# Gamess-UK Perfomance measurements on a cluster of RISC/6000

***Giuseppe Vitillaro***

Scientific and Technical

System Solutions

IBM Semea

VNET 75819593@ITHVM06

INTERNET peppe@ipgaix.unipg.it

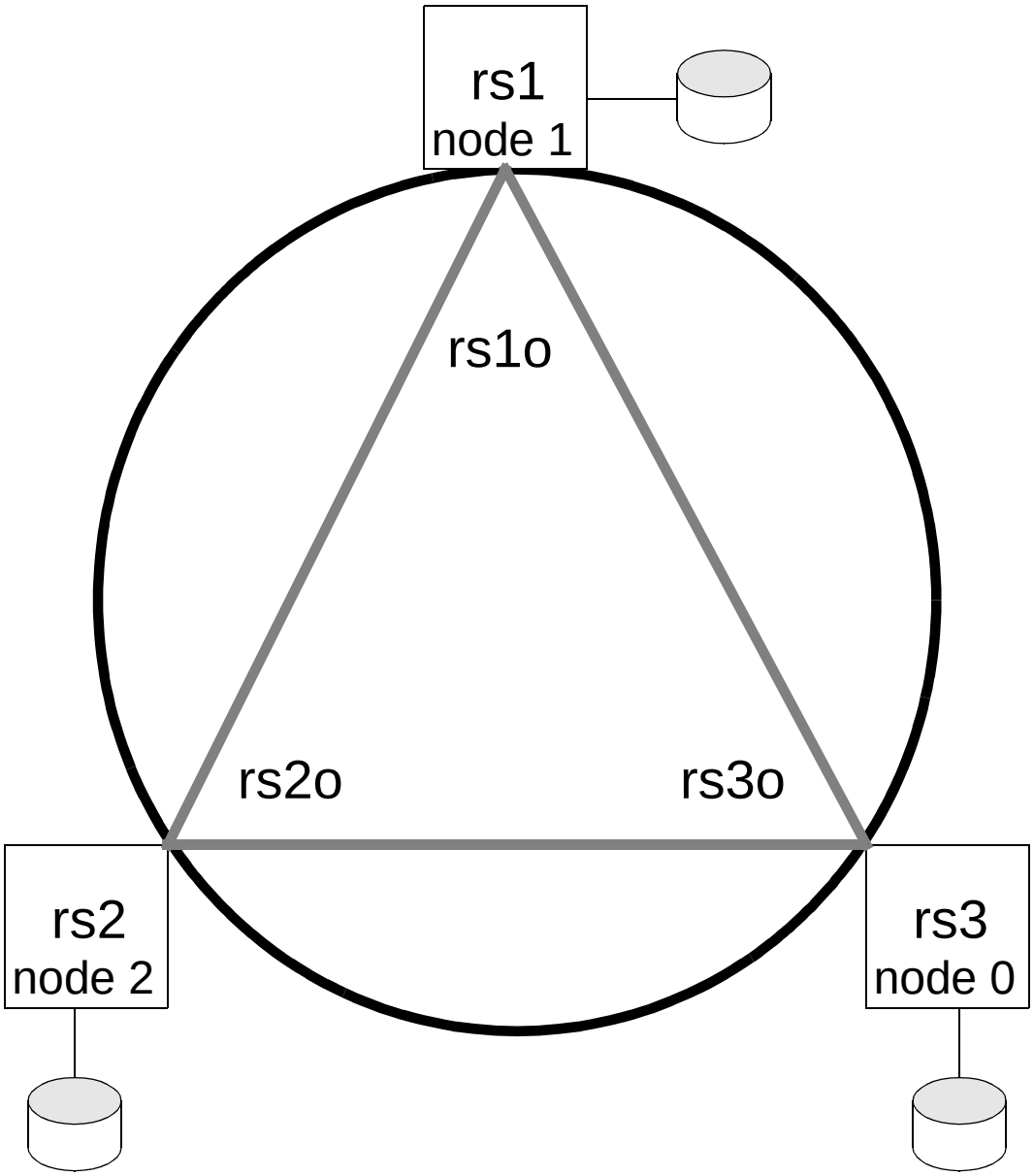
BITNET VITILLAR@BITNET

September 23, 1993

# Hardware Environment

- The cluster is composed of three IBM RISC/ 6000 Model 550:
  - **rs1** : 448 Mb of RAM
  - **rs2** : 128 Mb of RAM
  - **rs3** : 256 Mb of RAM
- They are connected in two ways:
  - Token Ring 16 Mbits/sec
  - Point to Point SOCC (Serial Optical Channel Converter) 220 Mbits/sec

# Hardware Environment




Token Ring 16 Mbit/sec

SOCC 220 Mbit/sec

# Software Environment

- All the nodes run under AIX/6000 3.2.3e
- All the nodes share the filesystems using NFS on both SOCC and TOKEN RING networks
- A modified SOCC device driver (provided by IBM ECSEC) is installed on all the nodes
- The Standard PVM 2.4.1 is installed on all the nodes
- The Special PVMe (provided by IBM ECSEC) is installed on all the nodes

# Benchmarks Environment

- Quantum Chemistry package GAMESS-UK 1992
- The parallelization of GAMESS-UK 1992 was done using the package TCGMSG 4.0 working on top of TCP/IP on both Token Ring and SOCC networks
- All the software was recompiled on the cluster using:
  - IBM AIX XL FORTRAN Compiler/6000 2.3.0
  - IBM AIX XL C Compiler/6000 1.2.1
- All the software was optimized

# Benchmarks Environment

- GAMESS-UK 1992 uses TCGMSG 4.0 to run in parallel
- TCGMSG 4.0 implements a message passing model over the standard TCP sockets
- The program uses the TCGMSG primitives to implement a SPMD (Single Process Multiple Data) model: one instance of the program runs on all the nodes doing the same computation on different data
- The parallel version of the program is started from a single node (node 0), using the driver program **parallel** that reads a configuration file and starts (using rsh) GAMESS on all the nodes

# Benchmarks Execution

- Serial Run:
  - node rs1
  - data on the local file system
- Parallel Runs:
  - 2 nodes

    [rs3, rs1] token ring

    [rs3o, rs1o] SOCC
  - 3 nodes

    [rs3, rs1, rs2] token ring

    [rs3o, rs1o, rs2o] SOCC
  - Node 0 (where the driver program runs) was always the machine rs3

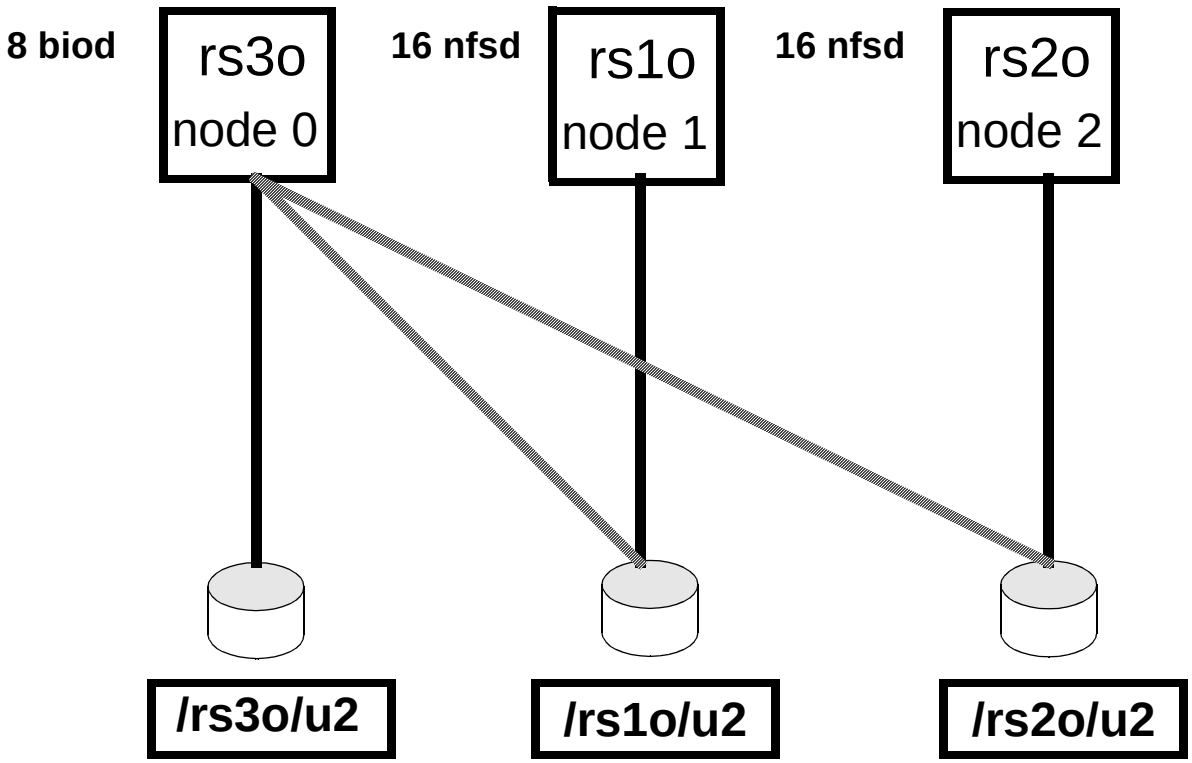
# Benchmarks Execution

- Each instance of GAMESS has the working directory on a file system that is local to the node where the instance runs
- The data on the working directory of the other nodes are accessed via NFS
- Through NFS mounts non local data are accessed over the same network (Token Ring or SOCC) where GAMESS exchanges TCGMSG messages
- The sequence number of the nodes was always

| | |
|---|---|
| rs3/rs3o | 0 |
| rs1/rs1o | 1 |
| rs2/rs2o | 2 |

# Benchmarks Execution

## Example: TCGMSG on SOCC
## File systems view for node rs3o




**Local File System**

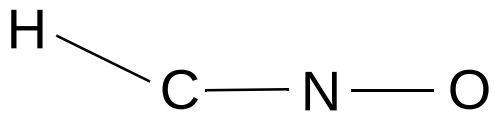
**NFS Mount via SOCC**

# Benchmarks Description

- Test case used for benchmarks:

  HCNO  Basis set: **TZVP**

  Symmetry $C_s$ :




- Timing data was collected for the following tasks:
  - AO Integral Evaluation (INT)
  - SCF
  - Integral Transformation (TRANSF)
  - Total for the previous tasks (INT+SCF+TRANSF)
  - Direct-ci (CI)
  - Total (INT+SCF+TRANSF+CI)
- A script running on rs1 started all the jobs

# Benchmarks Description

- The «test case» is described in the following GAMESS input file:

```
time 180
memory 2000000
file ed2 ed2
file ed3 ed3
file ed4 ed4
file ed5 ed5
file ed6 ed6
title
hcno tzv
super off nosym
geometry
0.0  0.347836405 -1.972678252 1 h
0.0  0.0  0.0    6 c
0.0  0.0  2.1864  7 n
0.0  0.0  4.4446  8 o
end
basis
tzv n
d n
1.0 1.2
d n
1.0 0.4
tzv o
d o
1.0 1.2
d o
1.0 0.4
f o
1.0 1.0
tzv c
d c
1.0 1.2
d c
1.0 0.4
tzv h
p h
1.0 1.2
p h
1.0 0.4
end
runtype ci
active
1 to 97 end
direct 22 11 86
enter 1
```

# Benchmark Results

- We will present the benchmark results for each network (TOKEN RING and SOCC)

- $T_1$ : elapsed time serial

  $T_N$ : elapsed time parallel, N nodes

  Speedup : $U = T_1/T_N$

  Efficiency : $e = U/N$

# TOKEN RING Results

**CPU Time (seconds)**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 179.60 | 91.70 | 61.72 |
| scf | 79.38 | 46.10 | 33.37 |
| transf | 448.25 | 262.95 | 184.18 |
| total | 707.23 | 400.76 | 279.27 |
| ci | 556.74 | 395.39 | 411.52 |
| total | 1263.98 | 796.15 | 690.79 |

**Wall Time (seconds)**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 190.56 | 96.50 | 65.98 |
| scf | 97.07 | 60.64 | 49.90 |
| transf | 603.75 | 463.91 | 679.16 |
| total | 891.39 | 621.05 | 795.05 |
| ci | 713.84 | 844.34 | 1144.27 |
| total | 1605.23 | 1465.39 | 1939.32 |

# TOKEN RING Results

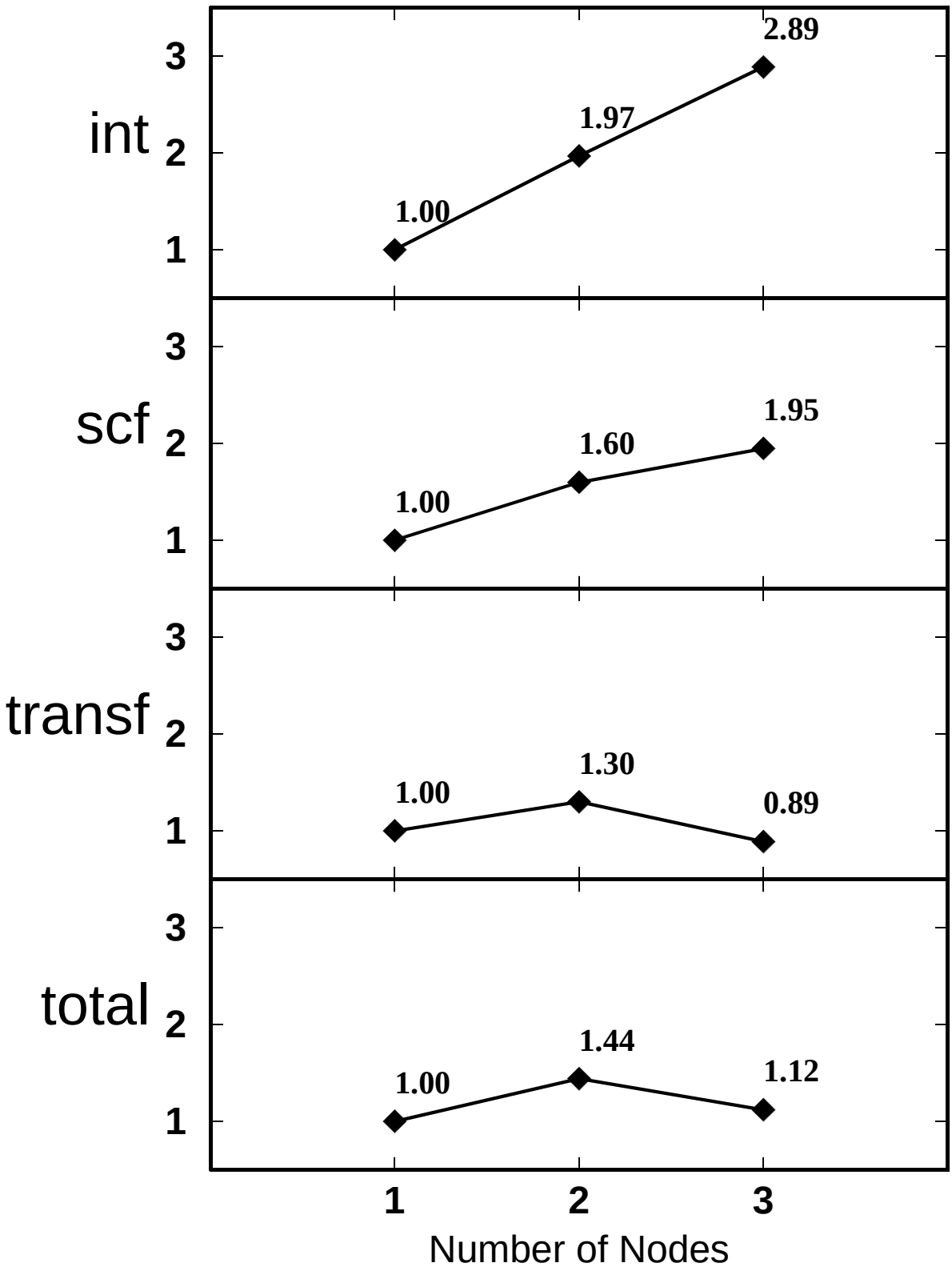
**Wall Speedup**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 1.00 | 1.97 | 2.89 |
| scf | 1.00 | 1.60 | 1.95 |
| transf | 1.00 | 1.30 | 0.89 |
| total | 1.00 | 1.44 | 1.12 |
| ci | 1.00 | 0.85 | 0.62 |
| total | 1.00 | 1.10 | 0.83 |

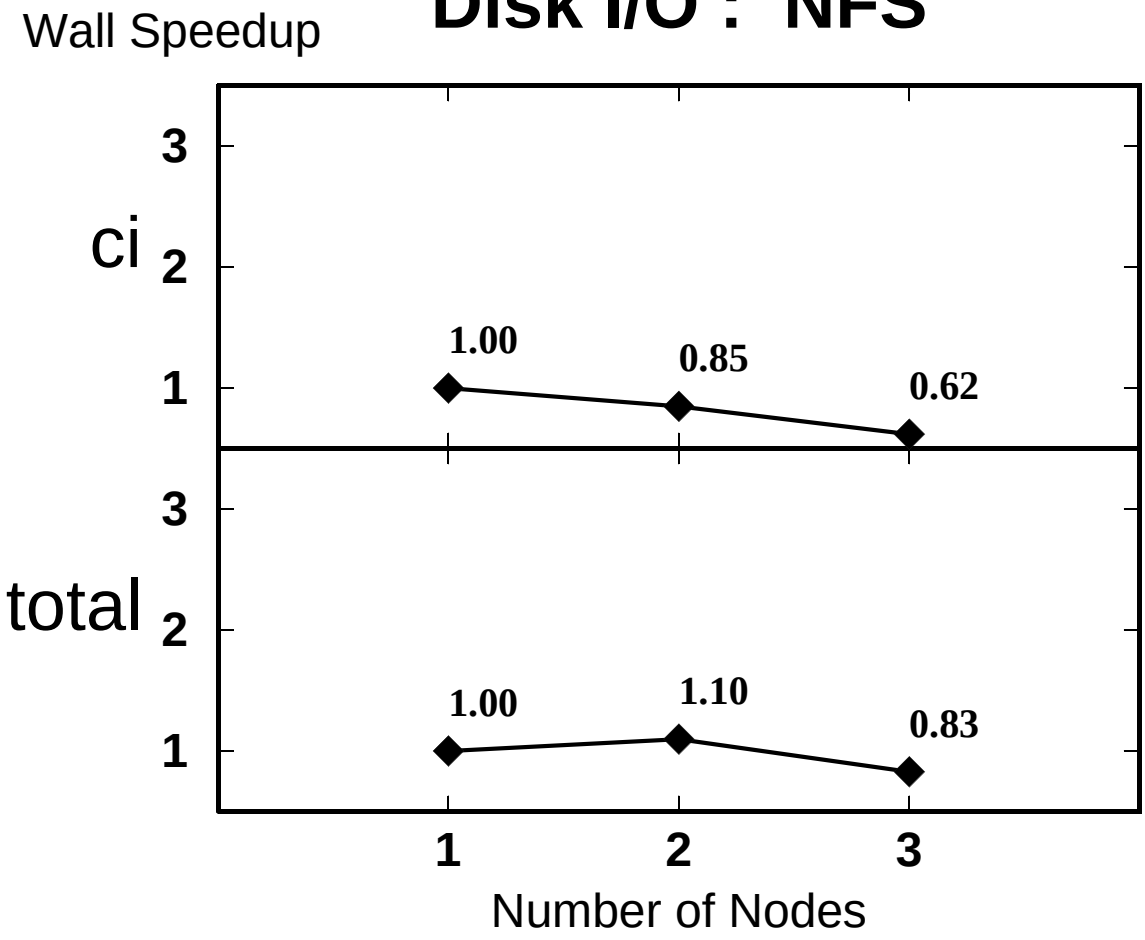
**Efficiency**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 1.00 | 0.99 | 0.96 |
| scf | 1.00 | 0.80 | 0.65 |
| transf | 1.00 | 0.65 | 0.30 |
| total | 1.00 | 0.72 | 0.37 |
| ci | 1.00 | 0.42 | 0.21 |
| total | 1.00 | 0.55 | 0.28 |

# Connection: TOKEN RING
# DISK I/O : NFS



# Connection: TOKEN RING
# Disk I/O : NFS



# SOCC Results

**CPU Time (seconds)**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 179.60 | 92.05 | 61.69 |
| scf | 79.38 | 46.01 | 33.29 |
| transf | 448.25 | 257.91 | 179.15 |
| total | 707.23 | 395.97 | 274.13 |
| ci | 556.74 | 382.20 | 394.26 |
| total | 1263.98 | 778.17 | 668.39 |

**Wall Time (seconds)**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 190.56 | 96.50 | 64.94 |
| scf | 97.07 | 59.17 | 42.96 |
| transf | 603.75 | 347.00 | 284.22 |
| total | 891.39 | 502.67 | 392.11 |
| ci | 713.84 | 685.43 | 684.92 |
| total | 1605.23 | 1188.10 | 1077.03 |

# SOCC Results

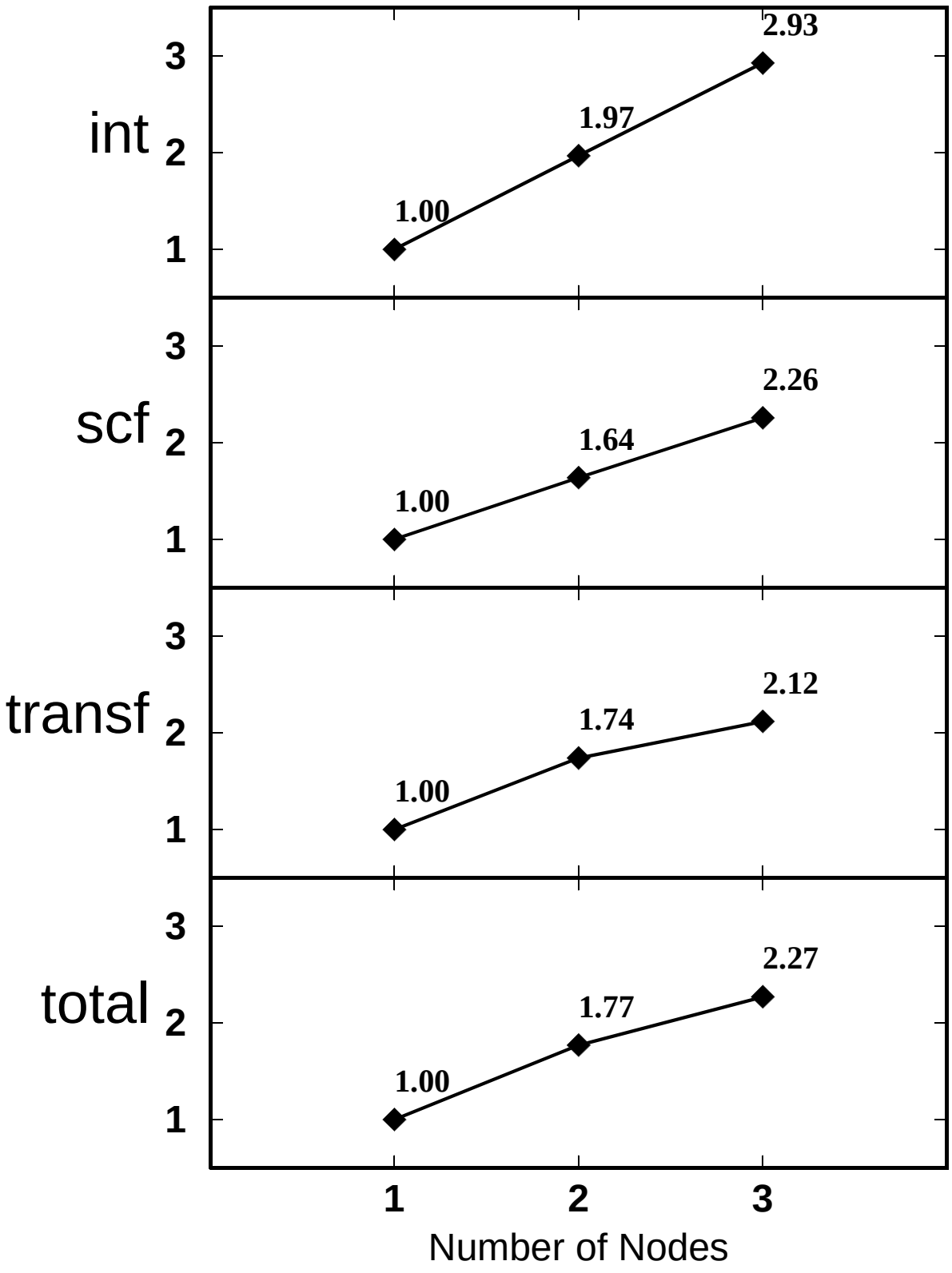
**Wall Speedup**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 1.00 | 1.97 | 2.93 |
| scf | 1.00 | 1.64 | 2.26 |
| transf | 1.00 | 1.74 | 2.12 |
| total | 1.00 | 1.77 | 2.27 |
| ci | 1.00 | 1.04 | 1.04 |
| total | 1.00 | 1.35 | 1.49 |

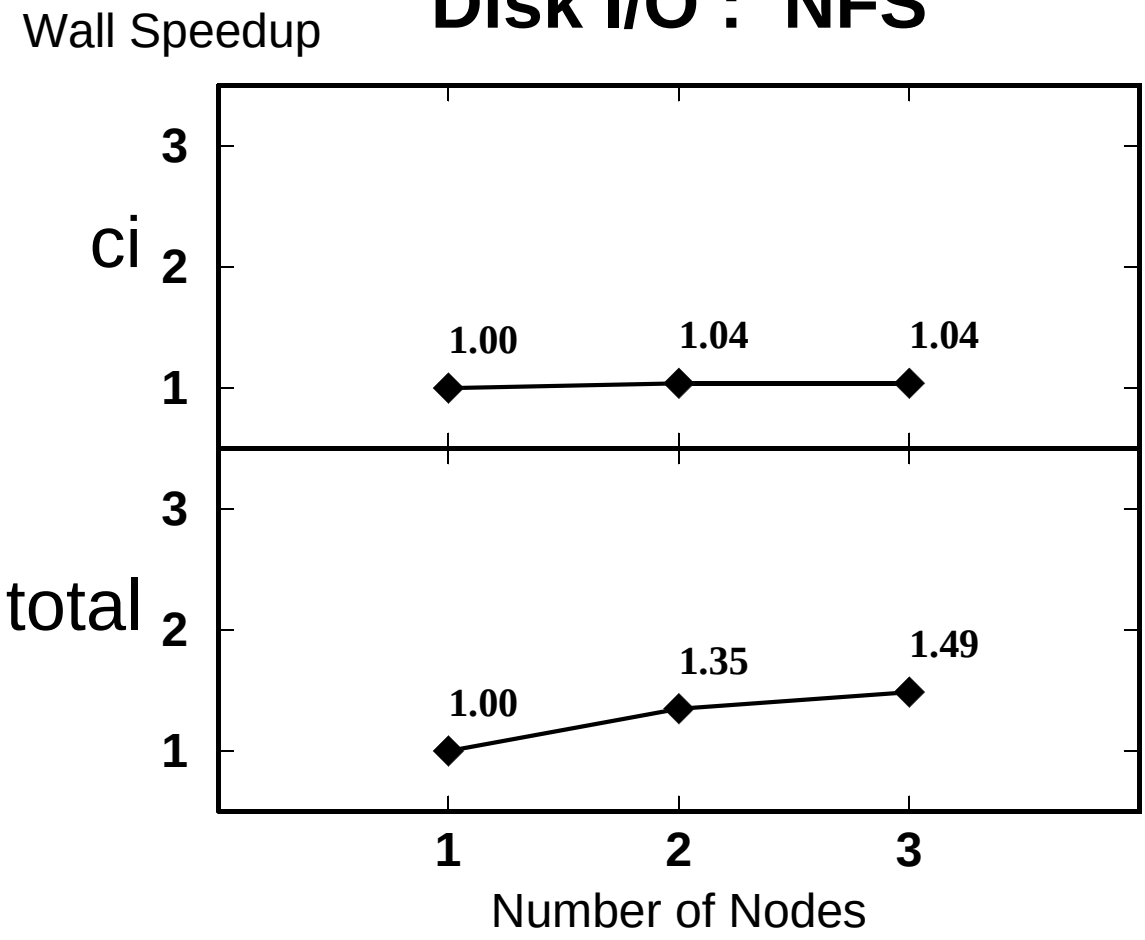
**Efficiency**

| Task | 1 | 2 | 3 |
|---|---|---|---|
| int | 1.00 | 0.99 | 0.98 |
| scf | 1.00 | 0.82 | 0.75 |
| transf | 1.00 | 0.87 | 0.71 |
| total | 1.00 | 0.89 | 0.76 |
| ci | 1.00 | 0.52 | 0.35 |
| total | 1.00 | 0.68 | 0.50 |

# Connection: SOCC
# DISK I/O : NFS



# Connection: SOCC
# Disk I/O : NFS



# NFS role

- NFS plays a heavy role on the performance of parallel GAMESS:
  - large files are involved
  - network I/O is expensive in terms of time
  - some GAMESS tasks cannot have all the files present on a local file system
  - NFS use TCP/IP (through RPC calls) and may not get full advantage of SOCC bandwidth
  - NFS use its own buffering scheme that may not be the best for GAMESS network I/O performance
- **Problem**: try to find an alternative to NFS for network I/O that may get more advantage from SOCC bandwidth

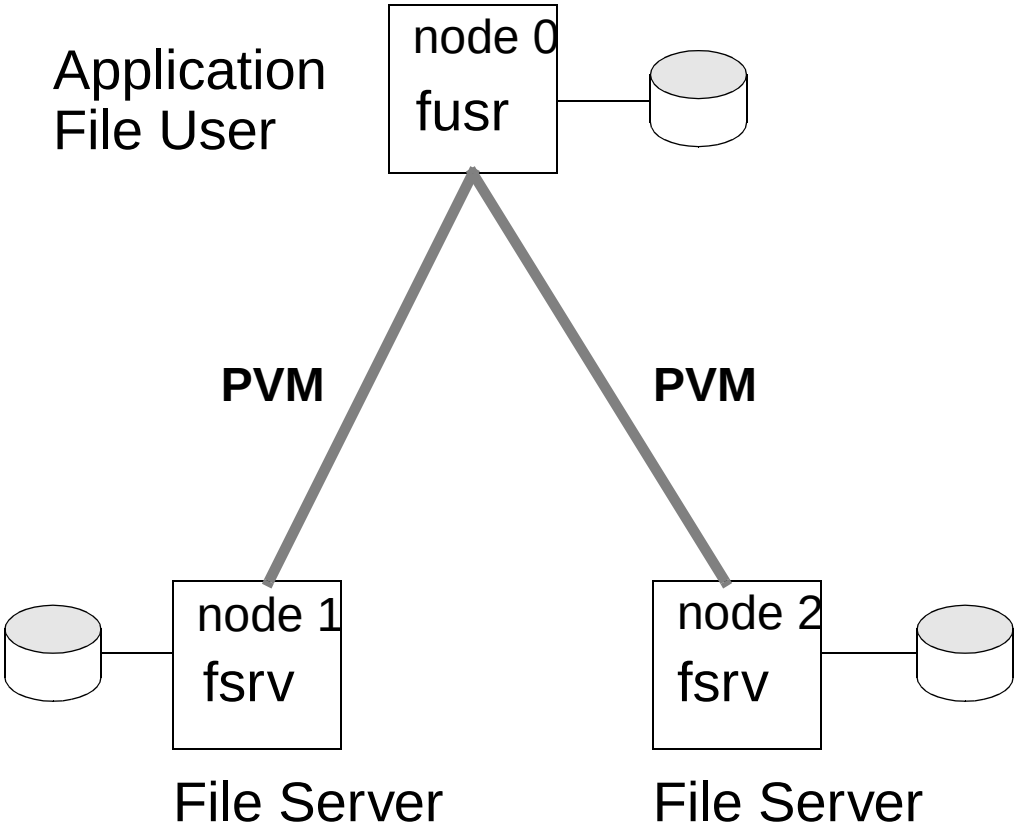
# PVM role

- We chose to use **PVM** to replace NFS for a particular GAMESS task: **INTEGRAL TRANSFORMATION**
- We used PVM in a rather simple way to implement a **Client-Server** model for network I/O
- We wrote a library of primitives that follow the UNIX scheme of **[open read write lseek close]** for **Unbuffered Network I/O**
- This library completely replaces NFS and uses only PVM primitives for data transfer
- The **PVM FileServer** library is written in C Language
- **PVM Standard** or **IBM ECSEC PVMe** may be used

# PVM FileServer Library

- The library defines a set of functions that let an application access local or remote files
- Local Files are accessed using direct standard unbuffered UNIX calls
- Calls that involve remote files are redirected by a local **FileUser** interface (**fusr**) to a PVM spawned **FileServer** (**fsrv**) that runs on the remote system.
- The library defines functions to:
  - Let the program enroll in and leave PVM as a **fusr**
  - attach a **fsrv** on a remote machine
  - execute [open read write lseek close] on local or remote files

# PVM FileServer Library




**PVM FileServer Library C Interface**

```
typedef int NODE;

int  fsinit ( void );
int  fsend  ( void );

NODE fsattach ( const char *hostname );
int  fsdetach ( NODE node );

int  rchdir  ( NODE node, const char *path );
int  runlink ( NODE node, const char *path );

int  ropen  ( NODE node, const char *path, int oflag );
int  rclose ( int handle );
int  rread  ( int handle, char *buf, unsigned int nbytes );
int  rwrite ( int handle, char *buf, unsigned int nbytes );
long rlseek ( int handle, long offset, int whence );

NODE rfnode ( int handle );

extern int fuerrno;
```

# Integral Transformation

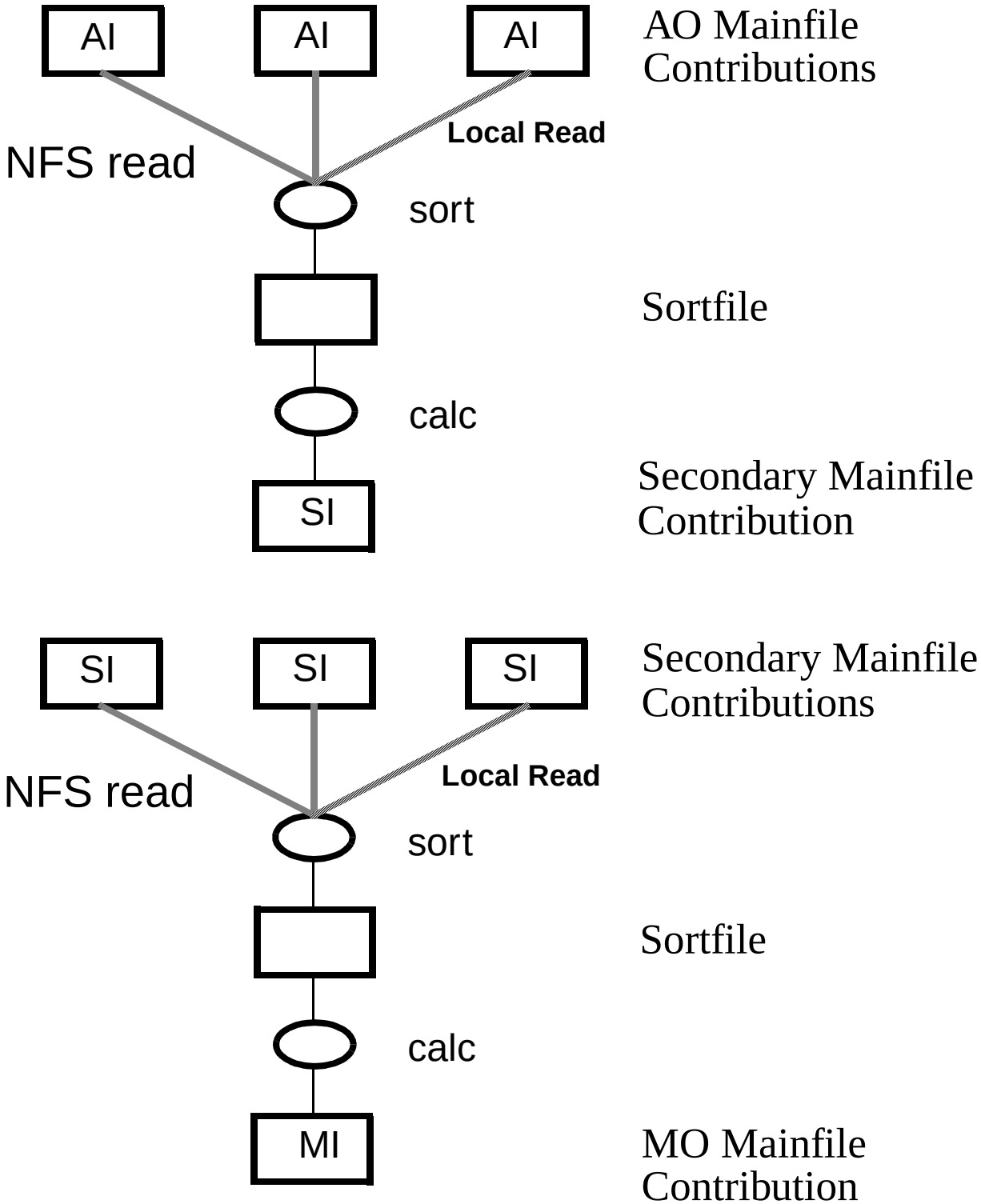
- **Integral Transformation** appears to be the task more suited to benchmark the PVM FileServer Library against NFS
- Integral Transformation in parallel GAMESS involves heavy network I/O through NFS.
  - The AO integral file («AO Mainfile») is split across nodes. Each node owns a local partial contribution.
  - During the integral sort phase each node reads its local mainfile and all the remote mainfiles. Each node select and sorts an ordered partial list of integrals. These are output to a local «Sortfile».
  - Each node processes its own sortfile to produce a local file of «Semitransformed Integrals» («Secondary Mainfile»). No communication is involved.
  - The process is repeated to produce local transformed integral file contributions from the «Secondary Mainfiles».

# Integral Transformation

- The files involved in the Integral Transformation are very large and two NFS read phases are involved in the process of transforming Atomic Integrals to Molecular Integrals

- A straightforward change in the Integral Transformation code was needed to use the PVM FileServer Library, instead of NFS

# Integral Transformation



# Transformation

- We have run benchmarks using:
  - PVM Standard 2.4.1 **«virtual circuits»** for Token Ring and SOCC networks

    this is the fastest way to use PVM Standard: it doesn't involve the pvm daemon for data transfer, it still uses TCP/IP sockets.

  - IBM ECSEC PVMe **«*IMCS* synchronous»** mode (mode 2) for SOCC network only

    this is the fastest way to use PVMe: it interfaces directly with the IMCS low-level SOCC protocol

# Transformation

- We will present the benchmarks results for the two different PVM we used and for Token Ring and SOCC; we tested the cases of 2 and 3 nodes
- We will compare the results obtained using NFS and PVM FileServer Library, under the same conditions
- Again we calculated «Wall Speedup» and «Efficiency» from execution elapsed time
- For the case of 1 node (serial version) «CPU Time» and «Wall Time» are always the same and refer to local I/O done through UNIX calls; they are presented to compare the results

# Transformation

## TOKEN RING Results

**CPU Time (seconds)**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 448.25 | 262.95 | 184.18 |
| PVM | 448.25 | 267.39 | 191.71 |

**Wall Time (seconds)**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 603.75 | 463.91 | 679.16 |
| PVM | 603.75 | 508.93 | 516.91 |

# Transformation

## TOKEN RING Results

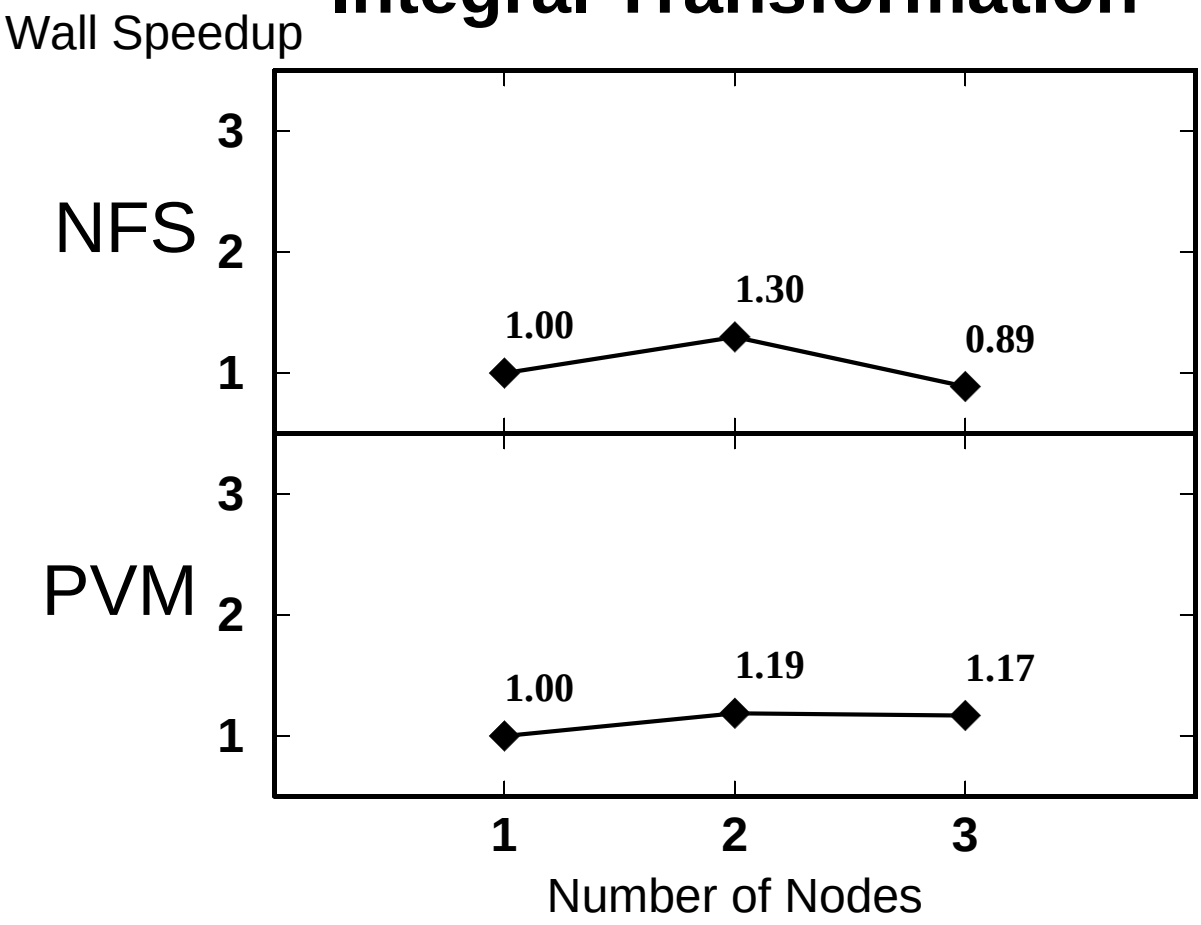
**Wall Speedup**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 1.00 | 1.30 | 0.89 |
| PVM | 1.00 | 1.19 | 1.17 |

**Efficiency**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 1.00 | 0.65 | 0.30 |
| PVM | 1.00 | 0.59 | 0.39 |

# Connection: TOKEN RING
# Integral Transformation



# Transformation

## SOCC Results

**CPU Time (seconds)**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 448.25 | 257.91 | 179.15 |
| PVM | 448.25 | 266.37 | 188.35 |
| PVMe | 448.25 | 256.91 | 182.05 |

**Wall Time (seconds)**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 603.75 | 347.00 | 284.22 |
| PVM | 603.75 | 444.46 | 405.91 |
| PVMe | 603.75 | 341.99 | 262.92 |

# Transformation

## SOCC Results

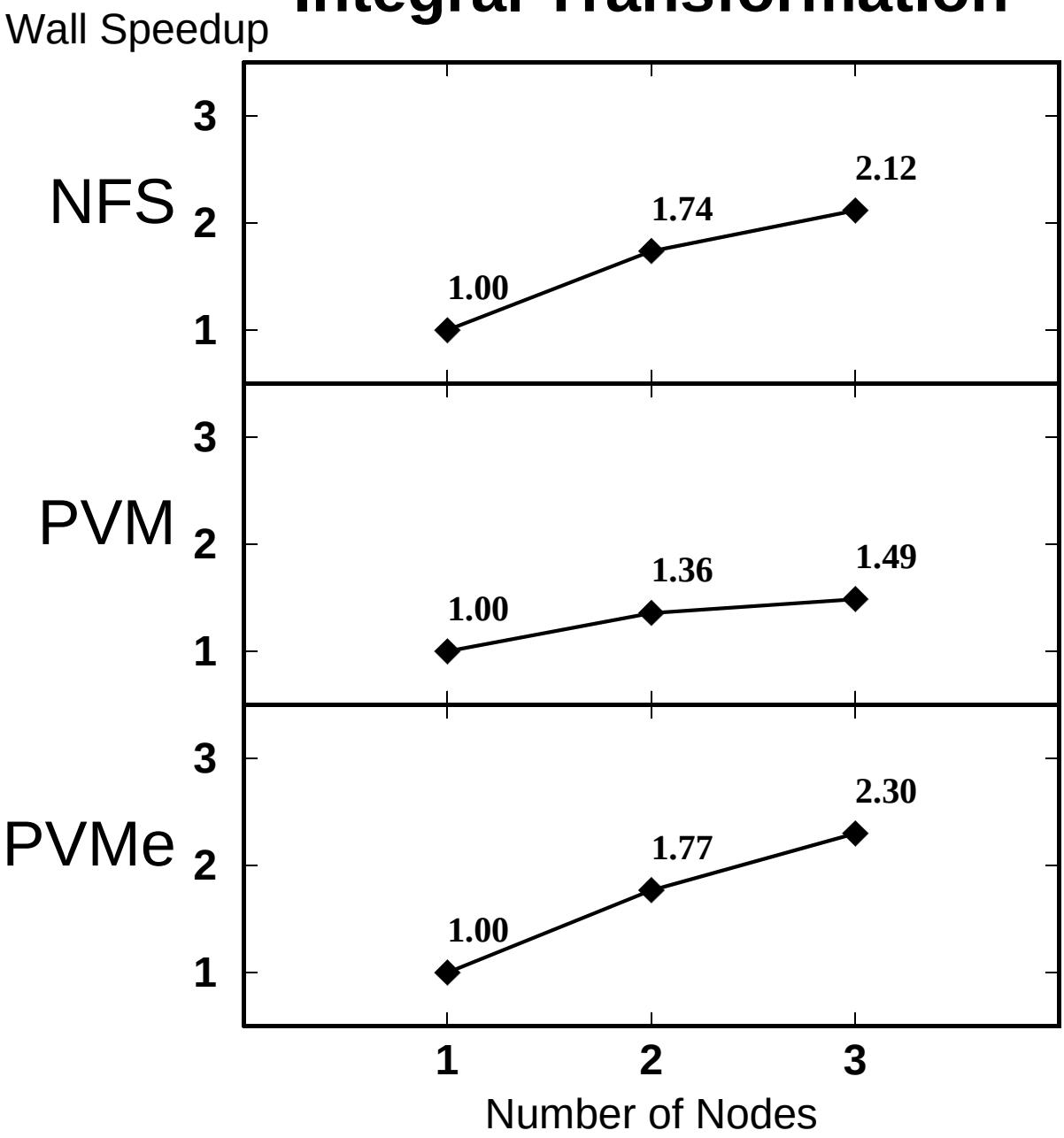
**Wall Speedup**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 1.00 | 1.74 | 2.12 |
| PVM | 1.00 | 1.36 | 1.49 |
| PVMe | 1.00 | 1.77 | 2.30 |

**Efficiency**

| I/O | 1 | 2 | 3 |
|---|---|---|---|
| NFS | 1.00 | 0.87 | 0.71 |
| PVM | 1.00 | 0.68 | 0.50 |
| PVMe | 1.00 | 0.88 | 0.77 |

# Connection: SOCC Integral Transformation



# Conclusions

## Scf

**Wall Speedup**

| Connection | 2 | 3 |
|---|---|---|
| Token Ring | 1.60 | 1.95 |
| SOCC | 1.64 | 2.26 |

- Significant increase in speedup when going from 2 to 3 nodes under SOCC compared to Token Ring.

- Reflects gathering of Fock matrix over a bandwidth which scales (SOCC) compared with a constant bandwidth (Token Ring).

# Conclusions

## Integral Transformation

**Wall Speedup**

| Connection | 2 | 3 |
|---|---|---|
| Token Ring | 1.30 | 0.89 |
| SOCC NFS | 1.74 | 2.12 |
| SOCC PVMe | 1.77 | 2.30 |

- The bad Token Ring perfomance seems again related with a constant and narrow bandwidth.
- The SOCC PVMe case is the best perfoming: a better use of the SOCC bandwidth is achieved with PVMe.

# Conclusions

## Direct-CI

**Wall Speedup**

| Connection | 2 | 3 |
|---|---|---|
| Token Ring | 0.85 | 0.62 |
| SOCC | 1.04 | 1.04 |

- Direct-CI is the worst perfoming GAMESS task.
- Investigation has to be done on the way Direct-CI is parallelized.
- Perfomance on SOCC are better than on Token Ring.

# Conclusions

## INT+SCF+TRANSF

**Wall Speedup**

| Connection | 2 | 3 |
|---|---|---|
| Token Ring | 1.44 | 1.12 |
| SOCC NFS | 1.77 | 2.27 |
| SOCC PVMe | 1.79 | 2.40 |

## INT+SCF+TRANSF+CI

**Wall Speedup**

| Connection | 2 | 3 |
|---|---|---|
| Token Ring | 1.10 | 0.83 |
| SOCC NFS | 1.35 | 1.49 |
| SOCC PVMe | 1.35 | 1.52 |