



# Optimisation Strategy for Exploiting the Cray X1 Architecture

**Bob Carruthers**  
Cray UK Ltd.  
*[crjrc@cray.com](mailto:crjrc@cray.com)*

# Overview

- Introduction
- Supercomputer Heritage
- Cray X1 Overview
- Compiler Optimisation
- Hand Optimisation
- Questions

# Vector Computer Types 1

- Memory to Memory
  - CDC STAR-100
  - CDC CYBER 205
  - ETA-10
- Characteristics
  - Segmented Functional Units
  - High Memory Bandwidth – Obligatory!
  - Virtual Memory
- No Longer Made!

# Vector Computer Types 2

- Register to Register
  - Cray Research Range
  - Japanese Supercomputers
  - Cray Inc Products
- Characteristics
  - Segmented Functional Units
  - High Memory Bandwidth
  - Shared Memory
  - Real Memory Model
  - Complex Interconnect Technology

# Interconnect performance

1979: Standard DRAM

- 16 Kbit**
- 1-bit wide interface**
- 5 Mb/s uniform access BW**
- 2 Mb/s random access BW**



1999: 200 MHz SDRAM

- 256 Mbit**
- 16 bit wide interface**
- 3200 Mb/s uniform access BW**
- 1000 Mb/s random access BW**

# Interconnect Performance

**1979 → 1999:**

- 16000X density increase
- 640X uniform access BW increase
- 500X **random access** BW increase
- 25X *less* per-bit memory bandwidth

# Architectural Implication

For vector system performance to advance at Moore's Law rate, vector systems must adopt *hierarchical memory*

- Distributed memory
- Caches
- Work decomposition and cache blocking
- Shorter vector lengths and multi-dimensional loops

# Multi Streaming Vector Processors

- Tightly couple four 2-pipe vector units
  - Can use shared cache, special sync primitives
- On long VL inner loops
  - apply all 8 pipes to same loop
  - same performance as wide-pipe unit
- On nested inner loops with short VL
  - each vector unit takes a whole inner loop, (one iteration of a 2nd level loop)
  - keeps vector chime time reasonable
  - avoids Amdahl's Law degradation
  - much more flexible than a single wide-pipe unit

# Multistreaming Example

Single Long Vector Loop

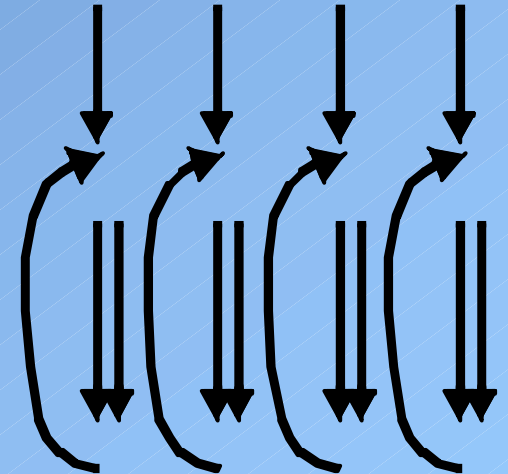
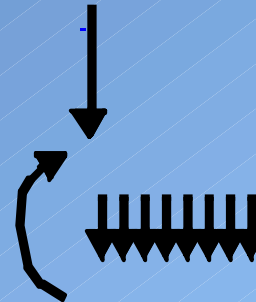
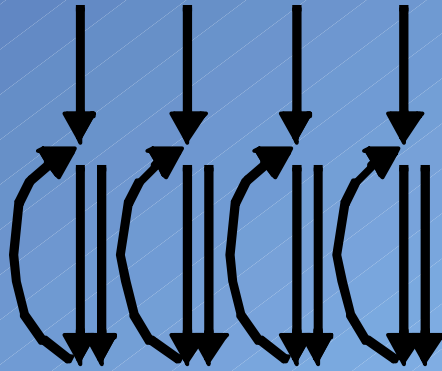
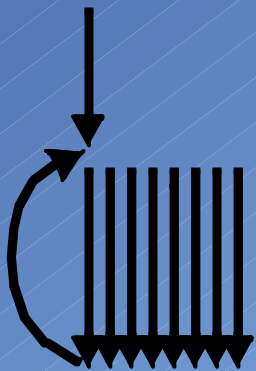
Short Nested Loops

8 pipe vector proc

4x2 pipe MSP

8 pipe vector proc

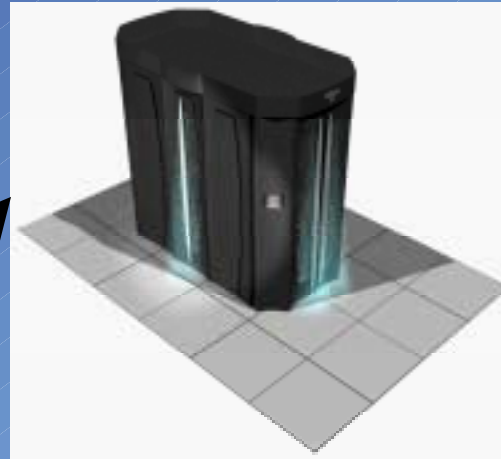
4x2 pipe MSP



# T3E Scalability and Vectors?

- Shared address space for user-level, load/store access to all memory
  - Even easier with custom processor (same access mechanism).
- Stream buffers for local memory latency tolerance
- Highly pipelined memory access/communication (512 E reg's)
  - Vector registers better! (larger, easier, scatter/gather)
  - Great *latency tolerance* for scalable systems.
- High bandwidth, low-latency interconnection network
  - Yes, vector processors can *really* use this.
- Remote address translation (source translation doesn't scale)
- Optimized coherence protocol (global caching gets in the way)
  - Must have this. Increase BW for vector processors.

# Extreme Scalability with High Bandwidth Vector Processors



**Cray X1**

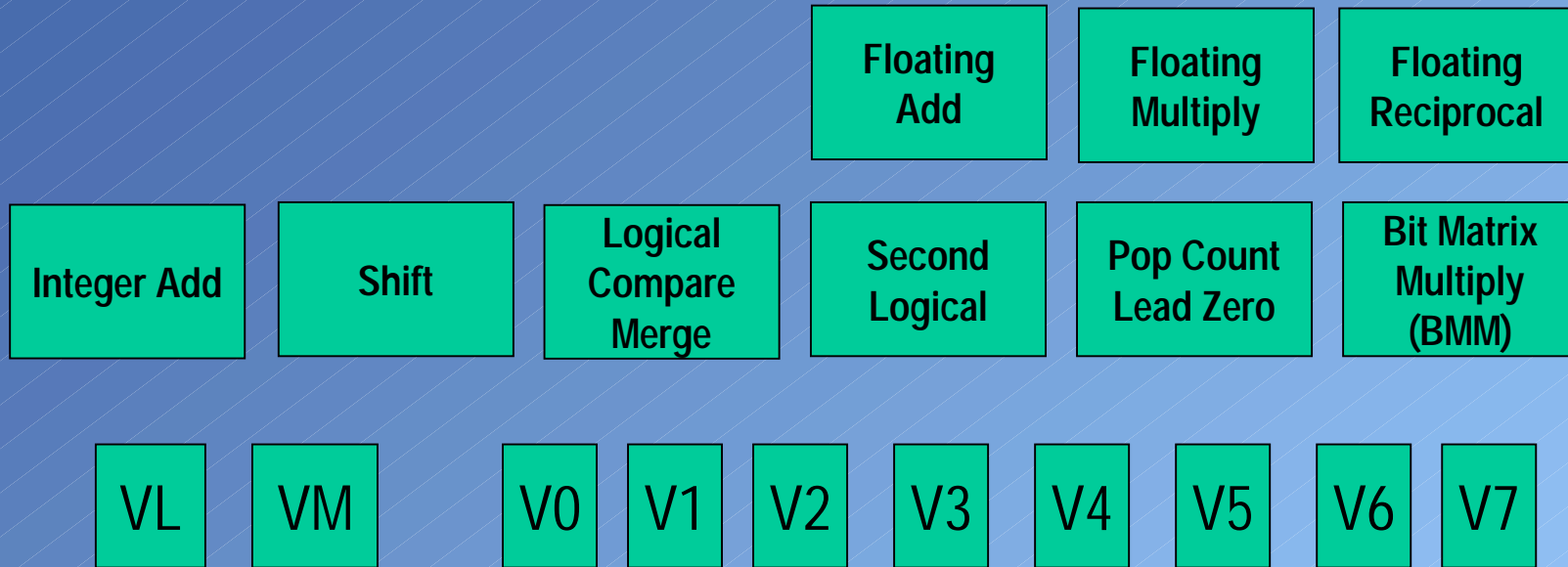
## **Cray PVP**

- Powerful vector processors
- Very high memory bandwidth
- Non-unit stride computation
- Special ISA features
- **Modernized the ISA**

## **T3E**

- Extreme scalability
- Optimized communication
- Memory hierarchy
- Synchronization features
- **Improved via vectors**

# Cray SV1ex Vector Units



**Functional units are dual pipe.**

# Cray SV1 Vector Registers

- 8 Registers.
- 64 elements (words) / register.
- 64 bits / element.
- VL register defines how many elements are used.
- VM register holds results of vector comparison operations and controls vector merge operations.
- 2 ports per CPU to memory (2 loads or 1 load and 1 store).
- Up to 4 registers may have loads active.

# Cray X1 Vector Units

## Group 1

Integer Add  
FP Add  
Logical 1  
Integer Compare  
FP Compare

## Group 2

Integer Multiply  
FP Multiply  
Shift

## Group 3

FP Divide  
Square Root  
Integer Convert  
FP Convert  
Copy Sign  
Absolute Value  
Logical 2  
Leading Zero  
Pop Count  
Bit Matrix Multiply  
Vector Merge  
Iota

**Register File**

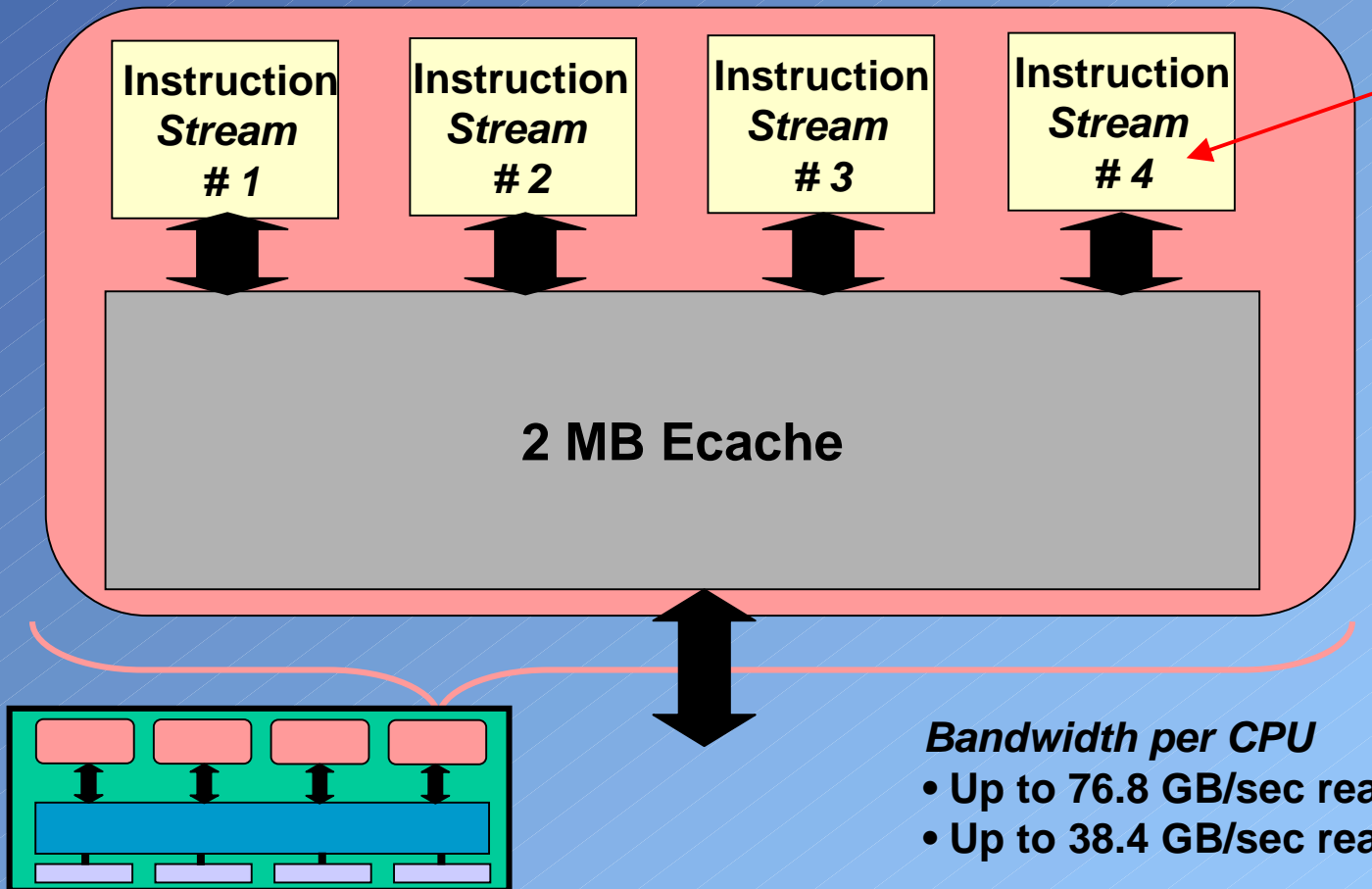
Functional units are dual pipe.

# Cray X1 Vector Registers

- **32 Registers.**
- **64 elements (words) / register.**
- **32 or 64 bits / element.**
- **VL register defines how many elements are used.**
- **8 VM registers control vector operations and hold results of vector comparison operations.**
- **Load buffers to allow 'load renaming'.**

# Multi-streaming Processor (MSP)

- New Cray Vector Instruction Set Architecture (ISA)
- 64- and 32-bit operations, IEEE floating-point



## *Each Stream:*

- 2 vector pipes (32 vector regs. of 64 element ea)
- 64 A & S regs.
- Instruction & data cache
- 1 x P-chip

## *MSP:*

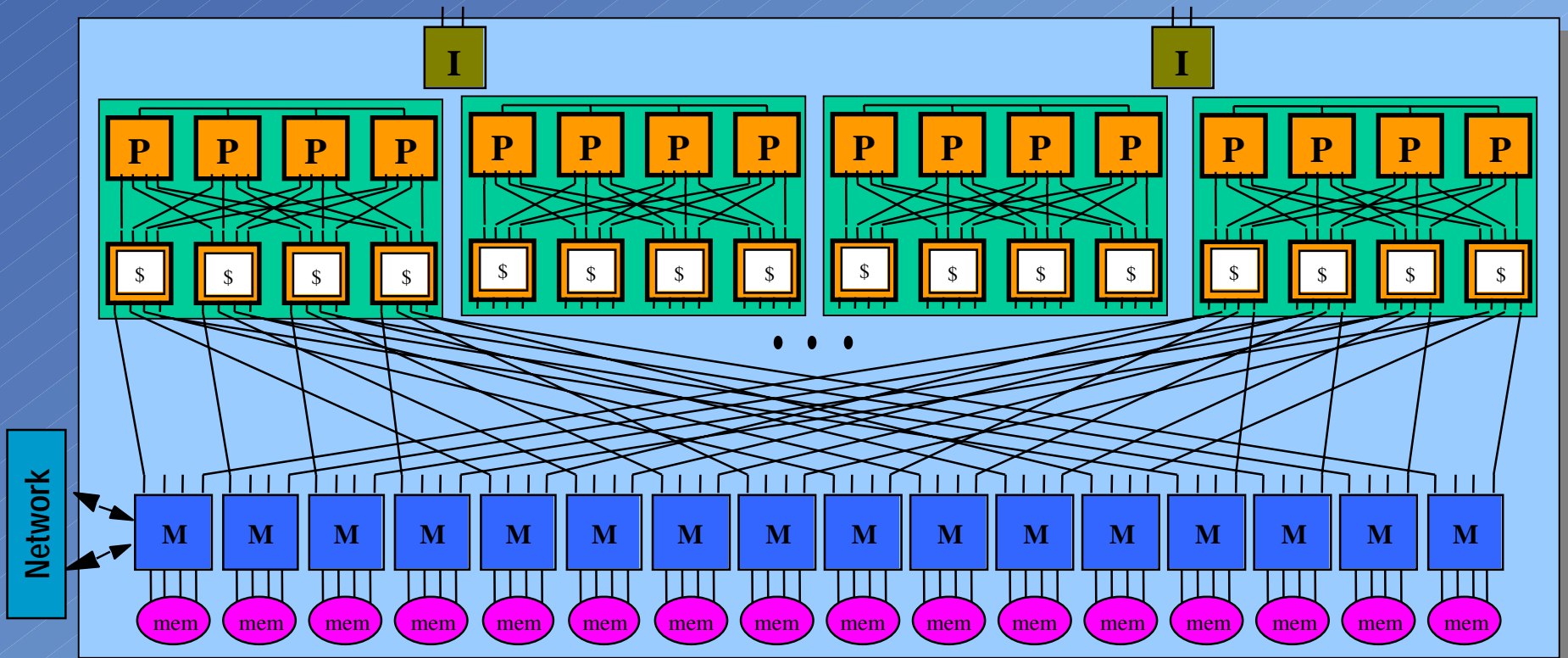
- 4 x P-chips
- 4 x E-chips (cache)
- shared cache
- sync. tokens

## *Bandwidth per CPU*

- Up to 76.8 GB/sec read/write to cache
- Up to 38.4 GB/sec read/write to memory

# SV2 Node - 50 GFLOPS (64-bit)

Two SPC I/O channels per I-chip



Inter-node network

Two ports per M-chip

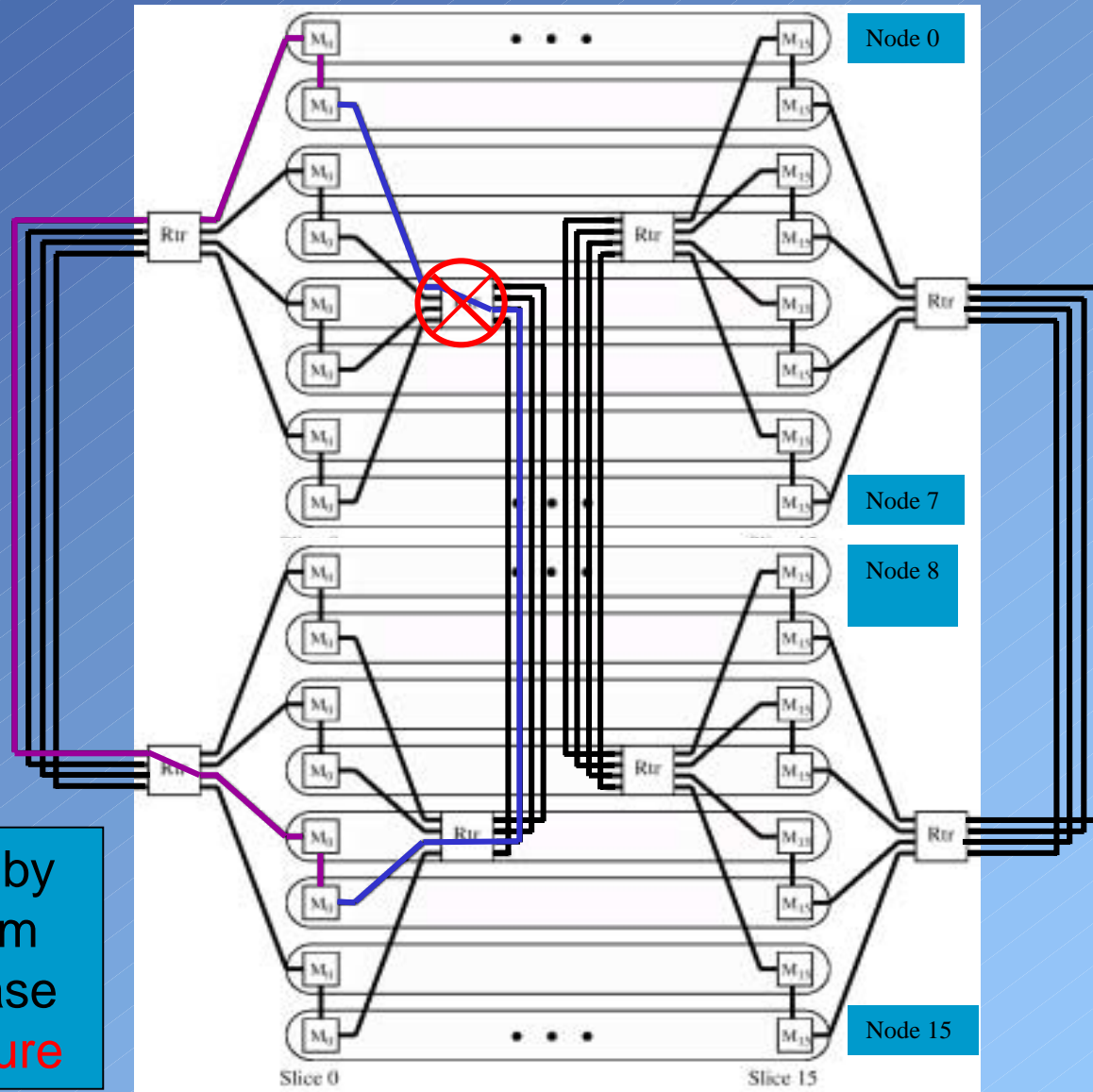
1.6 GB/s peak both directions per port

Local node memory

Peak BW = 16 slices x 12.8 GB/s/slice = 204.8 GB/s

Capacity = 16 to 64 GB

# 64 MSP System



redundancy by  
rerouting from  
— to — in case  
of **router failure**

# Compiler Optimisation

- Start with the Compiler
  - *STREAM*<sub>n</sub> Option
  - *VECTOR*<sub>n</sub> Option
  - Compiler Directives
    - *!DIR\$ IVDEP*
    - *!DIR\$ CONCURRENT*
    - *!DIR\$ PREFERVECTOR*
    - *!DIR\$ PREFERSTREAM*
    - *!DIR\$ INTERCHANGE*
    - *!DIR\$ SSP\_PRIVATE*
  - Simple Loop Re-arrangement
- Loosely Coupled memory Model

# Interchange Example 1

```
247. 1-----<      DO m=1,mt
248. 1
249. 1                ml=m0l(mfl)+m
250. 1                nl=iind(ml)
251. 1
252. 1                IF( ((m*100)/mt) > pct ) THEN
253. 1 2-----<      DO i=pct+1,((m*100)/mt)
254. 1 2                WRITE(UNIT=*,FMT="(a)",ADVANCE="no") "#"
255. 1 2----->      END DO
256. 1                END IF
257. 1                pct= ((m*100)/mt)
```

# Interchange Example 2

```
258. 1
259. 1 2-----< DO decal=-pas,pas
260. 1 2          xl=x(nl)
261. 1 2          yl=y(nl)
262. 1 2          zl=z(nl)
263. 1 2          if(decal.ne.0) then
264. 1 2              r=sqrt(yl**2+zl**2)
265. 1 2              theta=atan2(zl,yl)+float(decal)*thetarot
266. 1 2              yl=r*cos(theta)
267. 1 2              zl=r*sin(theta)
268. 1 2          endif
```

# Interchange Example 3

```
269. 1 2
270. 1 2 MV-----<      DO n = 1,ind_max
272. 1 2 MV                xn=x(n)
273. 1 2 MV                yn=y(n)
274. 1 2 MV                zn=z(n)
278. 1 2 MV                distance=(xn-xl)**2 + (yn-yl)**2 + (zn-zl)**2
282. 1 2 MV                IF( distance < dist(n) ) dist(n) = distance
284. 1 2 MV----->      END DO
285. 1 2----->          END DO
287. 1----->            END DO
```

# Interchange Example 4

```
259. 1 MV-----<      DO decal=-pas,pas
260. 1 MV                xl=x(nl)
261. 1 MV                yl=y(nl)
262. 1 MV                zl=z(nl)
263. 1 MV                if(decal.ne.0) then
264. 1 MV                  r=sqrt(yl**2+zl**2)
265. 1 MV                  theta=atan2(zl,yl)+float(decal)*thetarot
266. 1 MV                  yl=r*cos(theta)
267. 1 MV                  zl=r*sin(theta)
268. 1 MV                endif
269. 1 MV                i=i+1
270. 1 MV                txl(decal,m)=xl
271. 1 MV                tyl(decal,m)=yl
272. 1 MV                tzl(decal,m)=zl
273. 1 MV----->      enddo
274. 1----->        ENDDO
```

# Interchange Example 5

```
276.          !dir$ interchange (n,m,decal)
277. iC-----<      DO m=1,mt
278. iC i-----<    DO decal=-pas,pas
279. iC i          !dir$ prefervector
280. iC i iMV-----<      DO n = 1,ind_max
282. iC i iMV          xl=txl(decal,m)
283. iC i iMV          yl=tyl(decal,m)
284. iC i iMV          zl=tzl(decal,m)
285. iC i iMV          xn=x(n)
286. iC i iMV          yn=y(n)
287. iC i iMV          zn=z(n)
291. iC i iMV          distance=(xn-xl)**2 + (yn-yl)**2 + (zn-zl)**2
295. iC i iMV          IF( distance < dist(n) ) dist(n) = distance
297. iC i iMV----->      END DO
298. iC i----->        END DO
300. iC----->        END DO
```

# Simple Re-arrangement 1

```
109. 1-----<   DO K = K1M1,K2P1,KINC
110. 1           KS = -KS
111. 1 2-----<   DO J = J1,J2
112. 1 2       !DIR$ IVDEP
113. 1 2       !CDIR NODEP
114. 1 2       !OCL NOVREC
115. 1 2 V--<    DO I = I1,I2
116. 1 2 V      N = IND(I,J,K)
117. 1 2 V      V0(N) = EX1*V0(N+KS*NCK)+EX2*V0(N+KS*NCK2)
118. 1 2 V      V1(N) = EX1*V1(N+KS*NCK)+EX2*V1(N+KS*NCK2)
119. 1 2 V      V2(N) = EX1*V2(N+KS*NCK)+EX2*V2(N+KS*NCK2)
120. 1 2 V      V3(N) = EX1*V3(N+KS*NCK)+EX2*V3(N+KS*NCK2)
121. 1 2 V      V4(N) = EX1*V4(N+KS*NCK)+EX2*V4(N+KS*NCK2)
122. 1 2 V-->   ENDDO
123. 1 2-----> ENDDO
124. 1----->  ENDDO
```

# Simple Re-arrangement 2

```
134.      !dir$ concurrent
135. M-----<   DO J = J1,J2
136. M           KS = -1
137. M 2----<   DO K = K1M1,K2P1,KINC
138. M 2         KS = -KS
139. M 2   !DIR$ IVDEP
140. M 2   !CDIR NODEP
141. M 2   !OCL NOVREC
142. M 2 V--<   DO I = I1,I2
143. M 2 V       N = IND(I,J,K)
144. M 2 V       V0(N) = EX1*V0(N+KS*NCK)+EX2*V0(N+KS*NCK2)
145. M 2 V       V1(N) = EX1*V1(N+KS*NCK)+EX2*V1(N+KS*NCK2)
146. M 2 V       V2(N) = EX1*V2(N+KS*NCK)+EX2*V2(N+KS*NCK2)
147. M 2 V       V3(N) = EX1*V3(N+KS*NCK)+EX2*V3(N+KS*NCK2)
148. M 2 V       V4(N) = EX1*V4(N+KS*NCK)+EX2*V4(N+KS*NCK2)
149. M 2 V-->   ENDDO
150. M 2---->   ENDDO
151. M----->   ENDDO
```

# SSP\_PRIVATE

!dir\$ ssp\_private radlsw

.

DO JRL=1,IRLONR,KRPROMA

IBEG=JRL

IEND=MIN(IBEG+KRPROMA-1,IRLONR)

IRLONI=IEND-IBEG+1

!dir\$ stream

CALL RADLSW &

&( ibeg , iend , KSPROMA, KTDIA , KLEV &

&, KMODE , NAER &

&, ZRIIO &

&, ZSAER , ZSALD , ZSALP , ZSHPR , ZSPR , PCCO2 &

&, ZSCLC , ZSDP , ZSALTE , ZSALTW &

&, ZSSLM , ZSMU0 , ZSOZ , ZSWV &

&, ZSIWA , ZSLWA , ZSQS , ZSRWA , ZSRRA &

&, ZSHTI , ZSTI , ZSTS , ZSBAS , ZSTOP &

! FLUX OUTPUTS

&, ZSEMIT, ZSLWFC, ZSFRTH , ZSSWFC, ZSFRSO, ZSFRSOD &

&, ZSSUDU &

&)

ENDDO

# CSD Directives

- **!CSD\$ PARALLEL**
- **These directives mirror !OMP\$ directives**
- **Help in defining Local and Shared Variables**
- **Example is Complex Matrix Multiply**
- **This code sustains over 10 Gflops/s for a whole code**

# Complex Matrix Multiply

```
!CSD$ PARALLEL DO SCHEDULE(STATIC,1)
```

```
!dir$ unroll 2
```

```
DO J = 1, N
```

```
!dir$ prefervector
```

```
do i = 1, m
```

```
sum = c( i, j)*beta
```

```
!dir$ nextscalar
```

```
!dir$ unroll 4
```

```
DO L = 1, K
```

```
temp=ALPHA*B( L, J )
```

```
sum = sum + temp*A( I, L )
```

```
end do
```

```
C(I,J) = sum
```

```
end do
```

```
end do
```

```
!CSD$ END PARALLEL DO
```

# Communication Optimisation 1

- **MPI**
- **SHMEM**
- **Co-array Fortran**
- **UPC**
- **Direct addressing across the whole Machine**

# Communication Optimisation 2

Real (kind8) :: a(1000), b(1000)

POINTER (PTR, b)

INTEGER pe

!

! Get the virtual address of 'b' on the remote PE 'pe'

!

PTR = SHMEM\_PTR(b, pe)

!

! Send data to PE 'pe'

!

Do I=1,1000

  b(I)=a(I)

End do

# Questions

# End