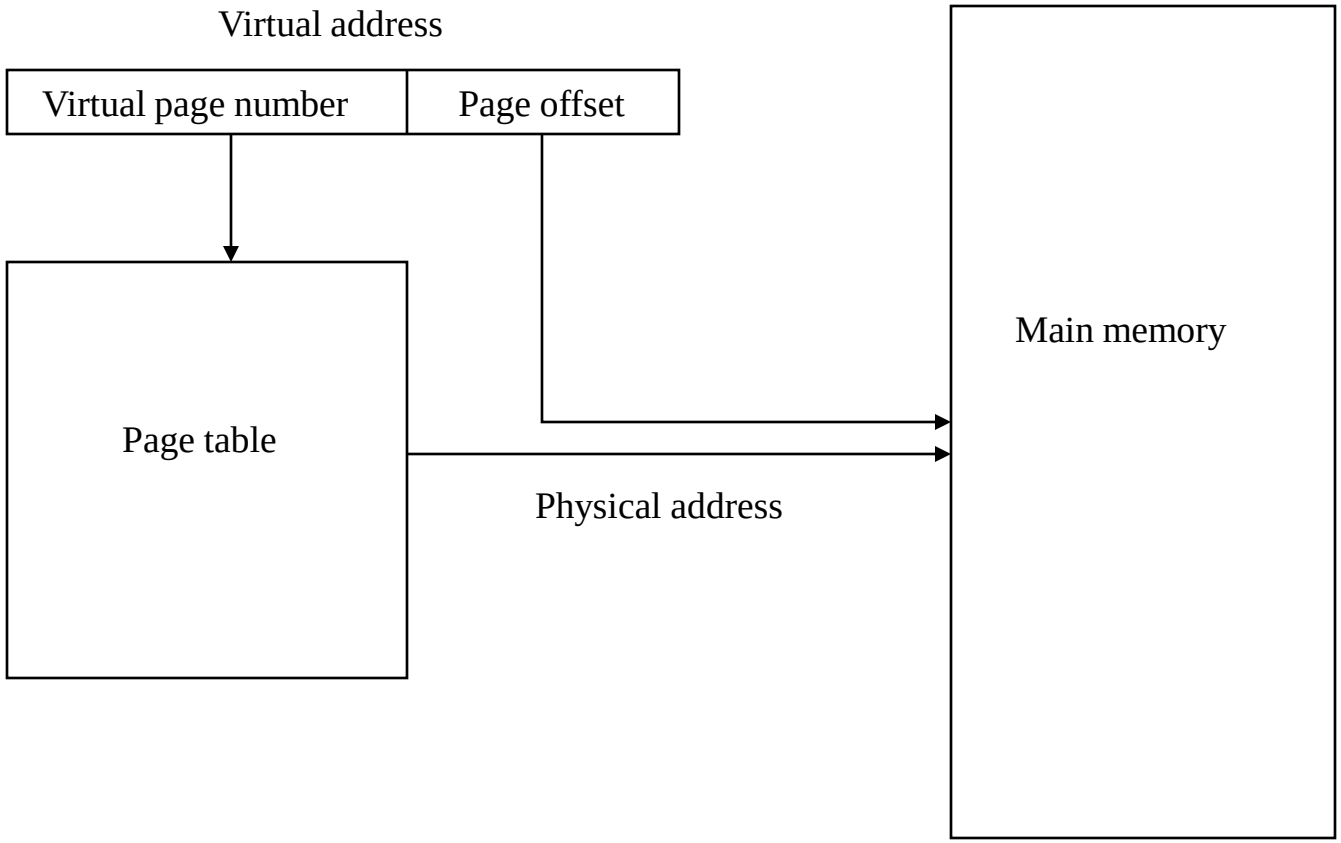
Background: Virtual Memory

- Modern programs operate in “virtual memory”
  - Each program thinks it has all of memory to itself
  - Fixed sized blocks (“pages”) vs variable sized blocks (“segments”)
- Virtual Memory benefits
  - Allow a program that is larger than physical memory to run
    - Programmer does not have to manually create overlays
  - Allow many programs to share limited physical memory
- Virtual Memory problems
  - Each virtual memory reference must be translated into a physical memory reference

Virtual Memory vs Physical Memory



Address Translation



Source: Computer Architecture A Quantitative Approach, by John L. Hennessy and David A. Patterson

Translation Speed

- Translation page table is stored in main memory
  - Each memory access logically takes twice as long – once to find the physical address, once to get the actual data
- Use a hardware cache of least recently used addresses
  - Called a Translation Lookaside Buffer or TLB

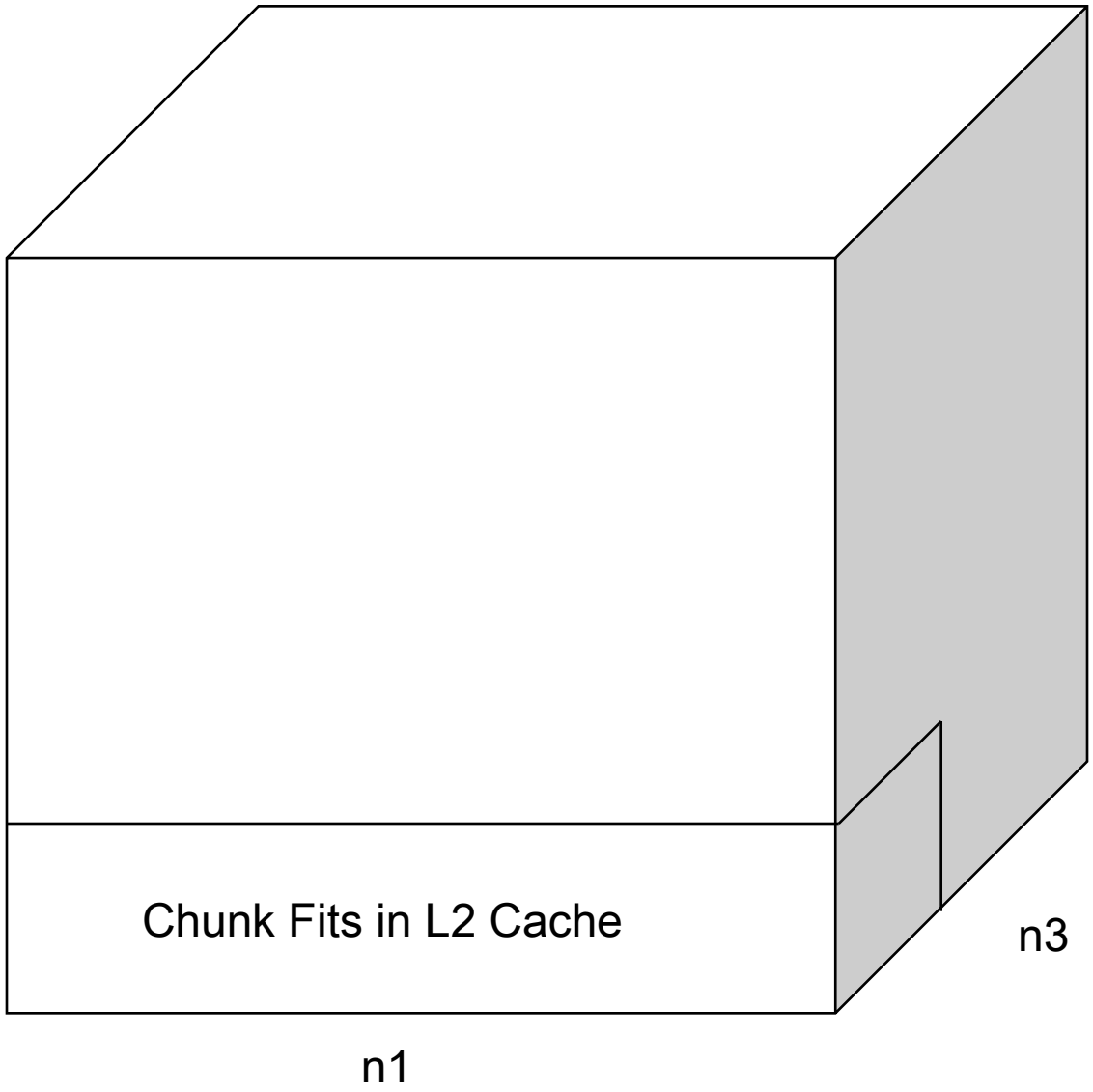
Performance Problem: TLB Refills

- AMD dual core opteron: 512 data TLB entries
- Covers 2MB of physical memory
  - OK if program fits (unlikely)
  - Large programs accessing data from all over their virtual memory range can trigger excessive TLB misses (“thrash”)
- One solution: huge pages

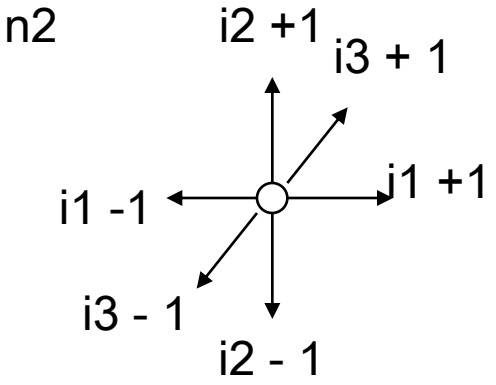
NPB MG routine RESID

```
      do i3=2,n3-1
           do i2=2,n2-1
              do i1=1,n1
                 u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
     >                   + u(i1,i2,i3-1) + u(i1,i2,i3+1)
                 u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
     >                   + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
              enddo
              do i1=2,n1-1
                 r(i1,i2,i3) = v(i1,i2,i3)
     >                       - a(0) * u(i1,i2,i3)
     >                       - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
     >                       - a(3) * ( u2(i1-1) + u2(i1+1) )
              enddo
           enddo
      enddo
```

```
==========================================================================
USER / resid_
--------------------------------------------------------------------------
  Time%                                              42.4%
  Time                                           12.397761
  Imb.Time                                        0.000370
  Imb.Time%                                           0.0%
  Calls                                                340
  PAPI_L1_DCA                 2719.188M/sec     33711498004 ops
  DC_L2_REFILL_MOESI            79.644M/sec       987402929 ops
  DC_SYS_REFILL_MOESI            4.059M/sec        50318116 ops
  BU_L2_REQ_DC                 129.172M/sec      1601429574 req
  User time                     12.398 secs      32233848320 cycles
  Utilization rate                                    100.0%
  L1 Data cache misses          83.703M/sec      1037721045 misses
  LD & ST per D1 miss                                 32.49 ops/miss
  D1 cache hit ratio                                  96.9%
  LD & ST per D2 miss                                669.97 ops/miss
  D2 cache hit ratio                                  96.9%
  L2 cache hit ratio                                  95.2%
  Memory to D1 refill            4.059M/sec        50318116 lines
  Memory to D1 bandwidth       247.723MB/sec      3220359424 bytes
  L2 to Dcache bandwidth      4861.112MB/sec     63193787456 bytes
--------------------------------------------------------------------------
```



Entire Cube does not fit in L2 Cache
256*256*256*3 arrays
= 402 MBytes



Take data in chunks that Fit in L2 Cache
256*16*32*3 arrays
= 1 MBytes

Tiling for better Cache utilization

```
do i3block=2,n3-1,BLOCK3
      do i2block=2,n2-1,BLOCK2
      do i3=i3block,min(n3-1,i3block+BLOCK3-1)
         do i2=i2block,min(n2-1,i2block+BLOCK2-1)
            do i1=1, n1
               u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
     >                + u(i1,i2,i3-1) + u(i1,i2,i3+1)
               u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
     >                + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
            enddo
            do i1=1, n1
               r(i1,i2,i3) = v(i1,i2,i3)
     >                     - a(0) * u(i1,i2,i3)
     >                     - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
     >                     - a(3) * ( u2(i1-1) + u2(i1+1) )
            enddo
         enddo
       enddo
       enddo
       enddo
```

```
==========================================================================
USER / resid_
--------------------------------------------------------------------------
  Time%                                          36.3%
  Time                                       8.753226
  Imb.Time                                   0.000596
  Imb.Time%                                      0.0%
  Calls                                           340
  PAPI_L1_DCA                3861.533M/sec   33800955933 ops
  DC_L2_REFILL_MOESI          116.399M/sec    1018867620 ops
  DC_SYS_REFILL_MOESI           2.755M/sec      24114222 ops
  BU_L2_REQ_DC                161.490M/sec    1413560527 req
  User time                     8.753 secs   22758444048 cycles
  Utilization rate                               100.0%
  L1 Data cache misses        119.154M/sec    1042981842 misses
  LD & ST per D1 miss                             32.41 ops/miss
  D1 cache hit ratio                              96.9%
  LD & ST per D2 miss                           1401.70 ops/miss
  D2 cache hit ratio                              98.3%
  L2 cache hit ratio                              97.7%
  Memory to D1 refill           2.755M/sec      24114222 lines
  Memory to D1 bandwidth      168.145MB/sec    1543310208 bytes
  L2 to Dcache bandwidth     7104.420MB/sec   65207527680 bytes
```

```
 do i3block=2,n3-1,BLOCK3
 do i2block=2,n2-1,BLOCK2
 do i3=i3block,min(n3-1,i3block+BLOCK3-1)
    do i2=i2block,min(n2-1,i2block+BLOCK2-1)
       do i1=1,n1
          u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
>                + u(i1,i2,i3-1) + u(i1,i2,i3+1)
          u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
>                + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
       enddo
       do i1=2,n1-1
          r(i1,i2,i3) = v(i1,i2,i3)
>                     - a(0) * u(i1,i2,i3)
>                     - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
>                     - a(3) * ( u2(i1-1) + u2(i1+1) )
       enddo
    enddo
 enddo
 enddo
 enddo
```

```
do i3block=2,n3-1,BLOCK3
 do i2block=2,n2-1,BLOCK2
 do i3=i3block,min(n3-1,i3block+BLOCK3-1)
    do i2=i2block,min(n2-1,i2block+BLOCK2-1)
       do i1=2,n1-1
          u21 = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
>               + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
          u21p1 = u(i1+1,i2-1,i3-1) + u(i1+1,i2+1,i3-1)
>               + u(i1+1,i2-1,i3+1) + u(i1+1,i2+1,i3+1)
          u21m1 = u(i1-1,i2-1,i3-1) + u(i1-1,i2+1,i3-1)
>               + u(i1-1,i2-1,i3+1) + u(i1-1,i2+1,i3+1)
          u11p1 = u(i1+1,i2-1,i3) + u(i1+1,i2+1,i3)
>               + u(i1+1,i2,i3-1) + u(i1+1,i2,i3+1)
          u11m1 = u(i1-1,i2-1,i3) + u(i1-1,i2+1,i3)
>               + u(i1-1,i2,i3-1) + u(i1-1,i2,i3+1)
          r(i1,i2,i3) = v(i1,i2,i3)
>                     - a(0) * u(i1,i2,i3)
>                     - a(2) * ( u21 + u11m1 + u11p1 )
>                     - a(3) * ( u21m1 + u21p1 )
       enddo
    enddo
 enddo
 enddo
 enddo
```

```
USER / resid_
------------------------------------------------------------------------
  Time%                                              37.7%
  Time                                            9.132935
  Imb.Time                                        0.003440
  Imb.Time%                                           0.1%
  Calls                                                340
  PAPI_TLB_DM                    0.139M/sec        1270096 misses
  PAPI_L1_DCA                 3694.219M/sec    33739238309 ops
  PAPI_FP_OPS                 2601.948M/sec    23763548027 ops
  DC_MISS                      111.833M/sec     1021371774 ops
  User time                      9.133 secs    23745753175 cycles
  Utilization rate                                  100.0%
  HW FP Ops / Cycles                                  1.00 ops/cycle
  HW FP Ops / User time       2601.948M/sec    23763548027 ops
25.0%peak
  HW FP Ops / WCT             2601.948M/sec
  Computation intensity                               0.70 ops/ref
  LD & ST per TLB miss                            26564.32 ops/miss
  LD & ST per D1 miss                                33.03 ops/miss
  D1 cache hit ratio                                  97.0%
  % TLB misses / cycle                                 0.0%
```

```
USER / resid_
------------------------------------------------------------------------
  Time%                                              39.6%
  Time                                           9.752716
  Imb.Time                                       0.002081
  Imb.Time%                                          0.0%
  Calls                                               340
  PAPI_TLB_DM                      0.115M/sec       1119418 misses
  PAPI_L1_DCA                   2792.319M/sec   27232706384 ops
  PAPI_FP_OPS                   3488.881M/sec   34026076279 ops
  DC_MISS                        104.718M/sec    1021283533 ops
  User time                        9.753 secs   25357072370 cycles
  Utilization rate                                   100.0%
  HW FP Ops / Cycles                                   1.34 ops/cycle
  HW FP Ops / User time         3488.881M/sec   34026076279 ops      33.5%peak
  HW FP Ops / WCT               3488.881M/sec
  Computation intensity                                1.25 ops/ref
  LD & ST per TLB miss                             24327.56 ops/miss
  LD & ST per D1 miss                                 26.67 ops/miss
  D1 cache hit ratio                                  96.2%
  % TLB misses / cycle                                 0.0%
```

```
USER / resid_
------------------------------------------------------------------------
  Time%                                              38.3%
  Time                                           9.162149
  Imb.Time                                       0.006363
  Imb.Time%                                          0.1%
  Calls                                              340
  PAPI_L1_DCA                  3682.405M/sec   33739250204 ops
  DC_L2_REFILL_MOESI            111.475M/sec    1021369289 ops
  DC_SYS_REFILL_MOESI             2.964M/sec      27157915 ops
  BU_L2_REQ_DC                  157.164M/sec    1439982850 req
  User time                       9.162 secs   23821945786 cycles
  Utilization rate                                 100.0%
  L1 Data cache misses          114.439M/sec    1048527204 misses
  LD & ST per D1 miss                               32.18 ops/miss
  D1 cache hit ratio                                96.9%
  LD & ST per D2 miss                             1242.34 ops/miss
  D2 cache hit ratio                                98.1%
  L2 cache hit ratio                                97.4%
  Memory to D1 refill             2.964M/sec      27157915 lines
  Memory to D1 bandwidth        180.914MB/sec    1738106560 bytes
  L2 to Dcache bandwidth       6803.916MB/sec   65367634496 bytes
```

```
USER / resid_
------------------------------------------------------------------------
  Time%                                              39.4%
  Time                                            9.699533
  Imb.Time                                        0.003564
  Imb.Time%                                           0.1%
  Calls                                                340
  PAPI_L1_DCA                  2807.643M/sec    27232738768 ops
  DC_L2_REFILL_MOESI            105.292M/sec     1021281565 ops
  DC_SYS_REFILL_MOESI             2.366M/sec       22945693 ops
  BU_L2_REQ_DC                  114.970M/sec     1115152062 req
  User time                       9.700 secs    25218702347 cycles
  Utilization rate                                   100.0%
  L1 Data cache misses          107.658M/sec     1044227258 misses
  LD & ST per D1 miss                                 26.08 ops/miss
  D1 cache hit ratio                                  96.2%
  LD & ST per D2 miss                               1186.83 ops/miss
  D2 cache hit ratio                                  97.9%
  L2 cache hit ratio                                  97.8%
  Memory to D1 refill             2.366M/sec       22945693 lines
  Memory to D1 bandwidth        144.388MB/sec     1468524352 bytes
  L2 to Dcache bandwidth       6426.524MB/sec    65362020160 bytes
```

Sparse CSR MV

Unroll q loop x times

```
do q = 1, n_rhs

    next_row_begin = row_start (1)

    do i = 1, n_rows

        row_begin      = next_row_begin
        next_row_begin = row_start (i +1)
        ip             = 0.0_wp

        do k = row_begin, next_row_begin - 1
            ip   = ip  +  values (k) * x (col_index (k), q)
        end do

        y (i, q) = ip

    end do
end do
```

Should Scream on Granite!

Prefetch x cachelines of values and y cachelines of col_index, z iterations ahead

Unroll k loop x times

+ 3 choices of compilers
+ implicit unroll options
+ zero / one based indexing

Prefetching example

An example using prefetch directives to prefetch data in a matrix multiplication inner loop where a row of one source matrix has been gathered into a contiguous vector might look as follows:

```
real*8 a(m,n), b(n,p), c(m,p), arow(n)
...
do j = 1, p
c$mem prefetch arow(1),b(1,j)
c$mem prefetch arow(5),b(5,j)
c$mem prefetch arow(9),b(9,j)
do k = 1, n, 4
c$mem prefetch arow(k+12),b(k+12,j)
c(i,j) = c(i,j) + arow(k) * b(k,j)
c(i,j) = c(i,j) + arow(k+1) * b(k+1,j)
c(i,j) = c(i,j) + arow(k+2) * b(k+2,j)
c(i,j) = c(i,j) + arow(k+3) * b(k+3,j)
enddo
enddo
```

This pattern of prefetch directives will cause the compiler to emit prefetch instructions whereby elements of arow and b are fetched into the data cache starting 4 iterations prior to first use. By varying the prefetch distance in this way, it is possible in some cases to reduce the effects of main memory latency and improve performance.

e.g. prefetch value