Using a Cluster as a Memory Resource: A Fast and Large Virtual Memory on MPI

DLM: Distributed Large Memory System

IEEE Cluster 2009, New Orleans, Aug.31- Sep.4

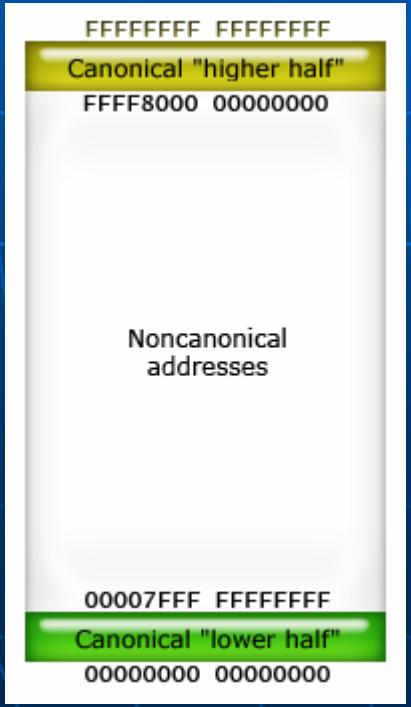
Hiroko Midorikawa, Kazuhiro Saito
Seikei University, Tokyo, Japan
Mitsuhisa Sato, Taisuke Boku
University of Tsukuba, Ibaraki, Japan

Introduction

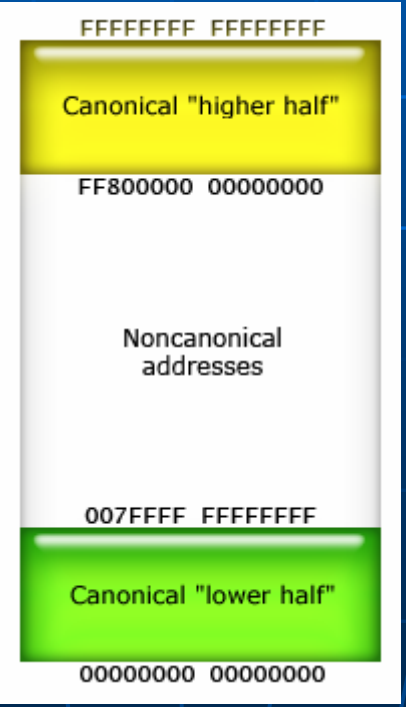
- Network speed is faster than I/O to local hard disks.
- 64bit OS gives us ample memory address space.

  Current x86_64(AMD64,Intel64) provides 256TB(2^{48})

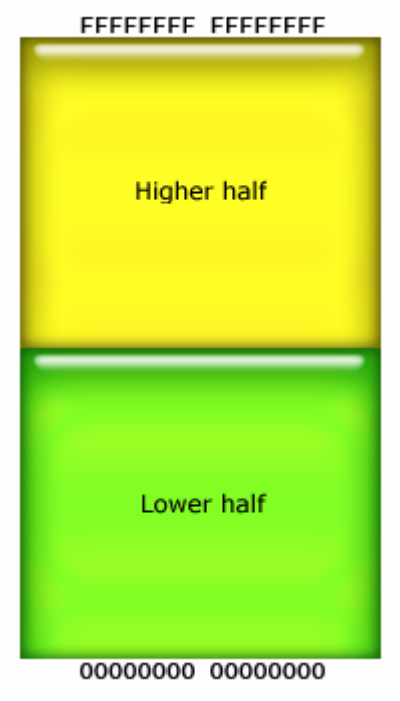
48bit adrs (256tebibytes)



56bit adrs (64pebibytes)



64bit adrs (16exbibytes)



Virtual large memory system using remote memory

- Kernel-level implementation

  Typical scheme: Replace a OS swap device driver with a newly designed device driver accessing remote memory, incorporated in OS page swapping

  - ○ Complete user transparency
  - × Low portability, low availability

- User-level implementation

  Typical scheme: use access detection by sigsegv signal and page swapping in user level software

  - ○ High portability, Easy to use for general users
  - × Low user transparency

    usually requires program modifications

Virtual large memory system using remote memory

How much of performance difference
does it deliver
in user-level implementation
and
kernel-level implementation?

Findings of DLM (DLM-socket), user-level software using conventional TCP socket on a10Gbps Ethernet cluster (cluster2008)

- Found that the DLM using only user-level software with general TCP socket achieves better performance than other low-level remote paging schemes, which typically use a block device for paging, RDMA, fast low-level communication protocols, kernel modification, and/or specially-designed NICs.
- Found that the stable behavior and higher performance of DLM benefit from DLM's independence of the conventional kernel swapping system.

  (DLM outperforms a conventional OS swapping to local hard disk even in a 1Gbps Ethernet cluster)
- Found that use of a larger page size than 64KB is more effective to achieve higher performance in application programs.

Findings of DLM (DLM-socket), user-level software using conventional TCP socket on a10Gbps Ethernet cluster (cluster2008)

User-level implementation achieves comparable or better performance compared to kernel-level implementation.

no drawback in performance

Programming interface

- Kernel level implementation

  Perfect Transparency, Low Portability

- User level implementation

  High Portability, Low Transparency

- JumboMem (cluster2007) dynamic linkable shared library,

  Use LD_PRELOAD to replace existing memory-related functions

  ○ no modification even for binary programs.

  × applicable to dynamic data allocation only, e.g. malloc(),
  NOT to static data declarations, e.g. large data arrays.

  × undesirable side effect for system software use.

  All malloc data have a possibility to swap out to remote memory.

- DLM (cluster2008) limits language: C,
  compiler and/or library supports to easy translation

Motivation

Expand the benefits of user level implementation.

Exploit more wider availability of fast and large virtual memory in a cluster for general users.

DLM must be

- Available in open clusters managed by batch queuing systems, as well as dedicated interactive cluster systems
- Independent from communication protocols and fabrics in clusters to use
- For general users who are unfamiliar with parallel programming to use a cluster, who have neither their dedicated clusters nor high-end memory-rich computers.

Goal

Using a Cluster as a Memory Resource

for sequential programs using large amount of memory

- Easy programming interface with high portability
- Runtime system which is applicable to a batch queuing system
- Multiple swap protocols according to user environments for wider availability

Overview of MPI-based DLM (DLM-MPI)

- MPI communications for all communications between a calculation process and memory server processes

  Independence from communication fabrics / protocols on lower layers

  higher-performance by MPI directly implemented on higher-speed network links

- SPMD-style Runtime system with init. function, dlm_startup()

  No automatic memory server process creation.

  Use MPI process on nodes allocated by a batch scheduler.

  - User program execution in cal process on rank0
  - Memory server processes on rank1-rankN

- Multiple swap protocols. DLM can be configured with an appropriate protocol according to MPI thread-support level in use

- Easy programming Interface.

  More simple DLM library for hand-translating,

  as well as DLM-MPI compiler for automatic translation

Example of DLM program for DLM-MPI compiler (matrix x vector multiplication : prog.c)

```
#include <dlm.h>
#define N 16384 // example: mem 2048MB + 32KB

dlm double a[N][N], x[N], y[N];  // DLM data

main(int argc, char *argv[])
{   int i,j;
    double temp;

    for(i = 0; i < N; i++)        //  Initialize array a
        for(j = 0; j <N; j++)     a[i][j] = i;

    for(i = 0; i < N; i++)    x[i] = i; // Initialize array x
        for(i = 0; i < N; i++){   // a[N][N]*x[N]=y[N]
            temp = 0;
            for(j = 0; j <N; j++)  temp += a[i][j]*x[j];
            y[i] = temp;
        }

    return 0;
}
```

dlm declaration.
DLM is applicable to both dynamic and static data declarations

An example of Compile Command
dlmcc prog.c -o prog

User can specified DLM data,
which are expandable to remote memory.
Unspecified data are guaranteed to be in local memory.
They are excluded from candidates for swap out data.

Add "dlm" to an original sequential program.

Example of DLM program for hand-translating

(prog.c)

```
#include <dlm.h>
#define N 16384 // example: mem 2048MB + 32KB
double (*a)[N];  //  a pointer for 2-D array      a[N][N]
double  *x, *y;   //  pointers for 1-D arrays   y[N],x[N]

main(int argc, char *argv[])
{   int i,j;
    double temp;
    dlm_startup (&argc, &argv);

    a = (double (*)[N]) dlm_alloc (  N * N * sizeof(double) );
    x = (double *) dlm_alloc ( N * sizeof(double) );
    y = (double *) dlm_alloc ( N * sizeof(double) );

    for(i = 0; i < N; i++)       //  Initialize array a
        for(j = 0; j <N; j++)    a[i][j] = i;

    for(i = 0; i < N; i++)    x[i] = i;     // Initialize array x
        for(i = 0; i < N; i++){    // a[N][N]*x[N]=y[N]
          temp = 0;
          for(j = 0; j <N; j++)  temp += a[i][j]*x[j];
          y[i] = temp;
        }
    dlm_shutdown ( );
    return 0;
}
```

An example of Compile Command

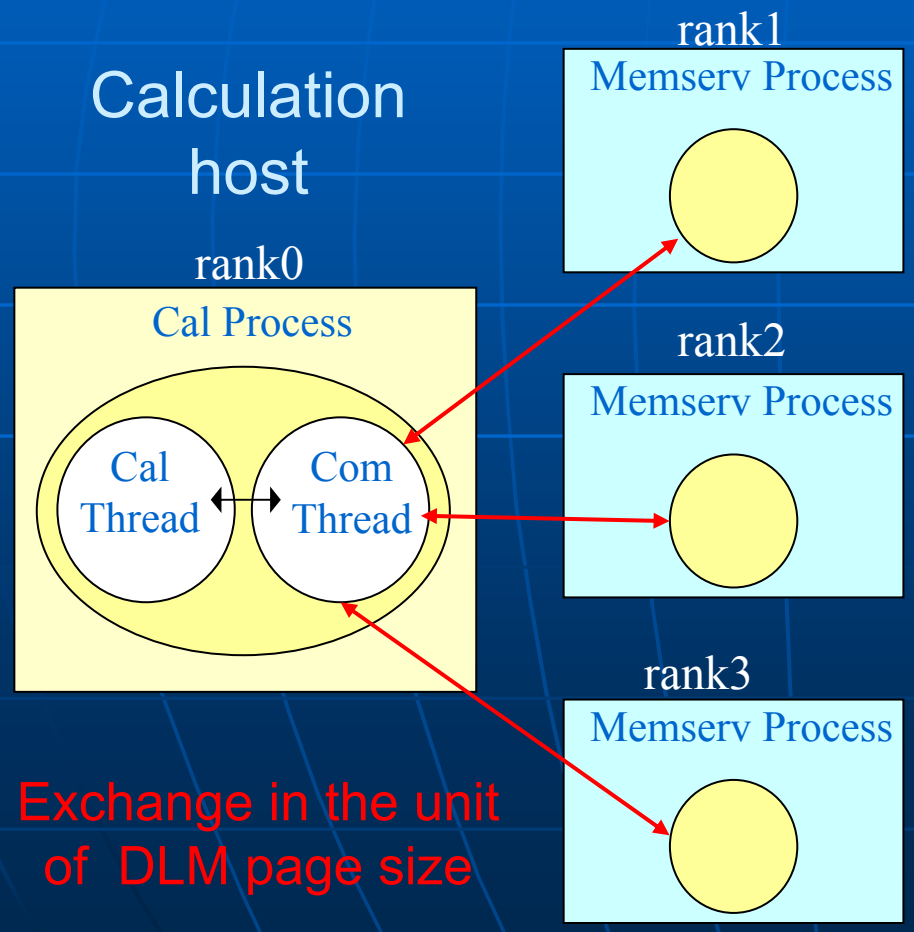
mpicc prog.c -o prog -ldlmmpi

dlm dynamic allocation

Core program codes need No modification

Still Sequential Program.
DLM is available with only library without special DLM compiler

Example of DLM Runtime System for MPI batch queuing system



Exchange in the unit of DLM page size

typical MPI batch shell script

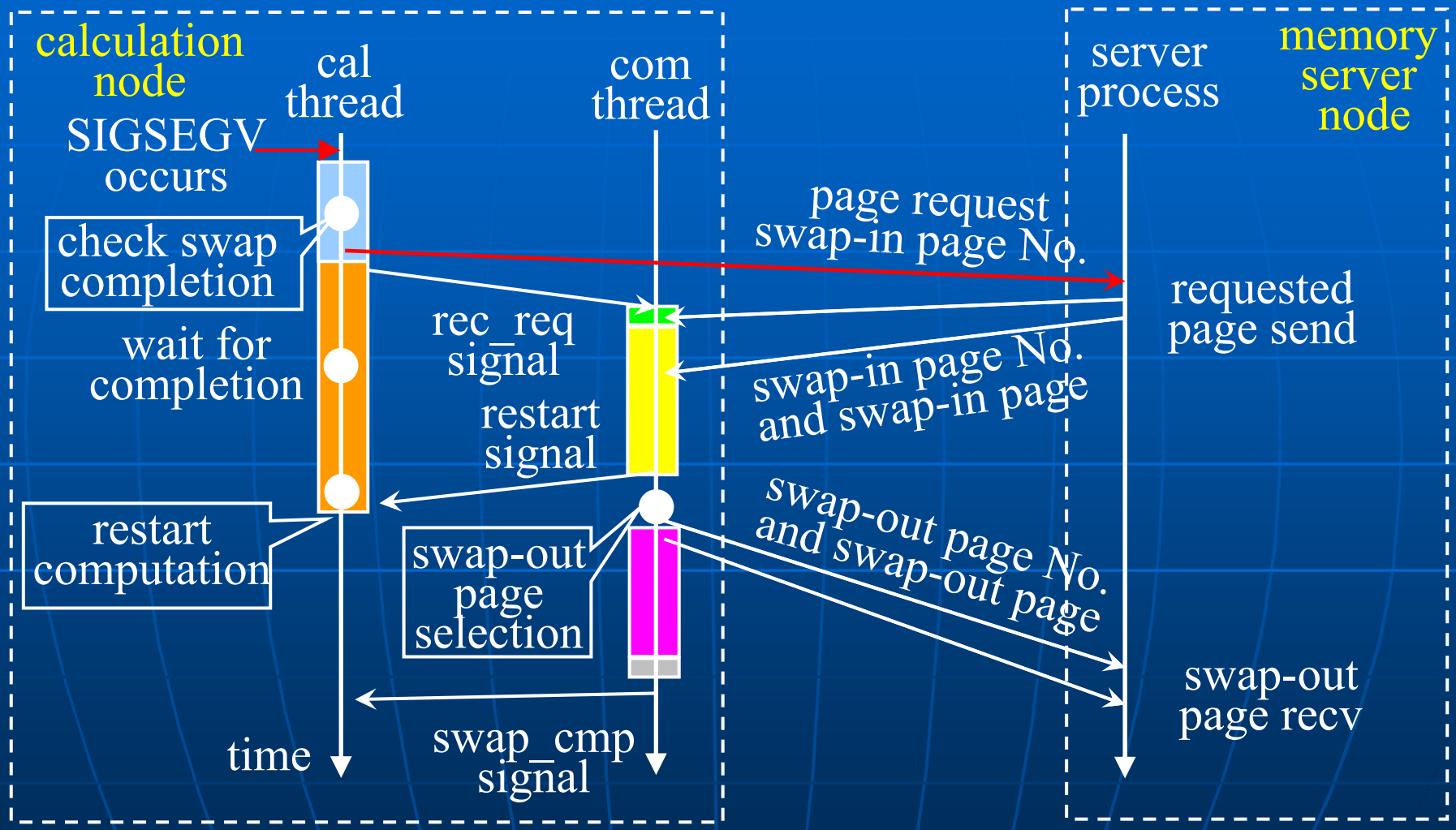
```
#!/bin/bash
#@$-q queueA          queue name
#@$-N 4               4 nodes
#@$-J T1              1process / node
#$-lT 8:00:00         time expected

cd programs/
mpirun -np 4 prog args -- –f memfile
```

Memory size / node specification is possible in option

memfile

24000	24GB for rank0
20000	20GB for rank1
10000	10GB for rank2
16000 :	16GB for rank3
:	: :

Swap Protocol for DLM-MPI



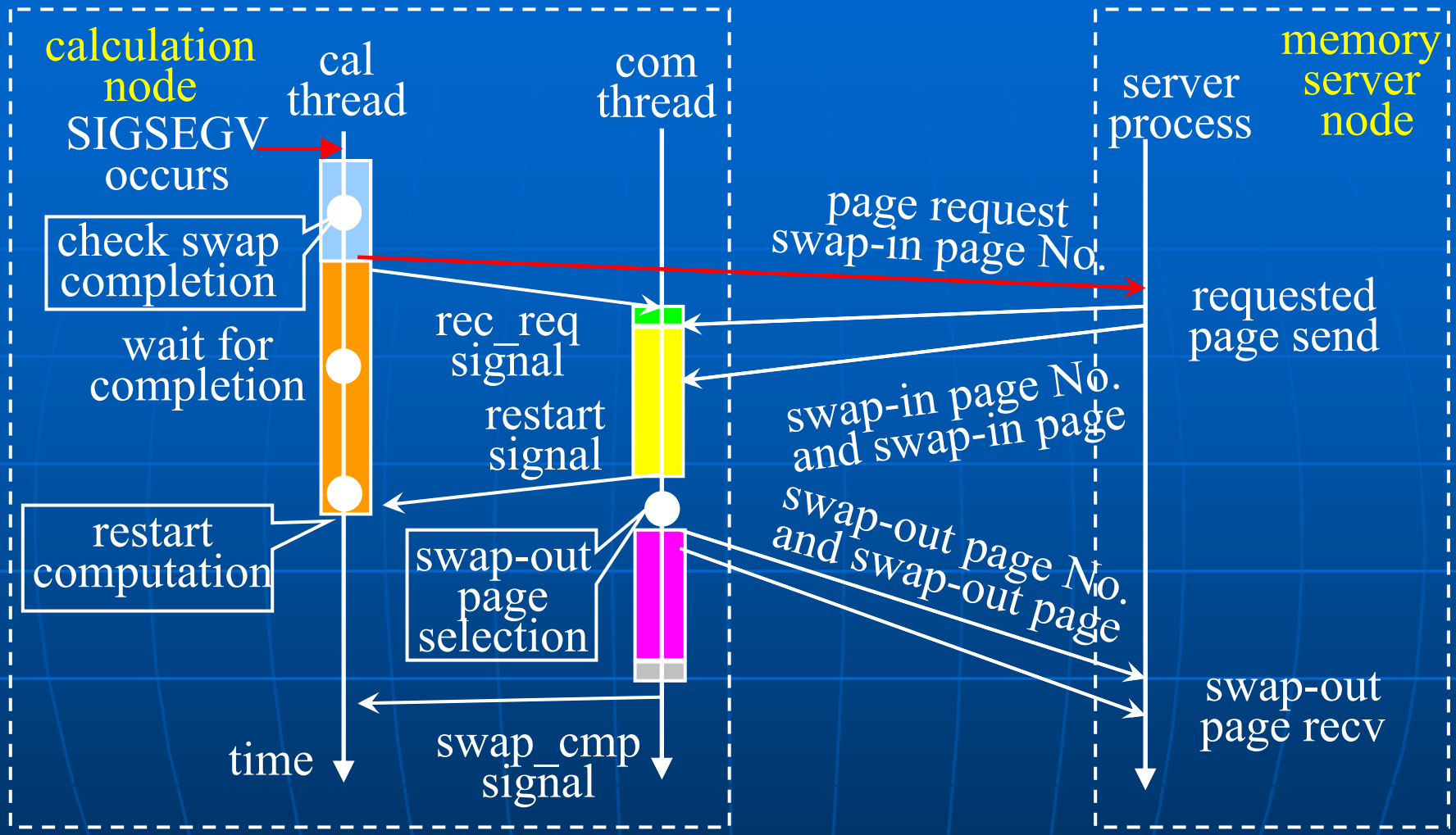
Aggressive Serial-protocol

Introduce concurrency among threads to Serial Protocol

MPI thread support level

1. MPI_THREAD_SINGLE
   Only for single user thread, prohibits use of multiple threads.
2. MPI_THREAD_FUNNELED
   Possible to use multiple threads,
   but MPI calls are available only for a main thread.
3. MPI_THREAD_SERIALIZED
   Possible to use multiple threads,
   but one MPI call is possible for one thread at a time.
4. MPI_THREAD_MULTIPLE
   Possible to use multiple threads,
   all threads can use MPI calls freely.

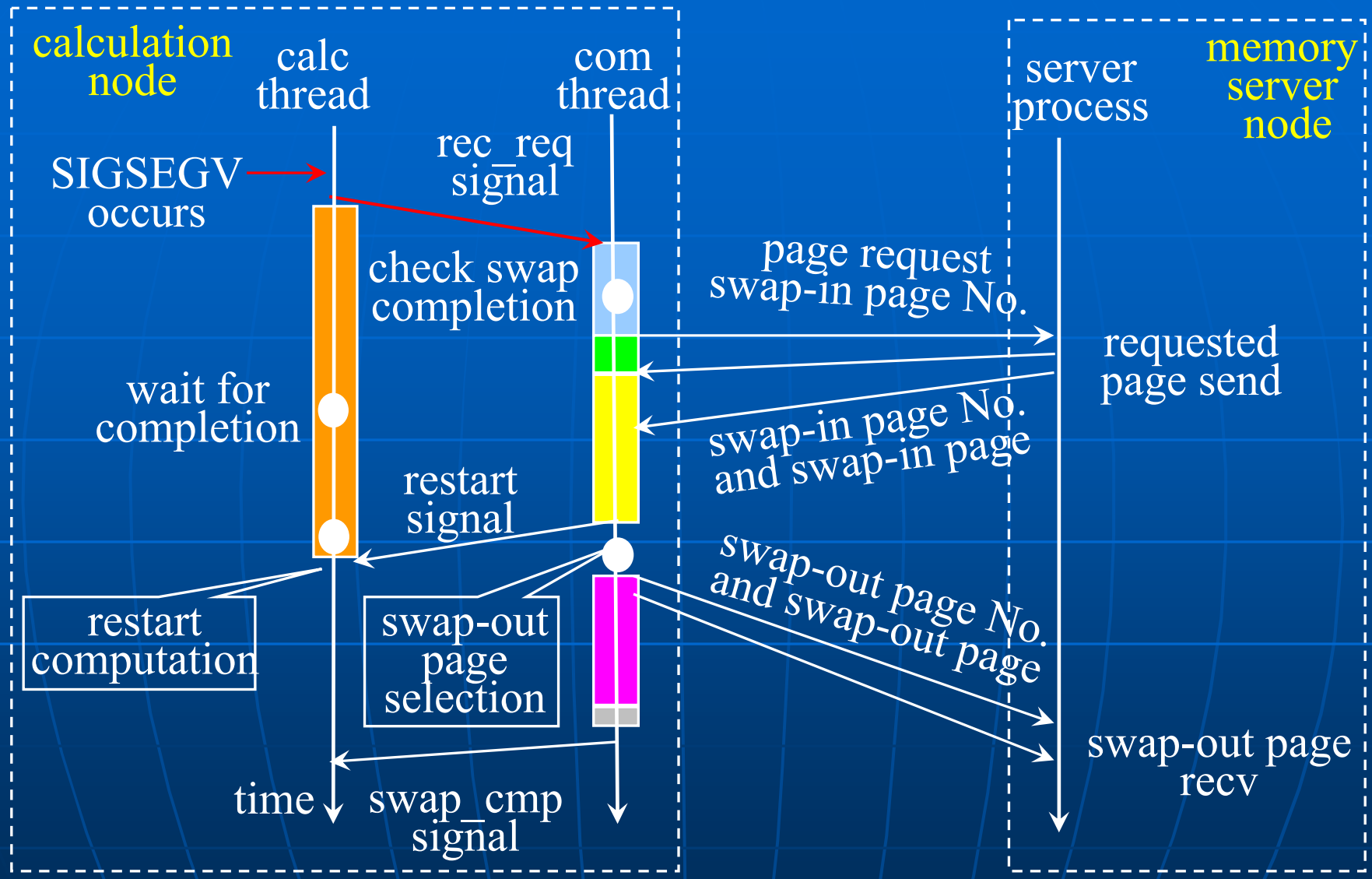
- MPI_Init_thread (int *argc, char ***argv, int required, int *provided)



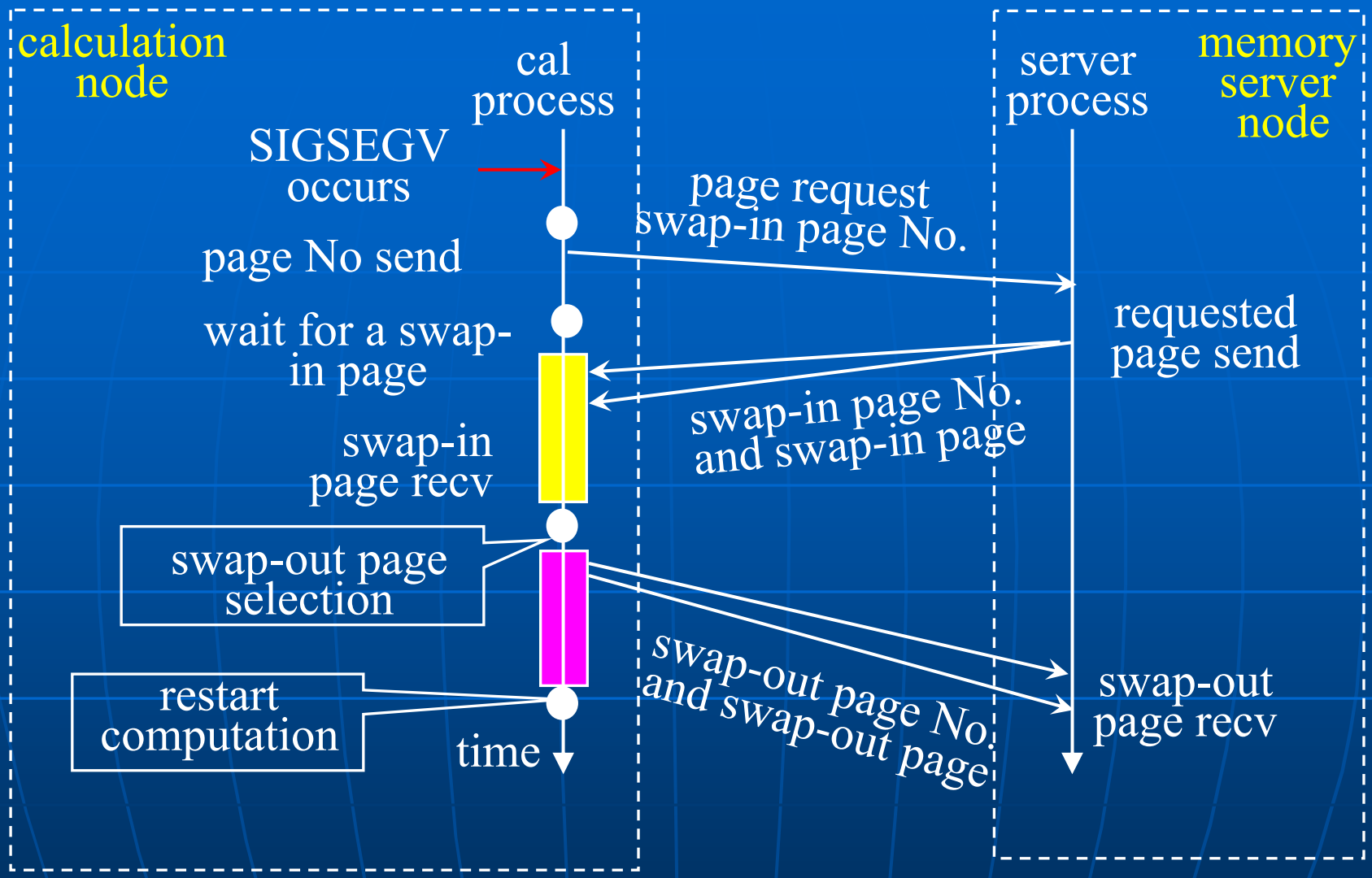
AS1-protocol : Aggressive Serial protocol for MPI_THREAD_SERIALIZED

It may be the fastest in serial protocols,

but requires the higher thread support, level 3, to MPI



AS2-protocol : Aggressive Serial protocol MPI_THREAD_FUNNELED



CS-protocol : Conservative Serial protocol for MPI_THREAD_SINGLE

Available in all MPI implementations : High portability and wide availability, Low efficiency (may be the slowest protocol)

Performance
on a open cluster with
MPI batch queuing system

- DLM page size : 1MB
- 1 process/ node, using 2 – 6 nodes

Terminology in this experiment

- Local memory ratio:

  The ratio of size of data in local memory to size of all data used in a user program

- Relative Execution time:

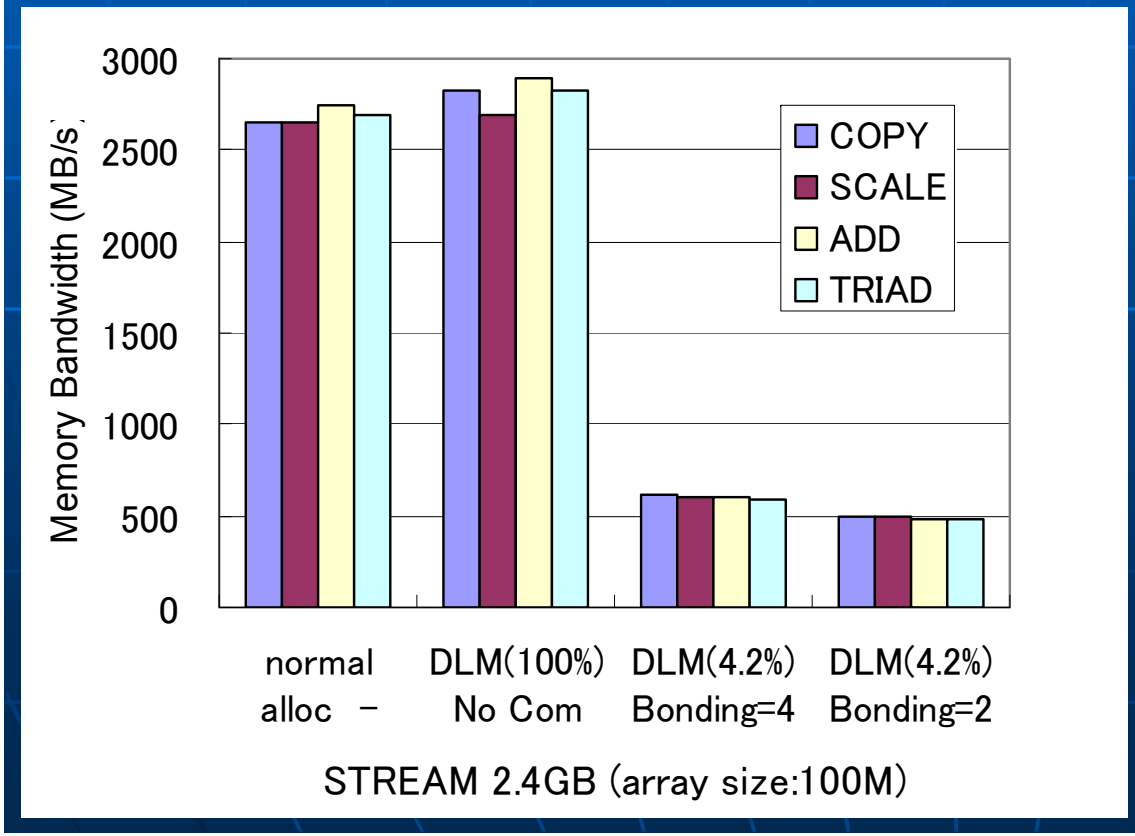
  Relative time to cc-compiled ordinary sequential program execution time using only local memory

Measurement platform
Open cluster with MPI batch queuing system

	T2K Open Supercomputer University Tokyo, HA8000
Machine	HITACHI HA8000-tc/RS425
CPU	AMD QuadCore Opteron 8356(2.3GHz) 4CPU/ node
Memory	32GB/node (936 nodes), 128GB/node (16nodes)
Cache	L2 : 2MB/CPU (512KB/Core), L3 : 2MB/CPU
Network	Myrinet-10G x 4, (5GB/s full-duplex) bonding 4 Myrinet-10G x 2, (2.5GB/s full-duplex) bonding 2
OS	Linux kernel 2.6.18-53.1.19.el5 x86_64
Compiler	gcc version 4.1.2 20070626, Hitachi Optimizing C mpicc for 1.2.7
MPI Lib	MPICH-MX (MPI 1.2)

Remote Memory Bandwidth on STREAM benchmark

	Kernel	Code
STREAM	COPY	a(i) = b(i)
	SCALE	a(i) = q*b(i)
	ADD	a(i) = b(i) + c(i)
	TRIAD	a(i) = b(i) + q*c(i)



DLM-MPI: MPI-MX
493MB/s（Myri-10G x 2）
613MB/s（Myri-10G x 4）

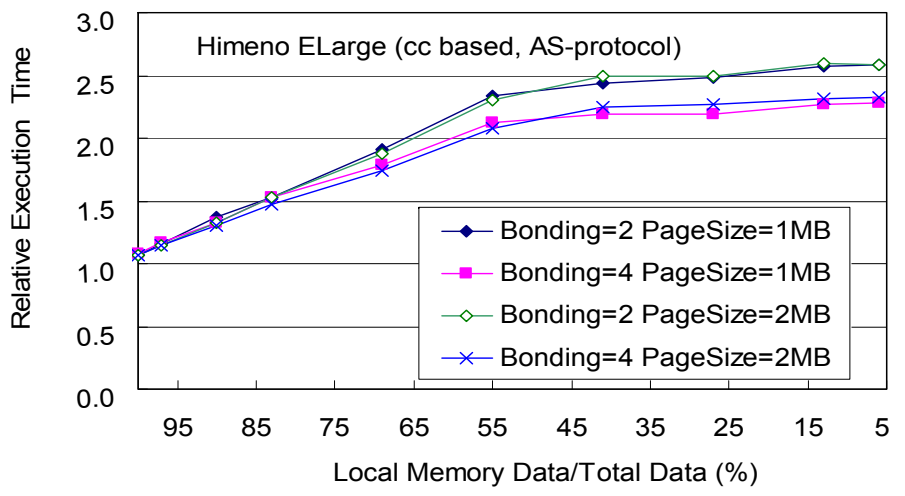
DLM-socket: TCP/IP
Ethernet on Myri-10G x 1
380MB/s

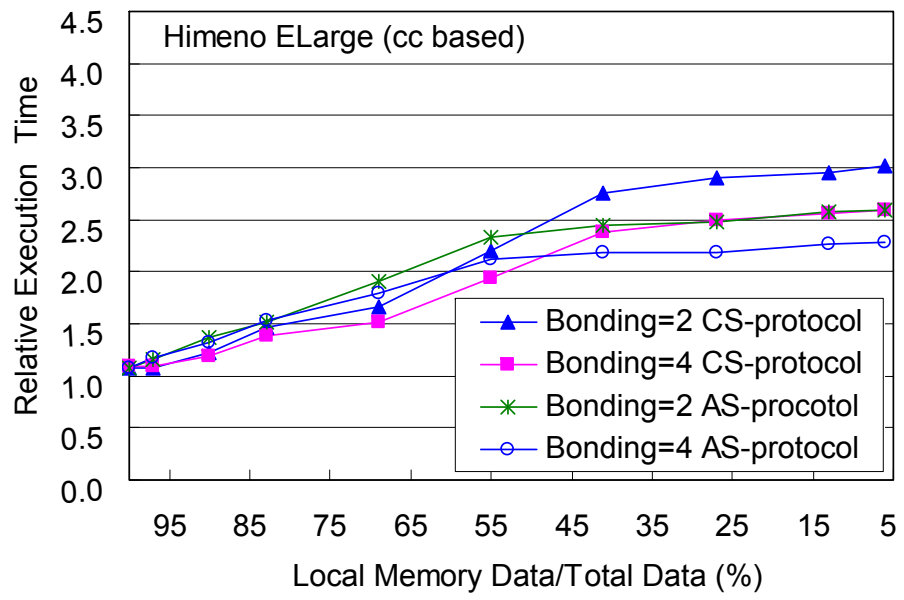
Himeno benchmark relative execution time

(ELarge: float 513 x 513 x1025,15GB,
relative to cc-compiled sequential program using only local memory)

A DLM program is 2 to 3 times slowdown at local memory ratio 6.9%

bonding 4 13% faster than bonding 2





Little difference in DLM page size of 1MB and 2MB

CS protocol is 17% slower than AS protocol at low local memory ratio

DLM Himeno benchmark works on more larger data size

- XLARGE (112GB) 179.4MFLOPS,
  Relative Time 2.32 (based on the time in Elarge,15GB)
  - float 1025 x 1025 x 2049
  - 20GB/node x 6 nodes
  - Local memory ratio 17.4%
  - Bonding = 4
- XLARGE-d (241GB) 88.8 MFLOPS,
  Relative Time 4.68 (based on the time in Elarge,15GB)
  - double 1025 x 1025 x 2049
  - 20GB/node x 12 nodes
  - Local memory ratio 8.1%
  - Bonding = 4

Time analysis of page swapping AS1 protocol and CS protocol

- Himeno Benchmark (MIDDLE, ELARGE)

  Average time component of 1 page swapping
- 6 communication types

  3 MPI implementations

  (MPI-MX, MPICH / Ethernet, OpenMPI / Ethernet)

  3 Communication fabrics, (2 machine platforms)

  （Myri-10G, Ethernet 1Gbps, 10Gbps)
- 2 local memory ratios (high & low)

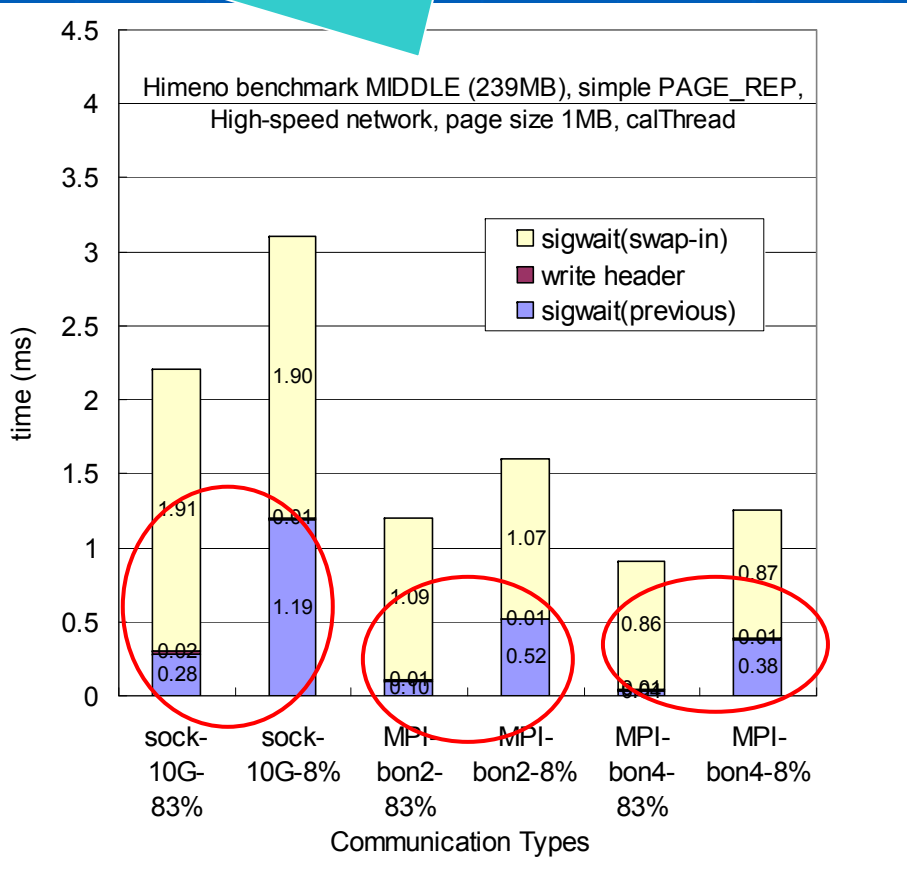
six communication types used in experiments

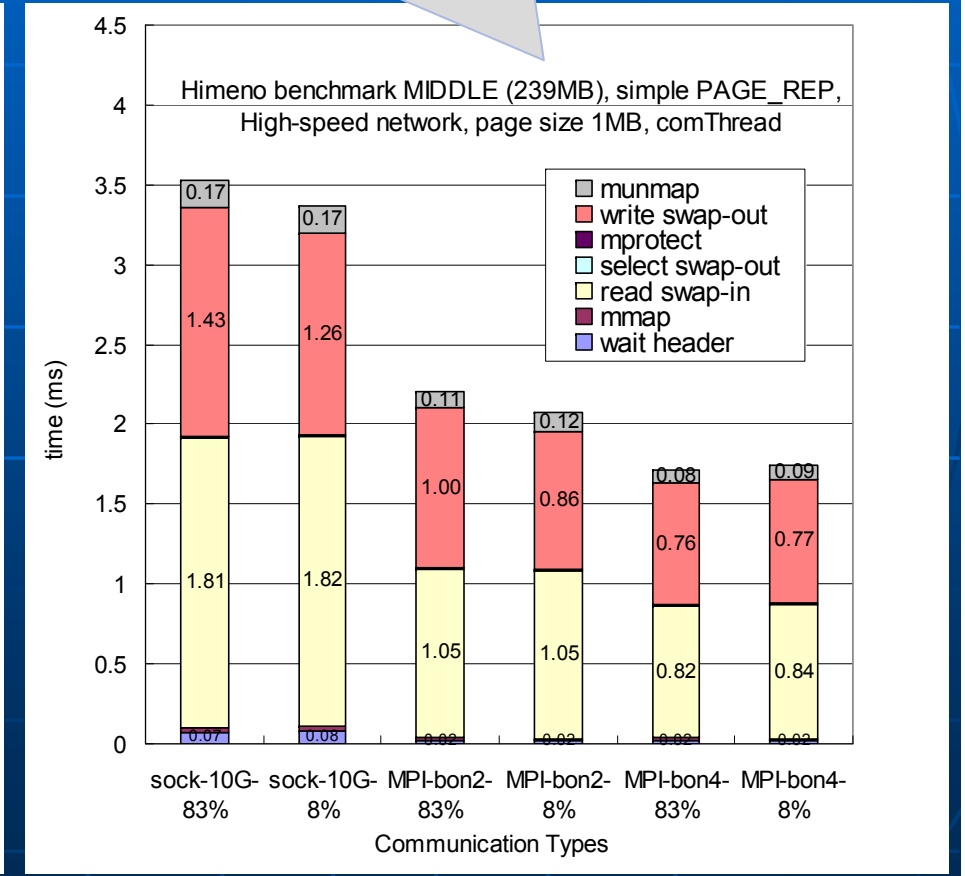
Name	Machine	Network fabric	Protocol
sock-10G	T2K	Myri-10G x 1, IP on Myrinet, 1.25GB/sec	socket TCP/IP
sock-HP	HP	1GbitEthernet	socket TCP/IP
MPI-bon2	T2K	Myri-10G x 2, bonding=2 2.5GB/sec	MPICH-MX (MPI -1.2)
MPI-bon4	T2K	Myri-10G x 4, bonding=4 5GB/sec	MPICH-MX (MPI-1.2)
mpich-HP	HP	1GbitEthernet	MPICH (MPICH-1.2.7)
openmpi-HP	HP	1GbitEthernet	Open MPI (OpenMPI-1.3.2)

Time components in AS1 swap protocol
(Himeno MIDDLE, T2K, Myri10G)

Swap time in calculation thread has longer waiting time at the smaller local memory ratio with the same communication link

page receive/send time is dominant.
Other components are small



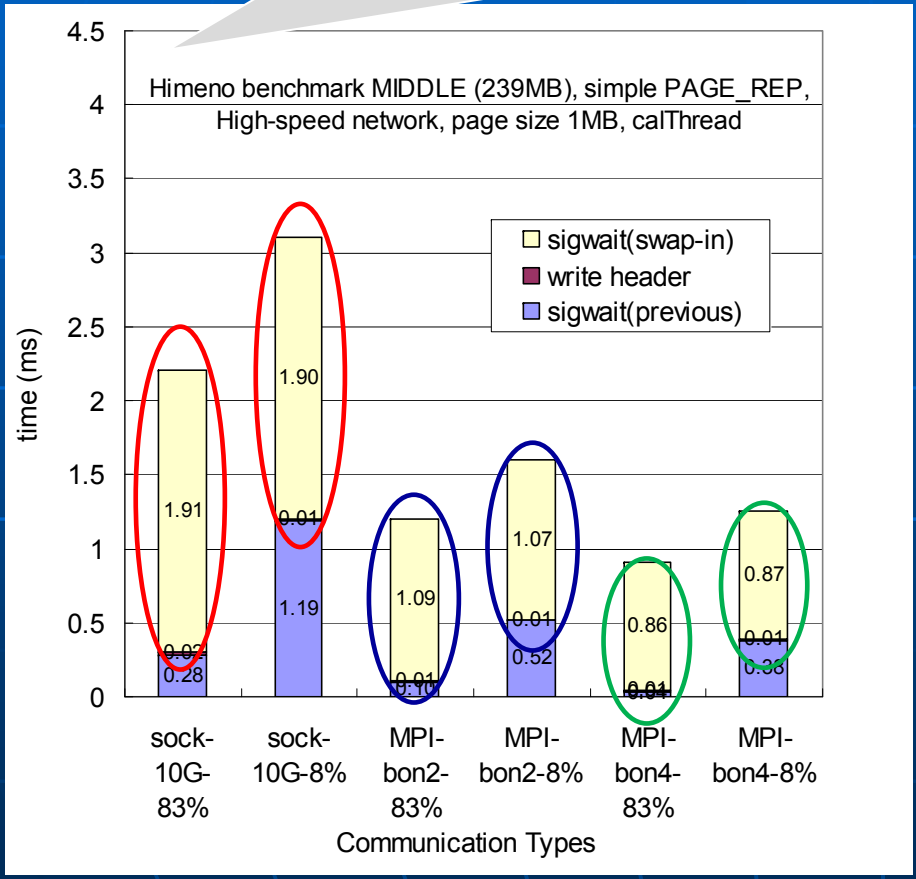


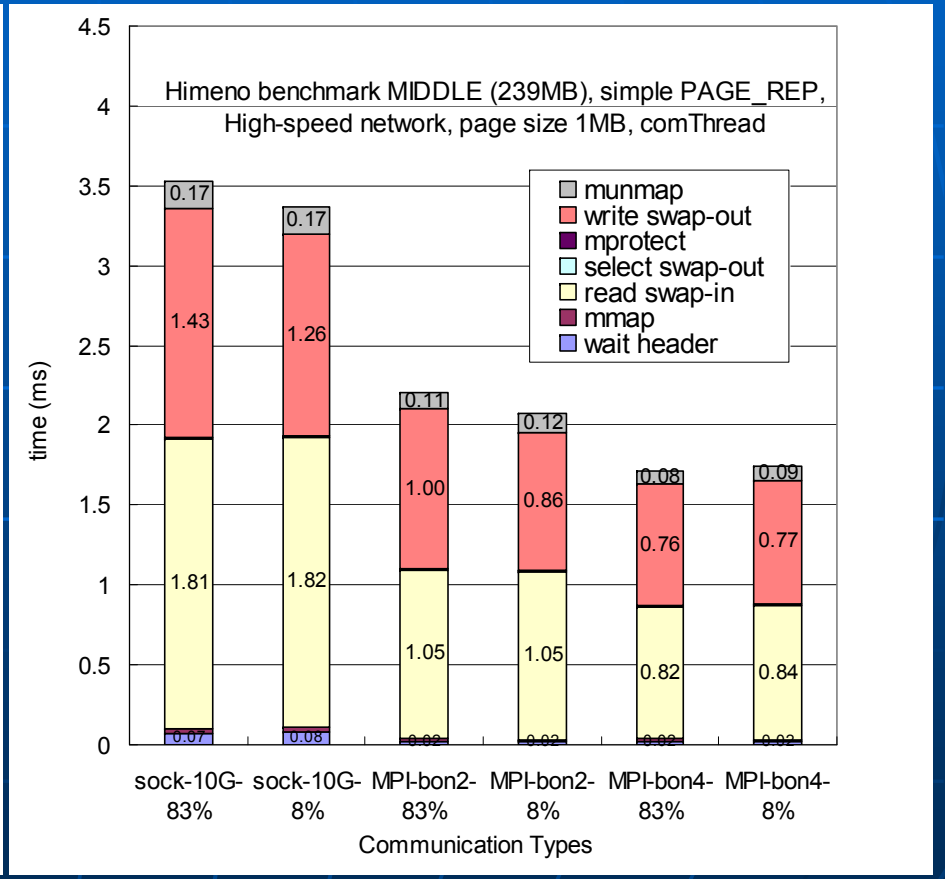
Calculation thread

Communication thread

Time components in AS1 swap protocol
(Himeno MIDDLE, T2K, Myri10G)

Calculated bandwidth by swap-in time in calculation thread
Socket 526MB/s, bonding2 935MB/s, bonding4 1149MB/s

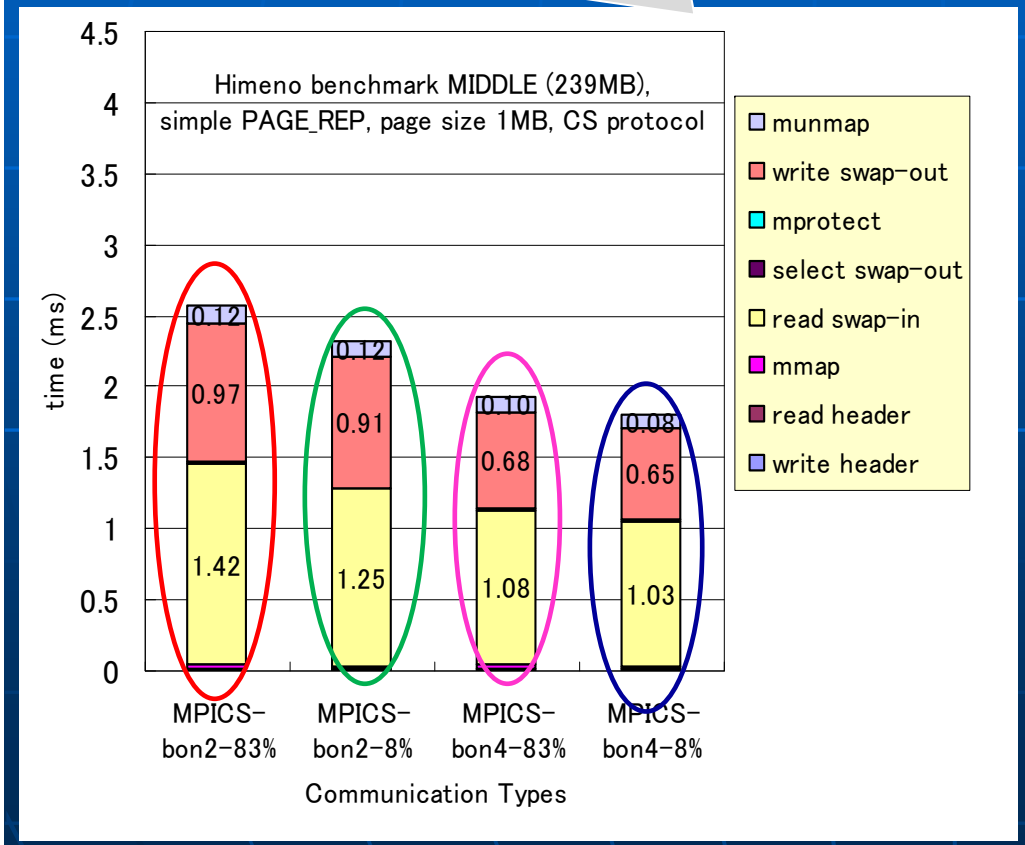


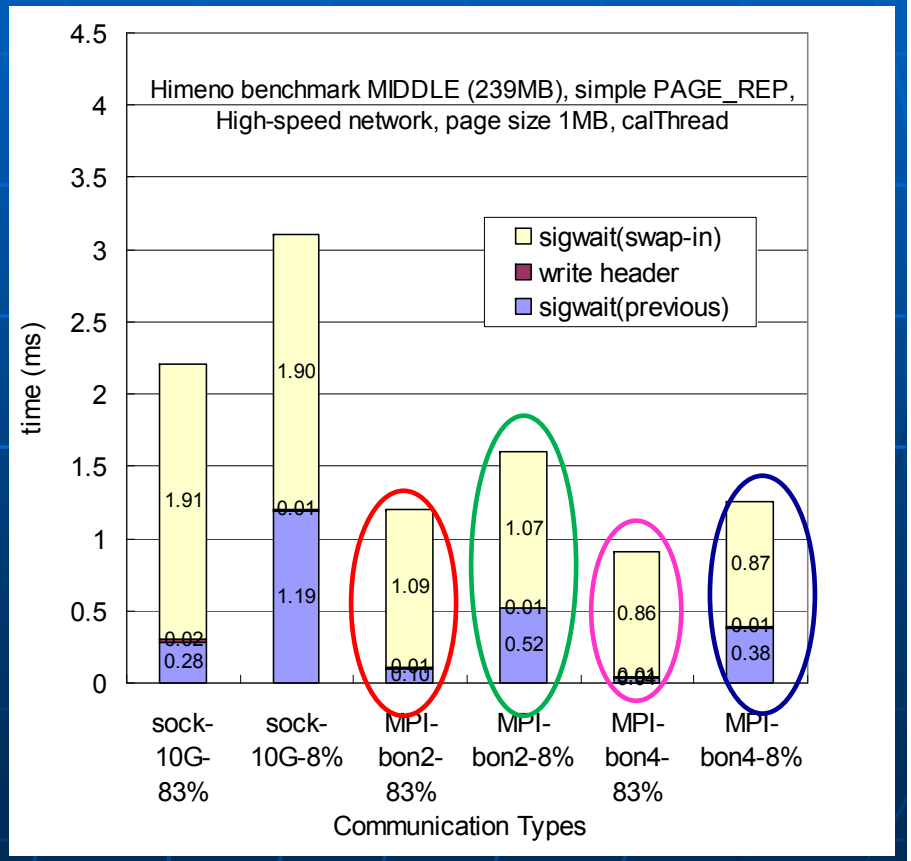


In loop test with MPI_Send and MPI_Recv of 1MB-message
Read page time : 1.57ms in bonding2, 1.73ms in bonding4

Time components in AS1 and CS swap protocol
(Himeno MIDDLE, T2K, Myri10G)

Perfomance in swap-in time in calculation thread
AS is 2 times better than CS at high local memory ratio
AS is 1.4 times better than CS at low local memory ratio





CS protocol

AS1 protocol

Performance evaluation using NPB (NAS Parallel Benchmarks)

- NPB is designed for parallel processing:
  outputs a execution time of a core calculation (iterations)
- 3 phases: Data Initialization, Core calculation, Verification
- data usage and time consuming ratios for 3 phases depend on applications

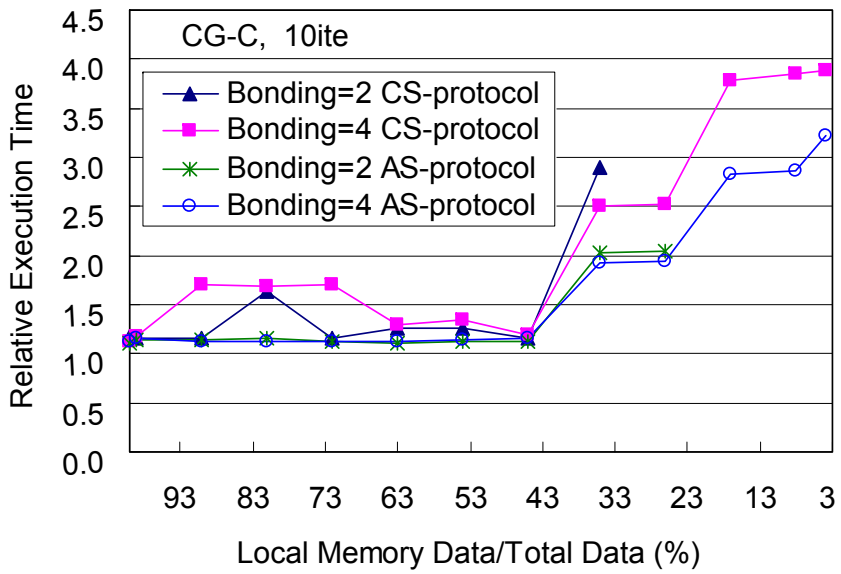
Experimental setting

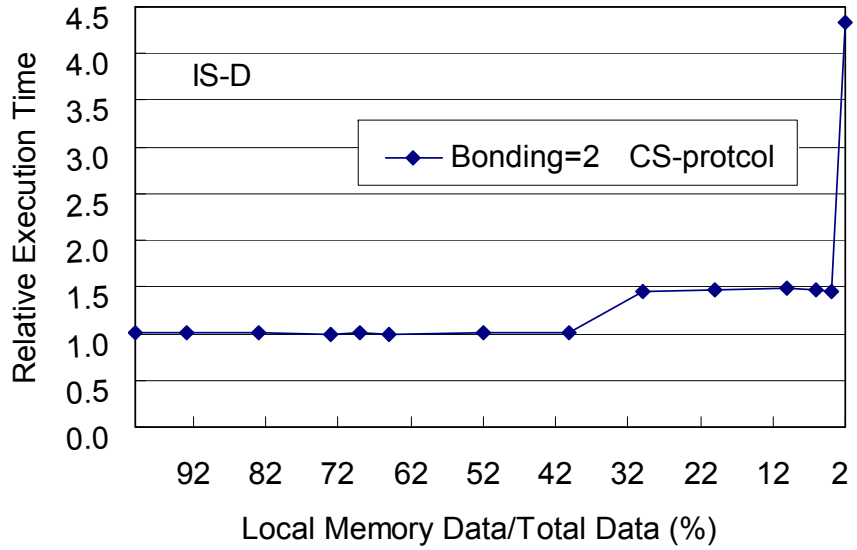
- Dominant large data are specified as DLM data
- Focus on core calculation time (NPB outputs)
- Use sequential C programs
  - NPB 2.3-omni-C : CLASS-C (the max class in the suit)
    IS, CG, MG, FT, BT, SP
  - NPB 3.3 : CLASS-D IS （51.4GB)

Relative execution time in NPB IS & CG

Core calculation time
AS 20% faster than CS
at local memory ratio 4.17%
No difference in link speeds
1.1 at local memory ratio > 40%
3 – 4 at local memory ratio 3%

Core calculation time
No slowdown local mem ratio > 40%
1.5 at local memory ratio 42 - 3%
4.3 at local memory ratio 2%





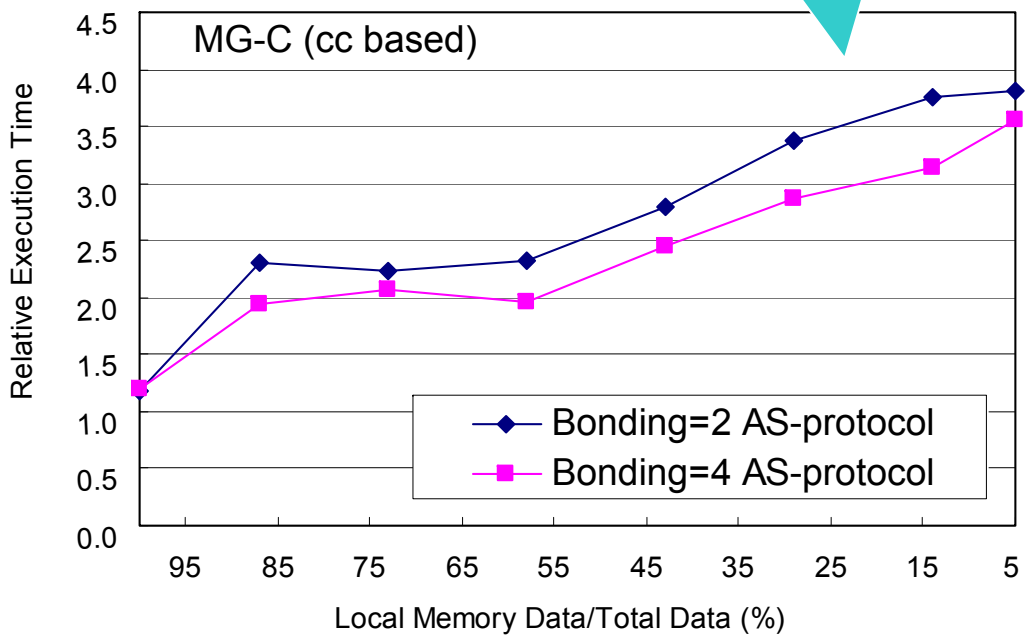
CG : CLASS-C, 15000, 1.15 GB

IS: CLASS-D, 51.5 GB

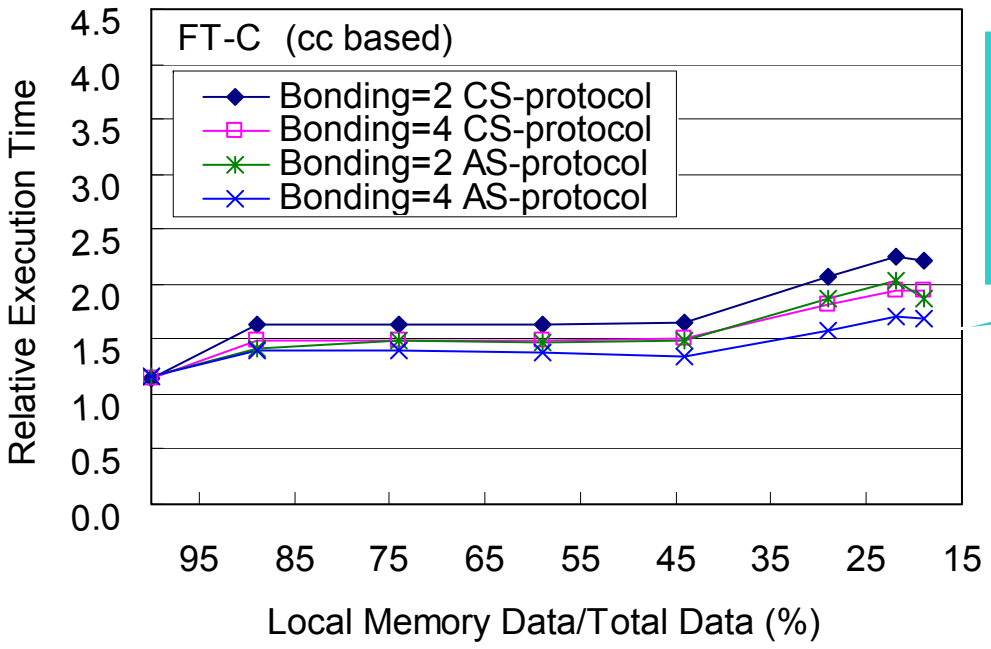
Relative execution time in NPB MG
(NPB-2.3-omni-C, 512 x 512 x 512, 3.6GB)

Num of dynamic allocations is 974,107 times

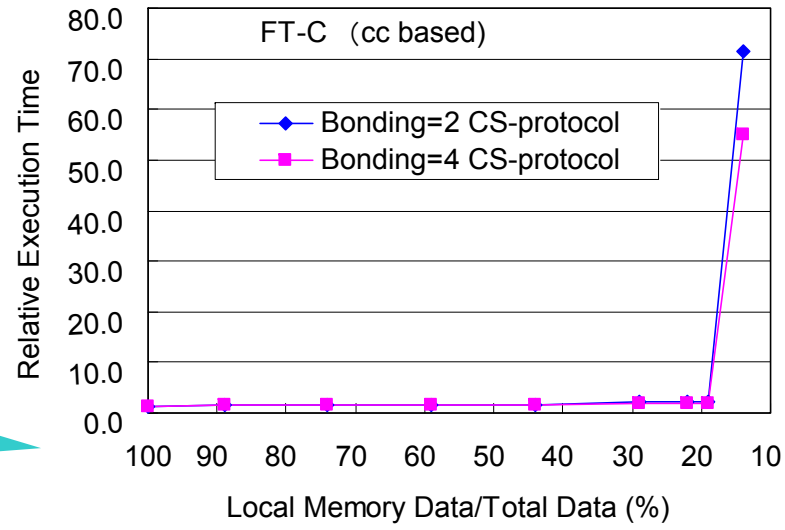
Core calculation time
Bonding 4 19% faster than bonding2 at the max



Relative execution time in NPB FT
（NPB-2.3-omni-C, 512 x 512 x 512, 7GB)



1.16 at local memory ratio 100%
1.69 (bonding4) and1.86 (bonding2) at local memory ratio19.4%



54.1 (bonding4) and 71.2 (bonding2) at local memory ratio14.9%

Conclusion

- MPI based DLM (DLM-MPI) is available on various kinds of clusters including open clusters as a fast and large virtual memory for general users.
- Some application benchmarks show relatively small performance degradation, 2 to 4 slowdown than normal execution, even if local memory ratio is 4% to 6%.
- DLM-MPI exploits a thread technology in multi-core CPU to provides multiple swap protocols for MPI thread support levels.
- It provides easy programming interface for general users who have no knowledge of parallel programming.

Remaining issues and future works

- Analysis of memory access locality in applications and the relationship between working set and DLM page size.

- Development of an effective and low-cost page replacement algorithm for user-level implementation

  (Simple algorithm is employed in this experiment.

  A swap-out page is selected in round-robin fashion over address)

- Design and evaluation for more wider varieties of swap protocols, cross/serial, MPI thread-support levels, communication fabrics used in MPI

- Enhancement for multi-thread user applications