# Cray SV1 Application Performance

**James Giuliani and David Robertson**

*Science & Technology Support Group*
*Ohio Supercomputer Center*
*1224 Kinnear Road*
*Columbus, OH  43212-1163*
*USA*

# Goals

To gain insight into performance issues arising from the new architectural features of the SV1

- Vector cache
- Multi-stream processors (MSPs)

To begin to study how these performance issues will impact the migration of codes, developed for Y-MP/C90/T90 series machines, to Cray’s new architecture roadmap

To characterize the machine performance on a variety of real-world applications

# OSC Systems

CrayT90

4 processors @ 450MHz

Shared memory - 1 Gbyte

Origin 2000

32 processors, R12K @ 300MHz

16 Gbytes mem.(1 gb/2 processors)

Cray SV1

16 processors, rev1b@300MHz

Shared memory - 16 Gbytes

CrayT3E

128 Processors, Alpha EV5
at 300 MHz

Distributed memory -16 Gbytes

128 Processors, Intel PIII Xeon
at 550 MHz

Distributed memory - 64 Gbytes
(2 gb/4 processors)

# Outline

SV1 Architecture Highlights

Single-Processor Performance Issues

Multi-Processor Performance Issues

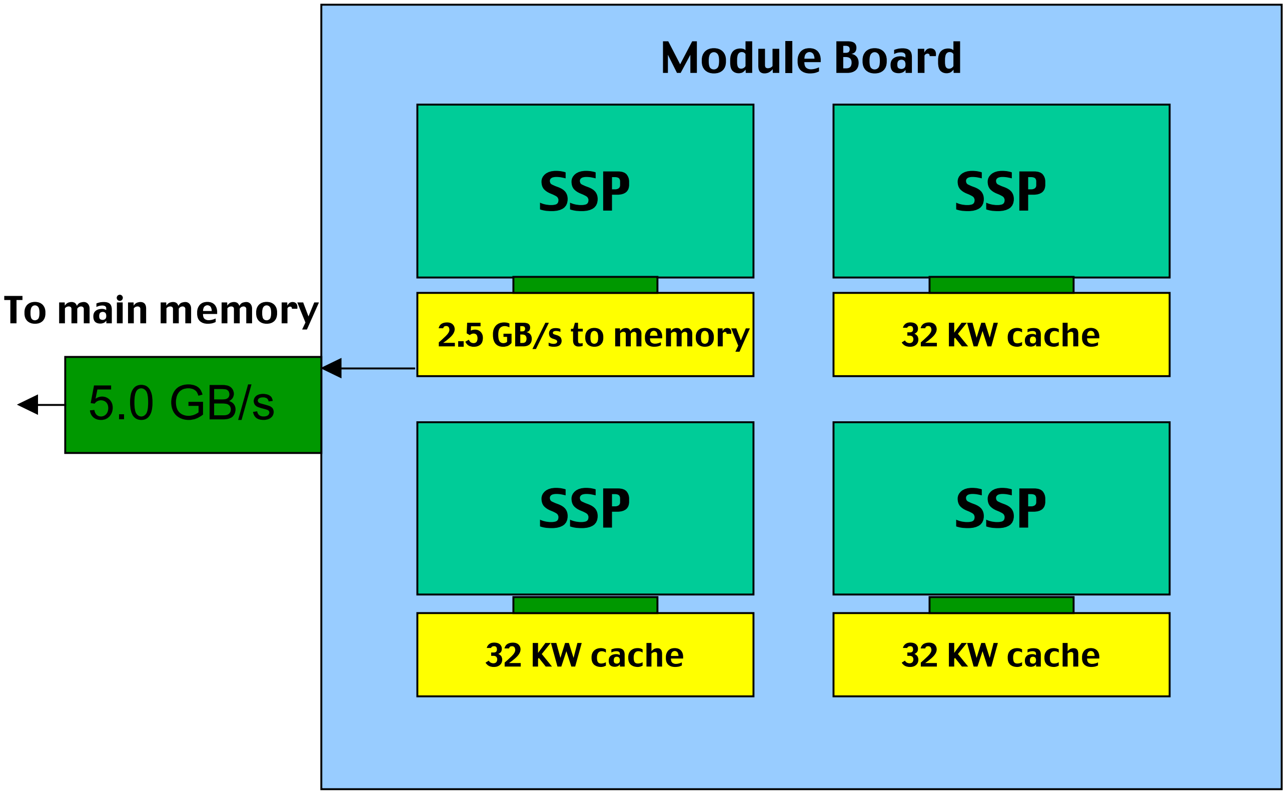
Conclusions

# SV1 Architecture Highlights

"Single Stream Processors" (SSPs)

- 300 MHz clock (3.33 ns clock period)
- Dual-pipe floating-point units
  - Can produce two add-multiply results per clock period
  - Scalar operations share one of the pipes
- Theoretical peak performance: 1.2 GFLOP/sec per SSP
- Vector length of 64 words
- 32 Kword (256 Kbyte) cache for vector, scalar and instruction loads
- Four SSPs per module board
  - Module-to-memory bandwidth is about 5 GB/sec (6.4 GB/sec peak)
  - Each processor can access around 2.5 GB/sec (3.2 GB/sec peak)

# SV1 Architecture



# The SV1 Data Cache

- Each SSP has its own cache for both scalar and vector loads
- 32K words (256 Kbytes) per SSP
- SSP-to-cache bandwidth:
  - Two read ports and one write port per pipe
  - 4 words/CPU CP for reads (9.6 GB/sec)
  - 2 words/CPU CP for writes (4.8 GB/sec)
- Four-way set associative
  - Replacement policy: Least Recently Used
- Line size of eight words for scalar loads
- Line size of one word for vector loads
  - No wasted bandwidth for irregular strides through main memory
- “Write through” - each store goes to main memory
- “Write allocate” - each store goes to cache

***“Software Coherent” for multiprocessor applications***

# The Multi-Stream Processor

Four SSPs configured to act like one super-processor

- 4.8 GFLOP/sec theoretical peak performance

Cache coherency currently a significant performance bottleneck

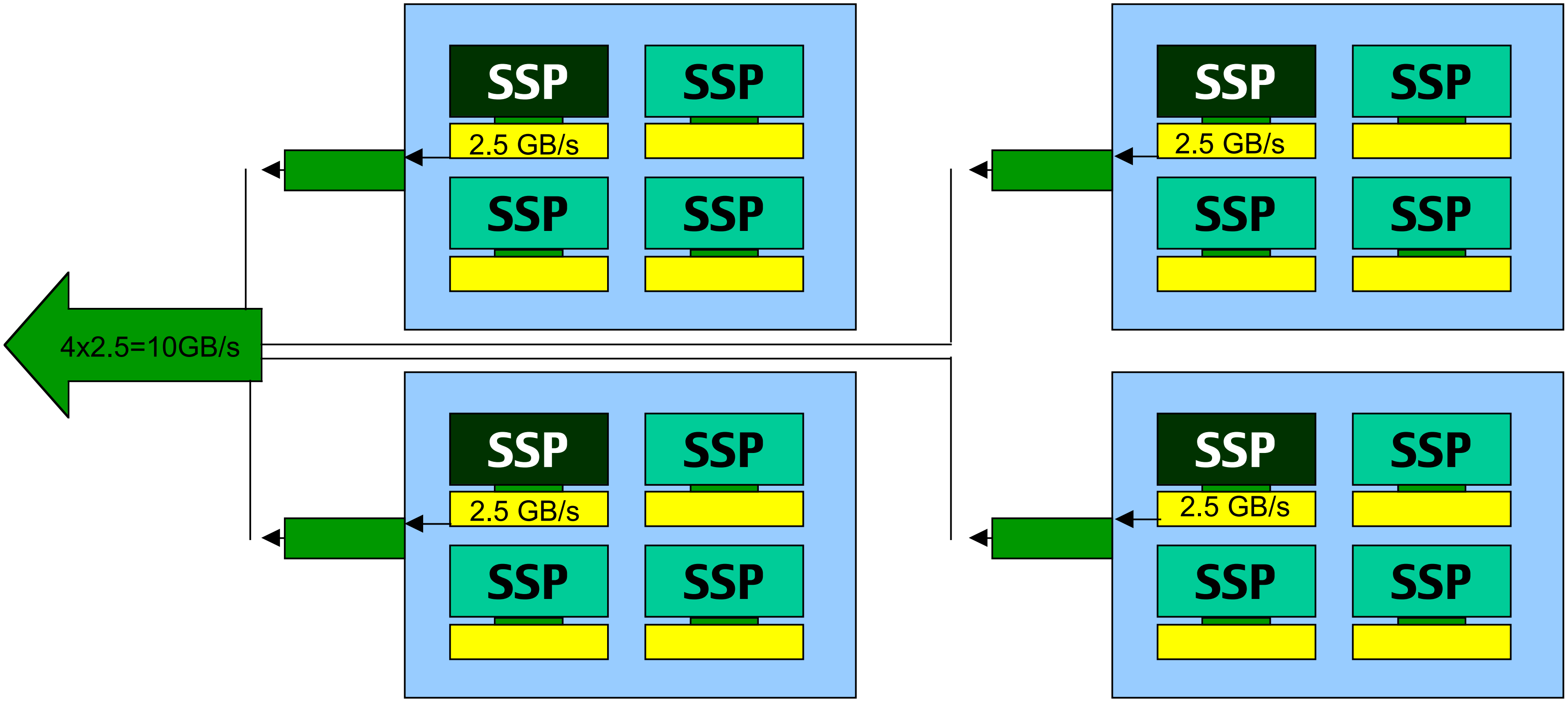
Shared B and T registers

Configured using one SSP from each of four module boards to achieve optimum access to main memory

- Minimizes contention for bandwidth to main memory
- Net memory-to-MSP bandwidth: ~10 GB/sec

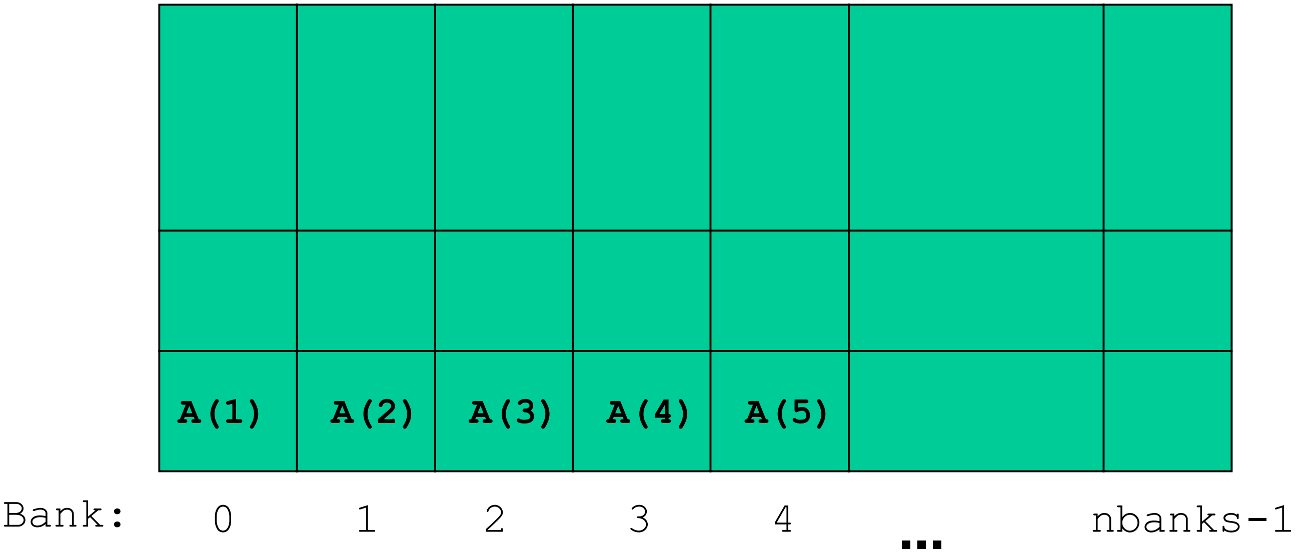
The SV2 will consist entirely of MSPs

# The Multi-Stream Processor



# Memory Structure

- Same memory hardware as the J90, although the SV1 CPU is six times faster
- Interleaved:



A(1) A(2) A(3) A(4) A(5)

Bank: 0 1 2 3 4 ... nbanks-1

- Bank busy time = 14 CP
- At OSC nbanks = 512

# Single Processor Performance Issues

General Considerations

Examples

- Loop performance
- User F90 application
- 3rd party application

Lessons Learned

# Vector Code Performance

Our experience suggests that typical performance is 30-50% of a T90 for codes that vectorize well

- Tends to the lower end of this range for long vector lengths, or codes requiring heavy memory bandwidth
- Tends toward the higher end for shorter vector lengths

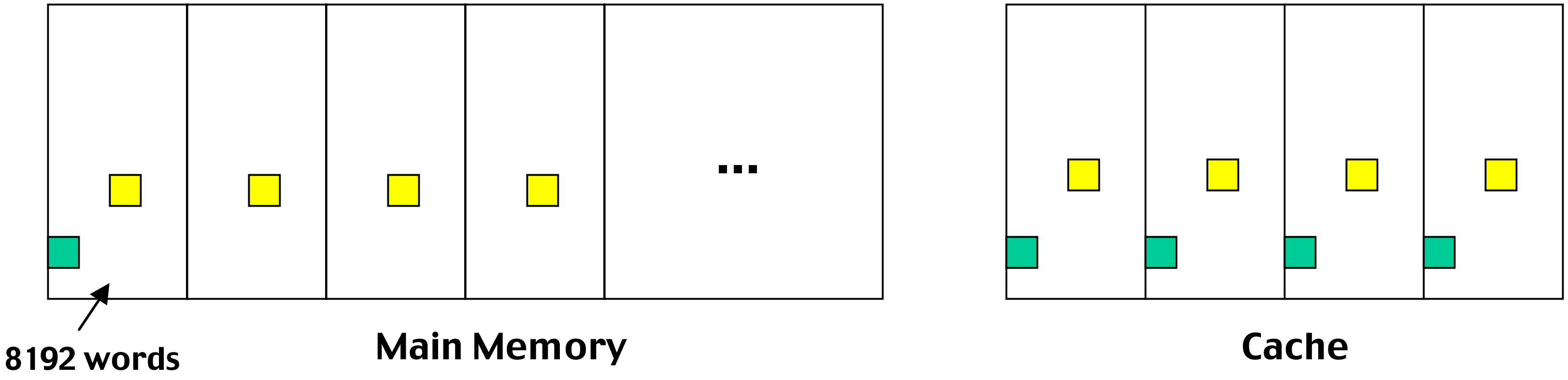
Problems that take advantage of the cache also run very well

- Problems that fit in cache
- Codes with exploitable temporal locality (data reuse)

For scalar code, performance is typically 60-80% of the T90

# Cache Considerations

- Latency when fetching data from the cache is 4-5 times lower than when fetching directly from main memory
- Effective cache use reduces contention for bandwidth to main memory
- Memory locations that are 8192 words apart map to the same four way cache "slot"
  - Four loads with this stride will fill it up ("thrashing")



# Memory Bank Conflicts

Traditional Cray vector computers fetch vector operands directly from main memory, so the potential for memory bank conflicts is very sensitive to the way algorithms step through memory

The SV1 vector cache now resides between memory and the CPU and it can have a significant mitigating effect on bank conflicts

- Ability to pre-fetch data into the cache
- Data may be resident in the cache, eliminating fetches to memory
- Cache can significantly improve performance when stride is power of two
- Cache only helps; performance is never worse than “cache off” scenario

# Example: Vector Code

```
do j=1,n
   c(j)=1.0/a(j) + f(j) + (1.0/b(j) + g(j)) * e
end do
```

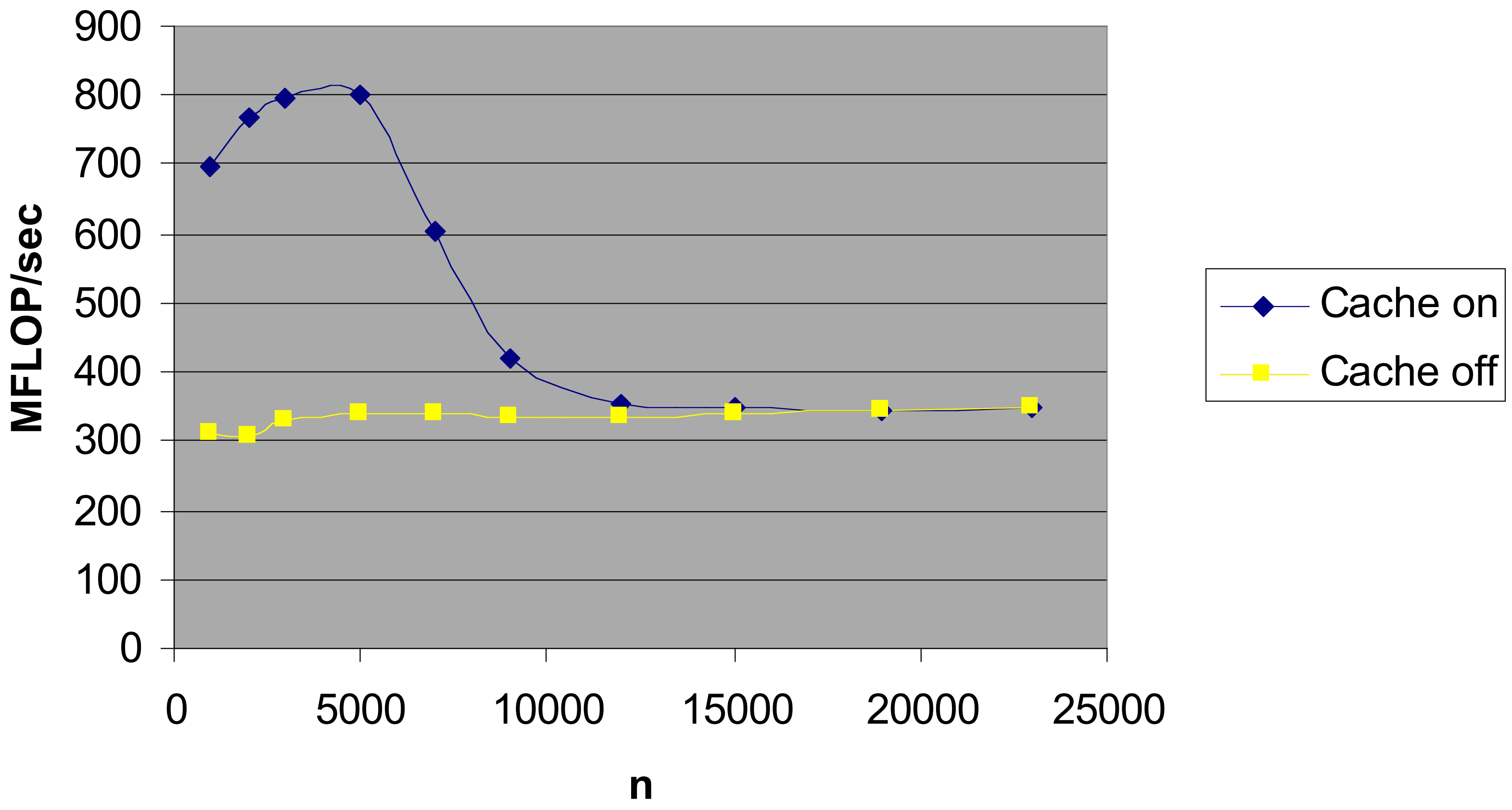
5 multiplies, 3 adds, 2 reciprocals per iteration

For `n` of order 2000

- vector lengths near maximum
- all arrays fit in cache

For `n` of order 7000, data does not all fit in cache

# In-Cache vs Out-of-Cache



# Example: Laplace Equation Solver

SOR code for solving Laplace's equation in 2-d space

Vector version

- Tuned for T90
- All computations well vectorized

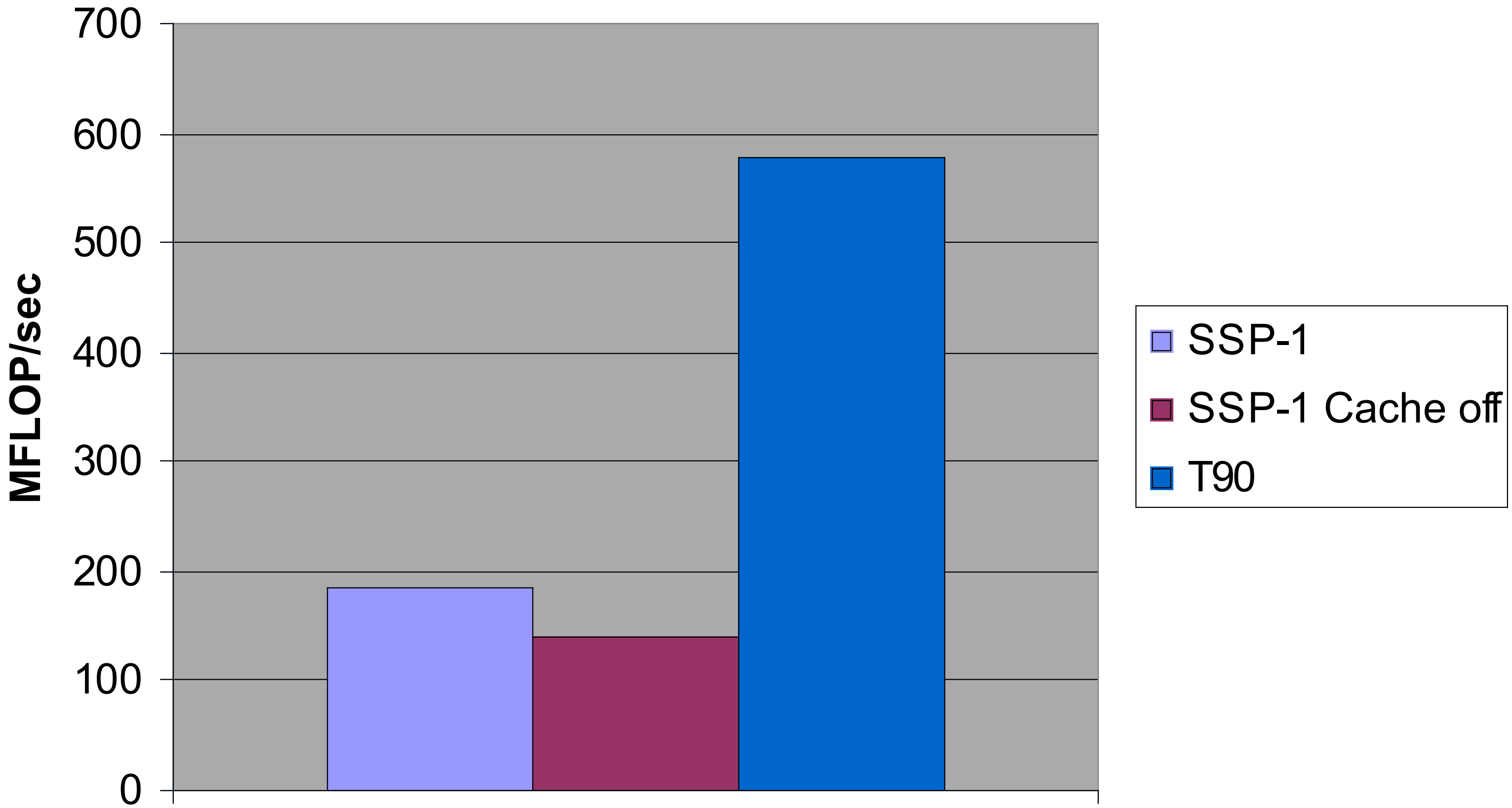
"Cache-friendly" version

- Tuned for microprocesors
- Designed for efficient re-use of cached data
- Scalar code

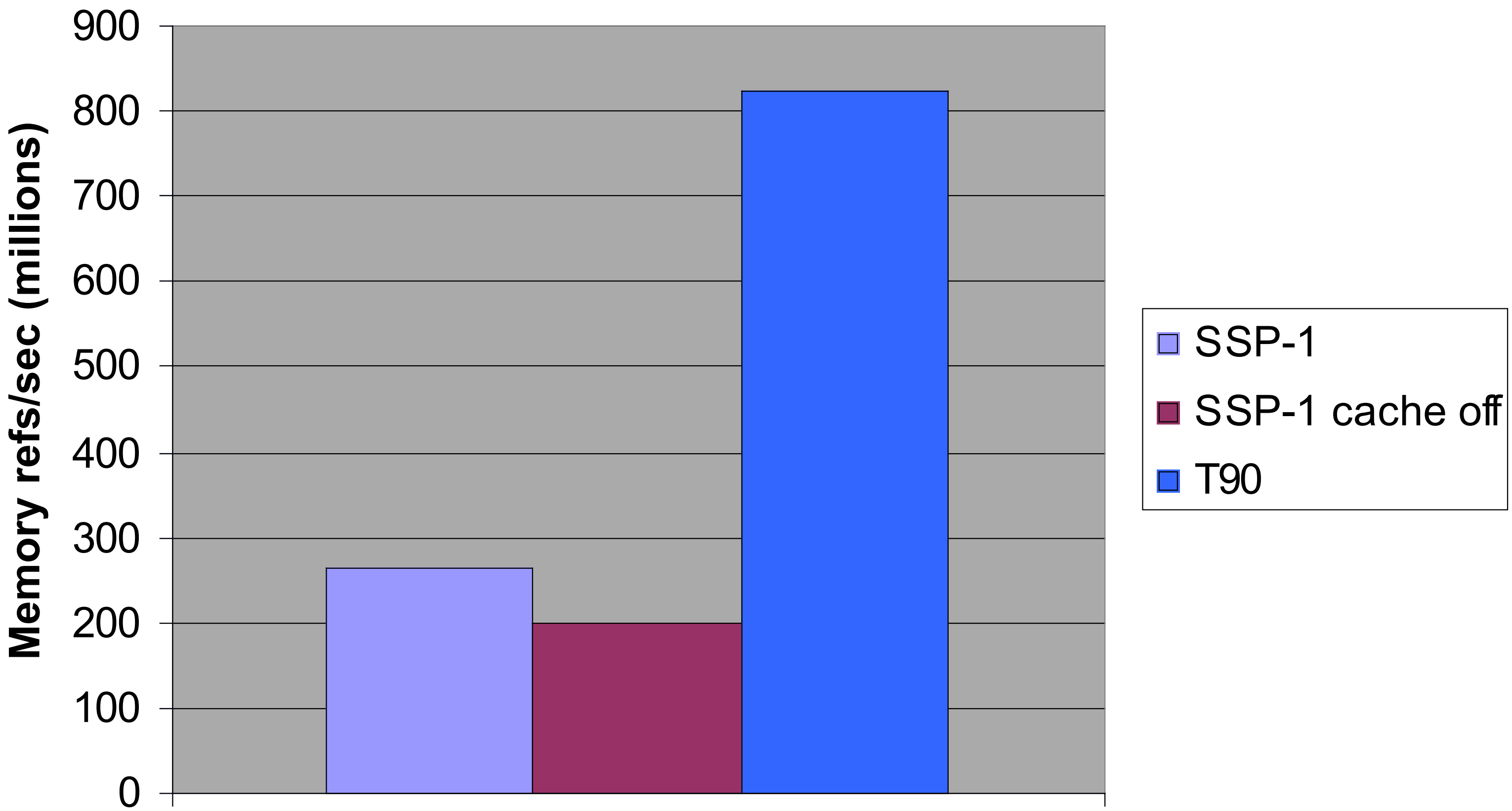
Test problem size: 2001 by 2001

- Data arrays don't fit in cache

# SSP Performance: Vector Version



# SSP Memory Bandwidth: Vector



# SSP Performance: Gaussian98

*Ab initio* quantum chemistry

- Version A.7

Probably the single most popular 3rd party application at OSC

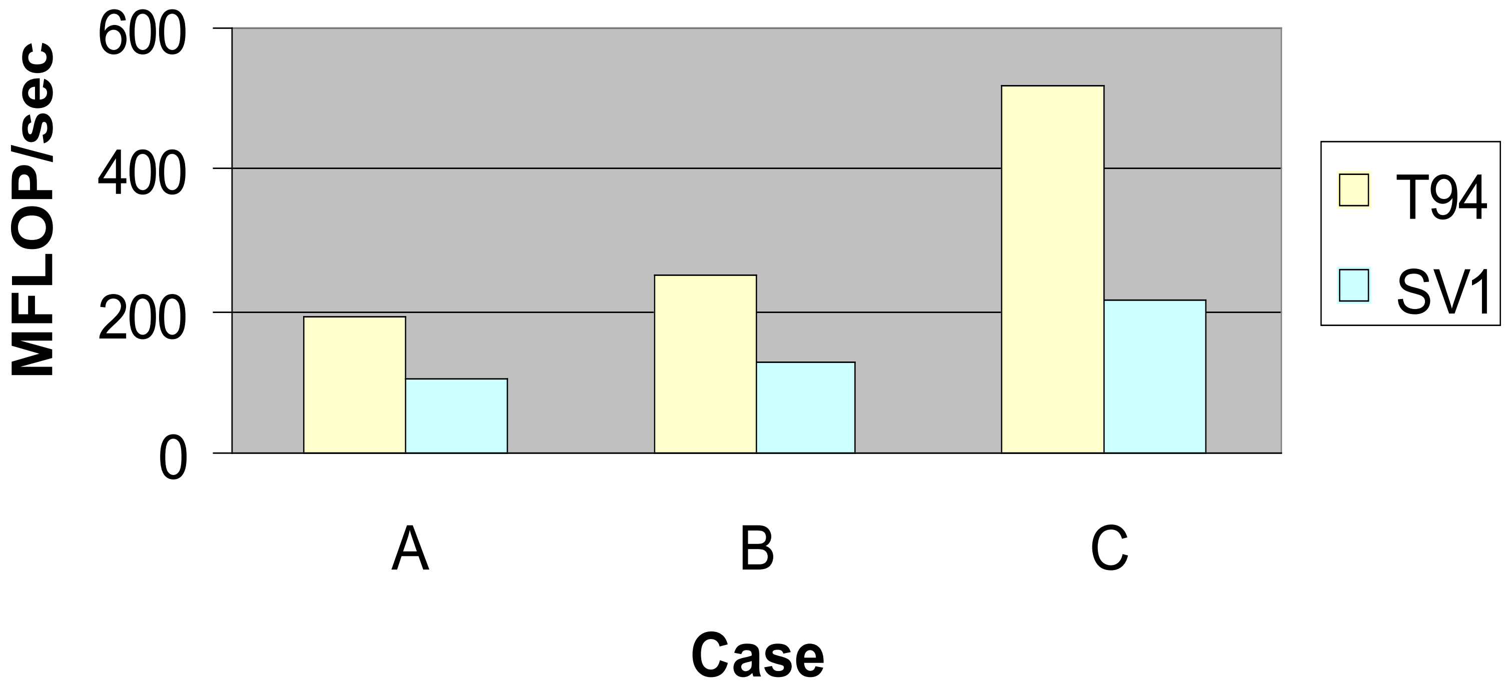
Consider mostly CPU bound calculations

Examine

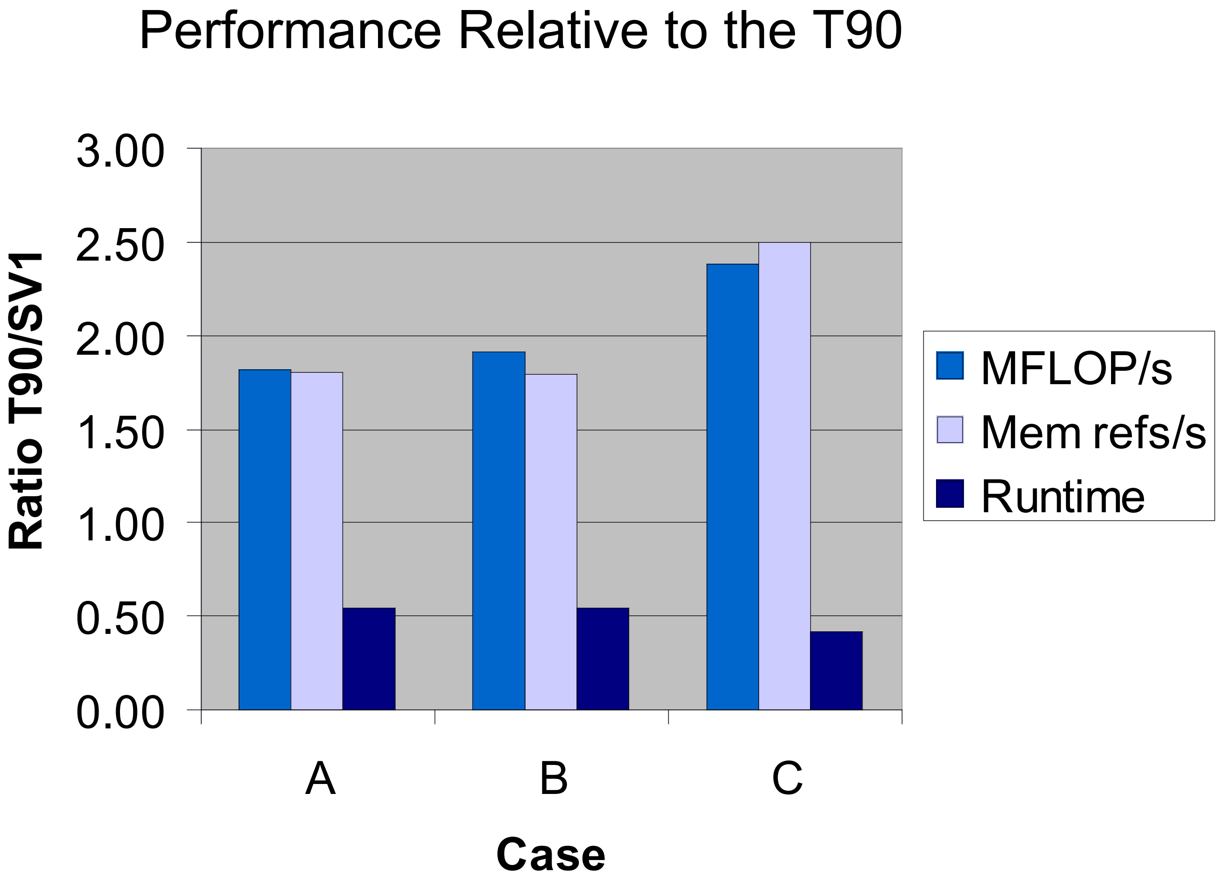
- Overall SSP performance (runtime)
- Floating-point performance
- Memory throughput

# Gaussian98

## Floating-Point Performance



# Gaussian98


Performance Relative to the T90
Ratio T90/SV1
3.00
2.50
2.00
1.50
1.00
0.50
0.00
A
B
C
Case
MFLOP/s
Mem refs/s
Runtime

# Single Processor Performance Issues
# Lessons Learned

- The SV1 is still a vector machine!
- Basic cache optimization rules obtain:
  - best to structure application so that data fits in cache (cache blocking)
  - best to access data with small memory strides
  - avoid strides that are a large power of two especially
- Probably best to focus on these first, rather than high VLs
- Unlike caches with line sizes greater than one, thrashing is no worse than computing without the cache
- The cache can hide the effects of bank conflicts
  - Also at their worst for power-of-two strides
- Acceptable cache hit rates are in the 40-60% range, depending on application
- Many codes can achieve 50% of T90 performance

# Multi–Processor Performance Issues

The Multi-Stream Processor

Examples

- Two user applications
- One 3rd party application
- One MPI application

Lessons Learned

# The Multi-Stream Processor

Multi-streaming works on loops

```
do j=1,1000
   a(j)=b(j)*2.71828
end do
```

| | |
|---|---|
| SSP0: | j=1, 250 |
| SSP1: | j=251, 500 |
| SSP2: | j=501, 750 |
| SSP3: | j=751, 1000 |

In simple cases it acts as a pipe/register/cache multiplier

– Can result in super-linear speedups for appropriately sized problems

Loops that don’t vectorize (e.g. outer loops) can be streamed

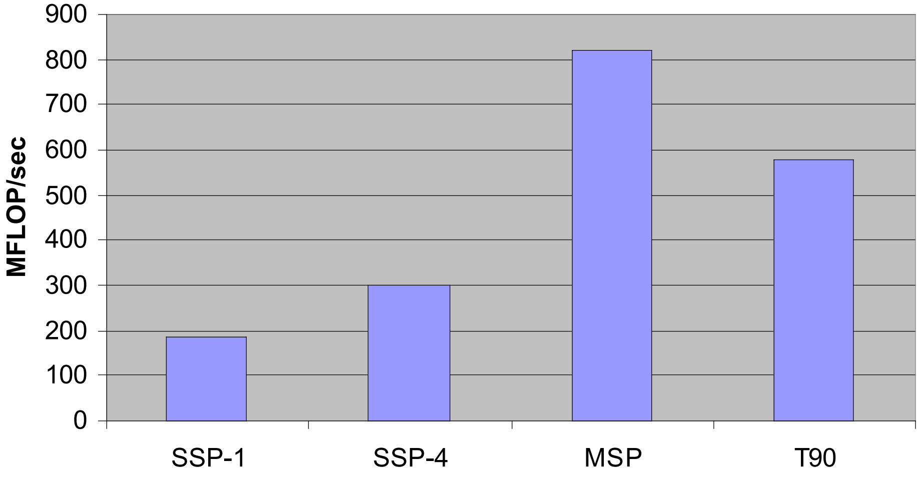
– Can stream outer loops while vectorizing inner loops

Must eliminate conditions that inhibit streaming

# Laplace Equation Solver – Streamed

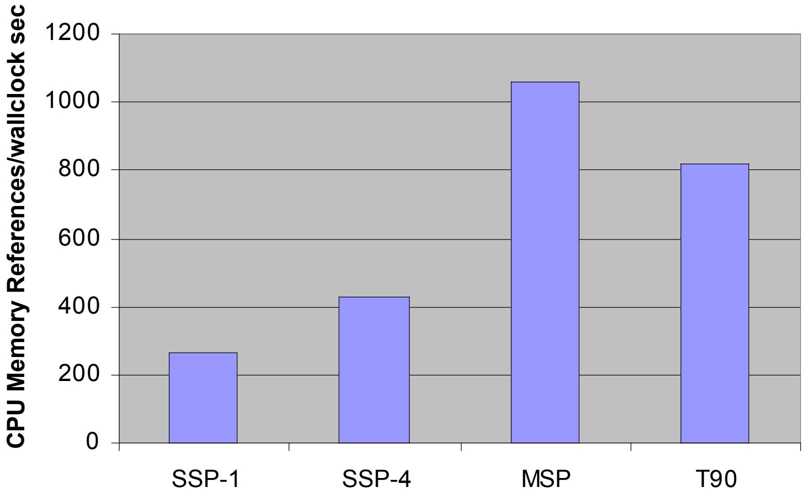
```
18               1 --------     do it=1,itmax
19               1                 dumax=0.0
20            M--1v -------        do j=2,jm1
21            M  12v ------           do i=2,im1
22            M  12v                     du(i,j)=0.25*(u(i-1,j)+u(i+1,j)
                                               +u(i,j-1)+u(i,j+1))-u(i,j)
23            M  12v ----->           enddo
24            M--1v ------>        enddo
25               12 -------        do j=2,jm1
26               12v ------           do i=2,im1
27               12v                     dumax=max(dumax,abs(du(i,j)))
28               12v ----->           enddo
29               12 ------>        enddo
30            M--12 -------        do j=2,jm1
31            M  12v ------           do i=2,im1
32            M  12v                     u(i,j)=u(i,j)+du(i,j)
33            M  12v ----->           enddo
34            M--12 ------>        enddo
35               1                 write (1,*) it,dumax
36               1 ------->     enddo
```

# Computational and Memory Performance



## Floating-Point Performance

- Larger apparent cache yields greater than 4x speedup for MSP executable
- Average vector lengths:
  T90: 99.58
  SV1: 60.53



## Memory Performance

- Again, the computational performance scales with memory performance
- MSP shows significantly better memory throughput over autotasked code

# Nonlinear Wave Equation Solver

Originally tuned for Cray-YMP

Loop intensive, but shorter average vector length

Three versions of the code:

- V1 - Most significant loops optimized for tasking and vectorization, but memory bank conflicts exist
- V2 - Bank conflicts eliminated
- V3 - Remaining serial loops optimized for tasking/streaming

Traditional development cycle would go through V1 and V2 to get highest single processor performance

Third step necessary to maximize performance on MSP

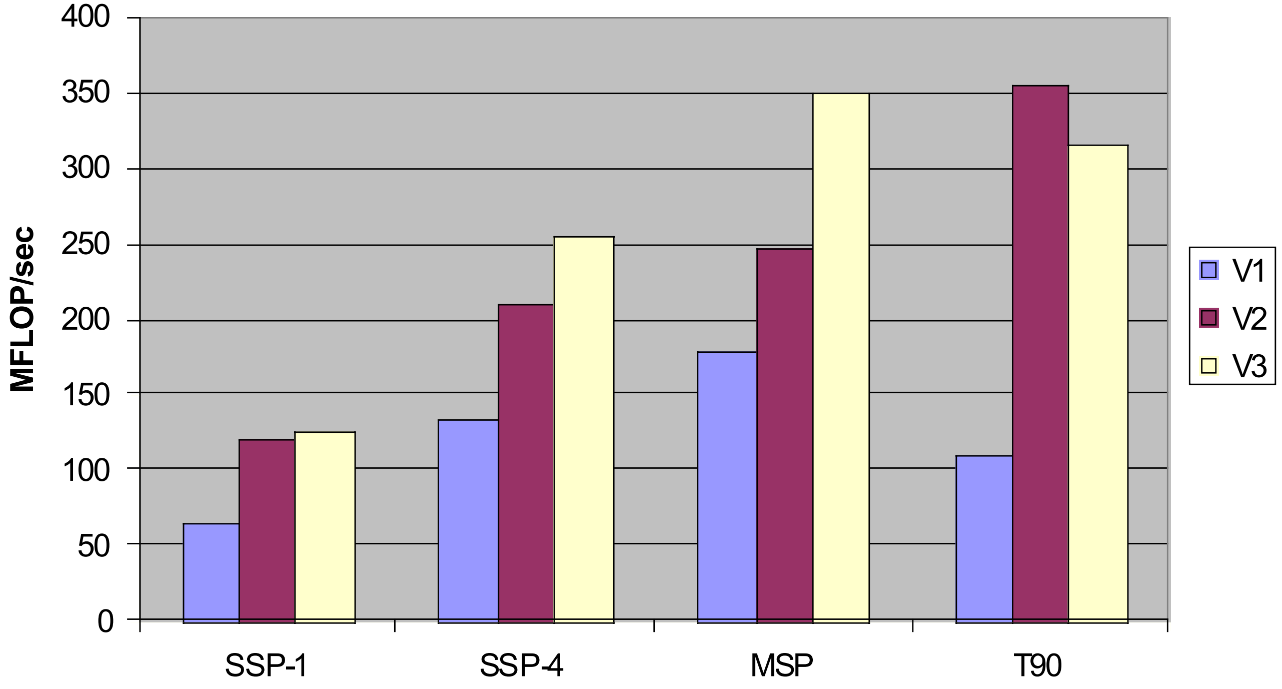
# Code Version 2 – Streamed

```
1                 ANUM = 0.0
1                 ADEN = 0.0
1v -------        DO 131 j=1,Njj
123 ------        DO 131 i=1,Nii
123v -----        DO 131 k=1,Nkk
123v              ANUM=ANUM+conjG(GDn(i,j,k))*HDn(i,j,k)
123v            &       +conjG(GVz(i,j,k)) *HVz(i,j,k)
123v            &       +conjG(GVr(i,j,k)) *HVr(i,j,k)
123v            &       +conjG(GVa(i,j,k)) *HVa(i,j,k)
123v              ADEN=ADEN+conjG(XDn(i,j,k))*XDn(i,j,k)
123v            &       +conjG(XVz(i,j,k)) *XVz(i,j,k)
123v            &       +conjG(XVr(i,j,k)) *XVr(i,j,k)
123v            &       +conjG(XVa(i,j,k)) *XVa(i,j,k)
123v ====>  131   CONTINUE
```

# Code Version 3 – Streamed

```
M-- 12 -------        DO 131 j=1,Njj
M   123 ------        DO 131 i=1,Nii
M   123v -----        DO 131 k=1,Nkk
M   123v              tmp1(i,j,k)=conjG(GDn(i,j,k))*HDn(i,j,k)
M   123v            &        +conjG(GVz(i,j,k)) *HVz(i,j,k)
M   123v            &        +conjG(GVr(i,j,k)) *HVr(i,j,k)
M   123v            &        +conjG(GVa(i,j,k)) *HVa(i,j,k)
M   123v              tmp2(i,j,k)=conjG(XDn(i,j,k))*XDn(i,j,k)
M   123v            &        +conjG(XVz(i,j,k)) *XVz(i,j,k)
M   123v            &        +conjG(XVr(i,j,k)) *XVr(i,j,k)
M   123v            &        +conjG(XVa(i,j,k)) *XVa(i,j,k)
M-- 123v ====>  131   CONTINUE
    1                 ANUM=SUM(tmp1)
    1                 ADEN=SUM(tmp2)
```

# Computational Performance



- For the T90, V2 is the better performing code
- When targeting MSPs, entire code must be examined as in V3 to reduce code with serial dependencies

# LS-DYNA

General purpose nonlinear finite element program

- Heavily used by automotive industry and military
- Explicit solver

Code characteristics

- Version: 950d
- Date: 03/23/2000
- Shared Memory Parallel
- Platform : CRAY SV1 or J90 System

Model

- LS-Taurus 100% frontal barrier crash
- ~29,000 elements
- General benchmark model, not optimized for a specific platform

# Configurations Examined

**Looked at 1,2, 4, 8, 12 & 16 processors**

## Non-dedicated

executed with standard call, i.e.

```
lsdyna -v950d I=input ...
```

## Dedicated

<u>2 processor run</u>

```
MP_HOLDTIME=3000000000
```

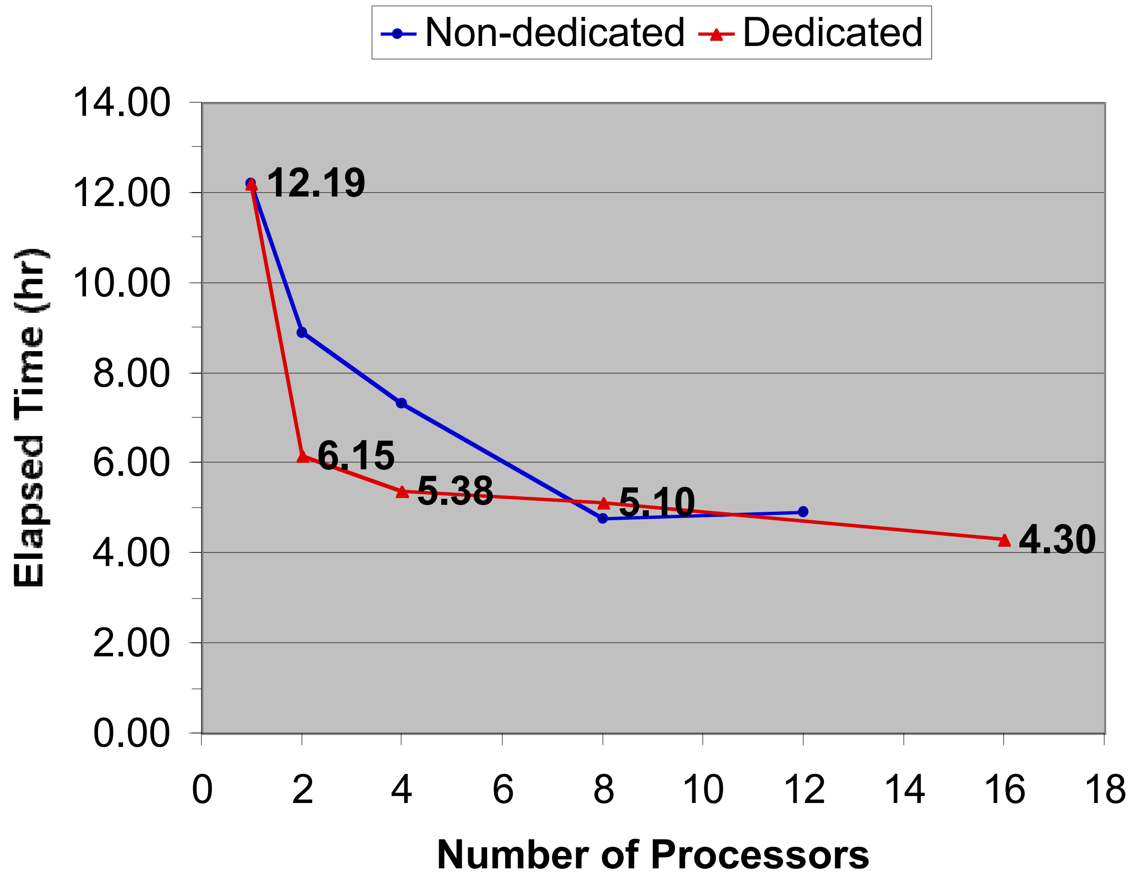
<u>4 and 8 processor runs</u>

```
/etc/cpu -a 1 lsdyna -v950d I=input
```

<u>16 processor run</u>

```
MP_DEDICATED=1
All user jobs checkpointed
```

# Elapsed Time



# LS-DYNA Observations

- Parallel performance sensitive to conflicts from other jobs
- Elapsed time relative to single process T90 run:
  - 2 processors (dedicated) runs ~20% longer
  - 4 processors (dedicated MSP) runs ~5% longer
  - 8 processors (dedicated MSP) approximately equal run times
- Elapsed times within 5% with 16 times the memory
  - Less waiting in the batch queue for memory resources
  - Ability to analyze larger models
- Recommend running with 2 or 4 processes *dedicated*.

2 processes

```
MP_HOLDTIME=large value
```

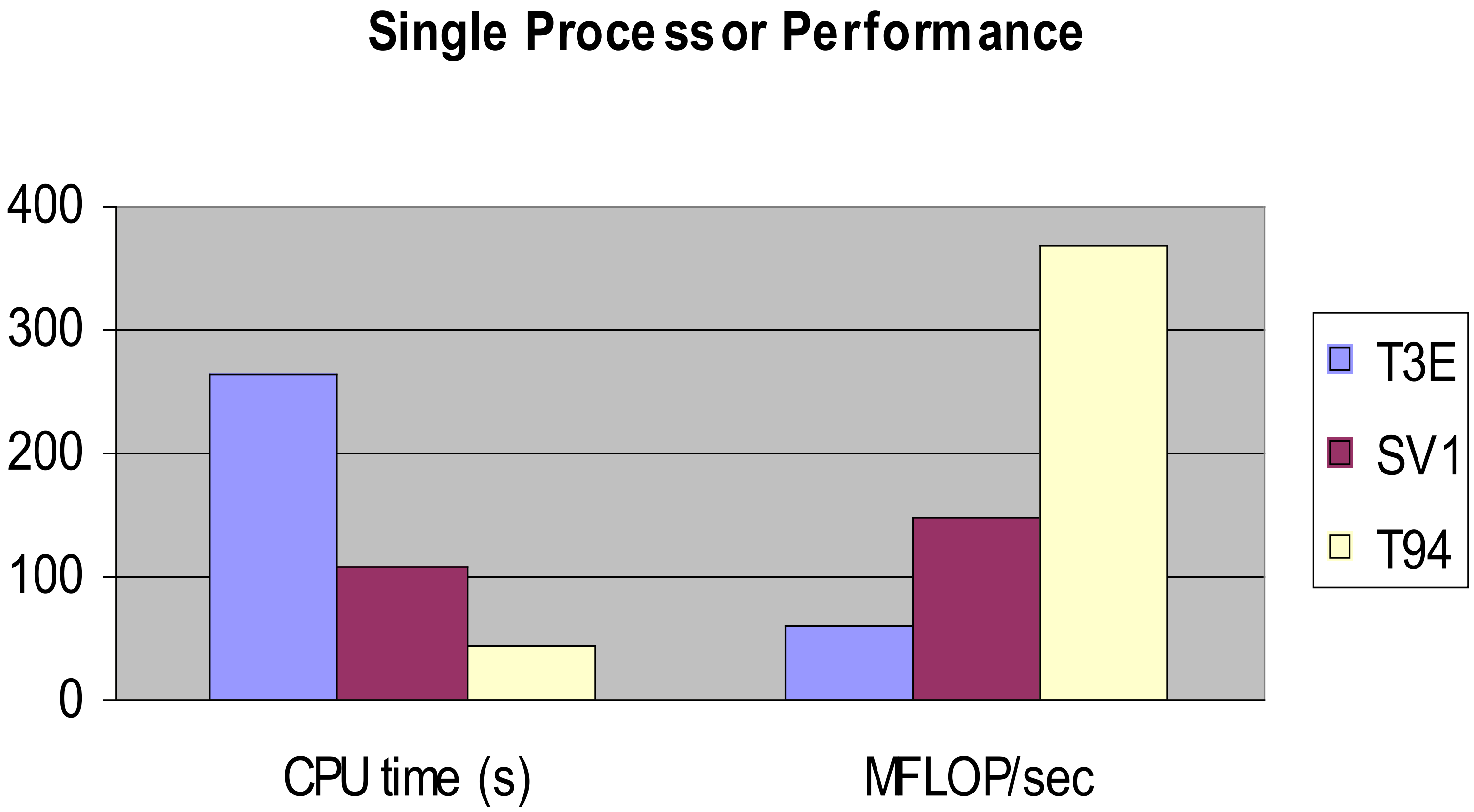
4 processes

```
NCPUS=4
/etc/cpu -a 1 lsdyna I=input …
```

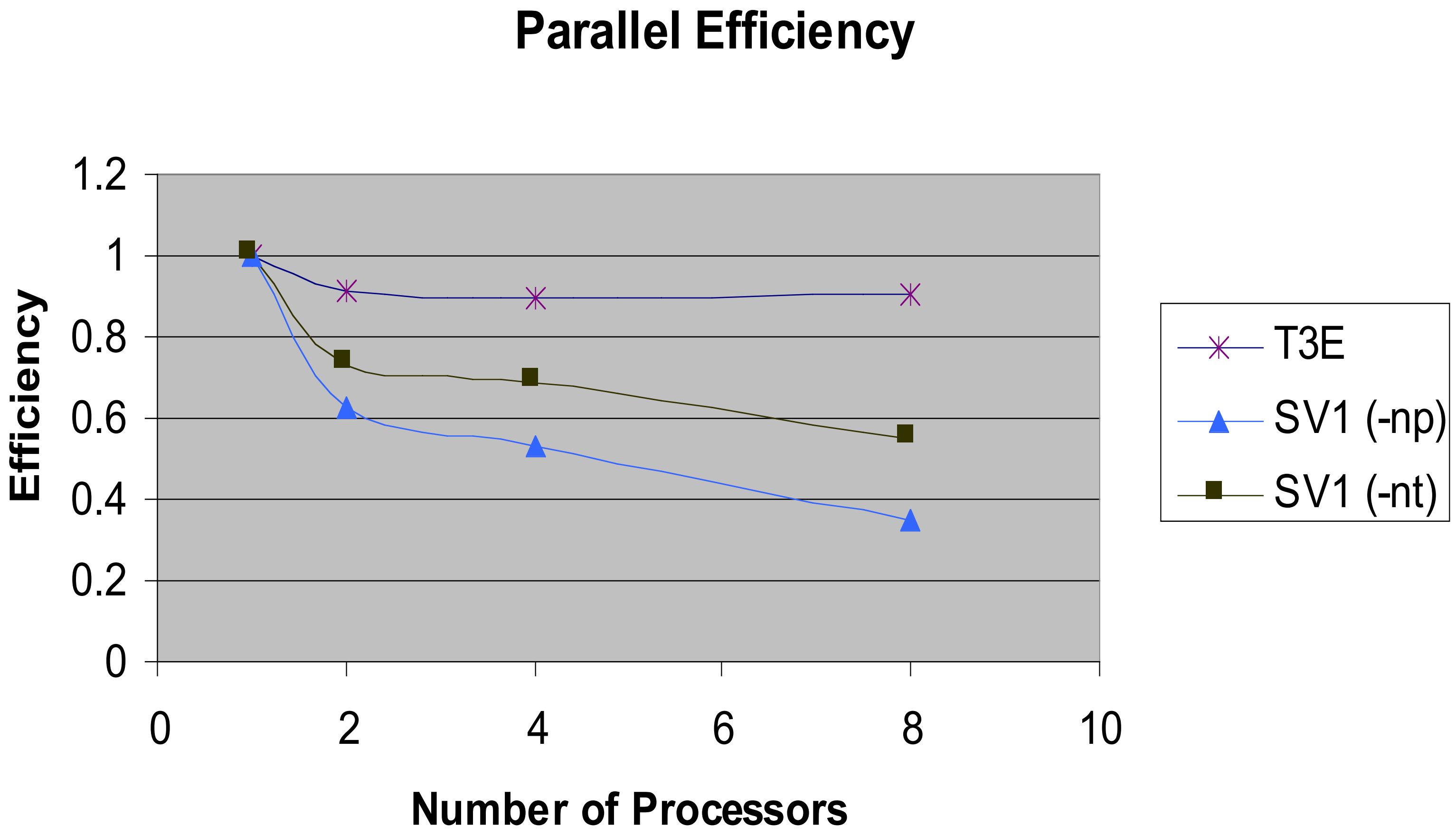
# MPI Performance: QCDMPI

- QCDMPI is a public-domain code for calculations in lattice Quantum Chromodynamics
- Modest size
- Reasonably cache friendly (developed for MPP)
- Scales nicely on a T3E
  - At OSC: 300MHz 21164, 128 processors
- Instrumented to provide both calculational performance and communications bandwidth data
- No special tuning performed; only compiler options
- Examine SV1 scalability, and compare TCP versus shared memory communication
  - `mpirun -np` **vs.** `mpirun -nt`
- **CF90 version:** `3.4.0.1.0`

# QCDMPI


Single Processor Performance
400
300
200
100
0
CPU time (s)
MFLOP/sec
T3E
SV1
T94

# QCDMPI


Parallel Efficiency
Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
Number of Processors
T3E
SV1 (-np)
SV1 (-nt)

# Lessons Learned

- MSP performance is often as good as or better than the T90
- MSP execution can approach factor-of-four speedup over SSP
  - Even more in some cases, due to larger effective cache
- Can be 30% - 200% better than “naïve” autotasking over four SSPs
  - Tighter hardware integration
  - Less contention for memory bandwidth
- Single-SSP loop nest optimizations generally seem to be best
- Must think about the cache!
- Eliminate conditions that (currently) inhibit streaming, e.g.
  - Data dependencies
  - I/O
  - Reduction operations

# Conclusions

- The SV1 offers very good performance for vector applications
- Still very much a vector machine
- New optimization issues naturally center on
  - Cache usage
  - Streaming
- As expected, code performance closely tracks memory throughput
- Future PE releases should enhance performance of the MSPs
  - Improved loop restructuring
  - Reduction of synchronization overhead
  - Multi-streaming of constructs in addition to loops
  - Additional directives, etc., to give programmers more flexibility in optimization
- We eagerly await these developments, not to mention the SV2!

# Additional Information

**Authors:**

Mr. Jim Giuliani jimg@osc.edu

Dr. Dave Robertson dgr@osc.edu

**Presentation:**

http://www.osc.edu/~jimg/Documents/SV1preso.ps

**Additional material:**

"Performance Tuning for the Cray SV1"

Presented at spring 2000 CUG Conference

http://www.osc.edu/~jimg/Documents/SV1performance.ps

http://www.cugoffice.org