CLUSTER 2017 IEEE International Conference on Cluster Computing, September 5th - 8th, 2017, Hawaii, USA.

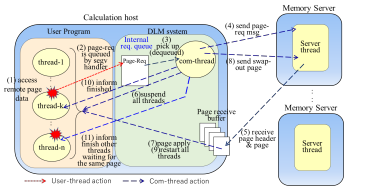
Efficient Swap Protocol for Remote Memory Paging in Out-of-Core Multi-Thread Applications

Hiroko Midorikawa, Kenji Kitagawa, Hikari Ohura, JST CREST & Seikei University, Tokyo Japan
midori@st.seikei.ac.jp http://www.ci.seikei.ac.jp/midori/paper

Abstract

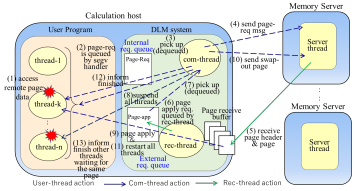
A new page swap protocol is proposed for user-level remote memory paging systems to accelerate the performance of out-of-core processing with multi-thread user programs and libraries written in OpenMP and pthread. The original swap protocol has a bottleneck in efficient page swapping, which is requested by multiple threads in a user program, because all the MPI communications to the memory servers and page swap managements are allocated to one system thread. However, this protocol has a benefit that it is widely available anywhere even if MPI thread-support-level is not so high. The new protocol uses two independent system threads: one for page swapping, and the other for managing user thread requests. Although it requires the highest MPI thread-support-level (multiple), which is usually considered to degrade the MPI communication performance compared to than in lower MPI thread-support-level, the new protocol achieves higher performance than the original protocol in micro benchmark, Stream benchmark, matrix multiplication, stencil computation, and 3-Dimentional FFT.

Original page swap implementation (base: r3-protocol)



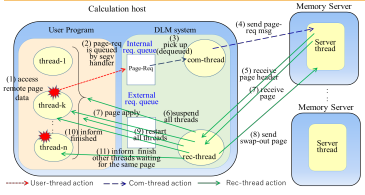
Centralized control system by com-thread
- No mutual exclusions for accessing the DLM internal system data.
- Widely available anywhere, MPI thread-support-level is low. (FUNNELED, SERIALIZED)
- Bottleneck in efficient page swapping. All requests from user threads are managed by a single *com-thread*, one by one.

Revised page swap implementation (revised 1: r58-protocol)



Introducing receiver thread to the centralized control system (r3-protocol)
- No mutual exclusions for accessing the DLM internal system data.
- Concurrent MPI communications require MPI thread-support-level MULTIPLE.
- Rec-thread dedicates receiving swap-in pages in receiver-buffers.
- *Com-thread* manages sending page-requests, page-applying, and sending swap-out pages.

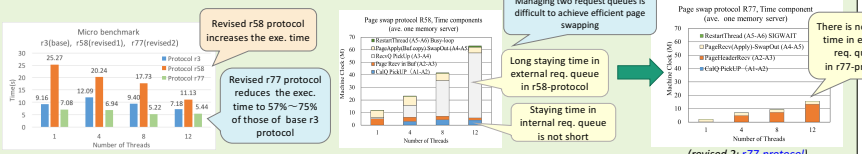
Revised page swap implementation (revised 2: r77-protocol)



Two system threads manage their works independently, without page-buffers.
- Mutual exclusions are required to access the DLM internal system data.
- Two MPI Communication threads requires MPI thread-support-level MULTIPLE.
- Two system thread run independently. Page-buffers are removed.
  Rec-thread manages all page swapping (receive , apply, send) .
  Com-thread manages only user requests.

Performance evaluations : base protocol vs. revised protocols

Micro Benchmark (Fig.3.)



Time base(r3) protocol vs. revised (r58 and r77) protocols

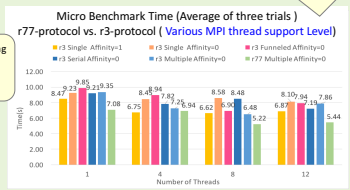
(revised 1: r58-protocol)

(revised 2: r77-protocol)

Time breakdown for one page swapping (Average of all page swap) in revised protocols

Micro Benchmark (Fig.3.)

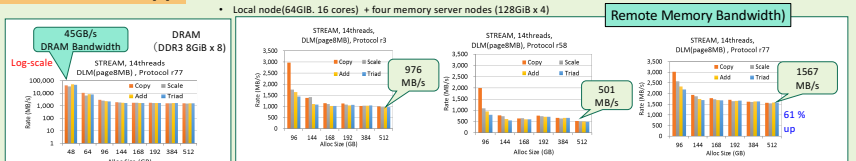
The best is r77 protocol.
The second is r3 protocol with SINGLE and thread-affinity (stable behavior)



STREAM Benchmark [3]

- Problem Size : 48GiB ～512GiB
- Local node(64GiB. 16 cores) + four memory server nodes (128GiB x 4)

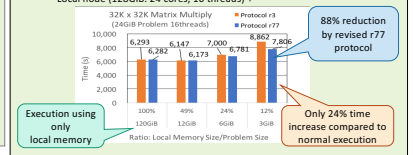
Remote Memory Bandwidth



(revised 2: r77-protocol) *(base: r3-protocol)* *(revised 1: r58-protocol)* *(revised 2: r77-protocol)*

Matrix Multiplication

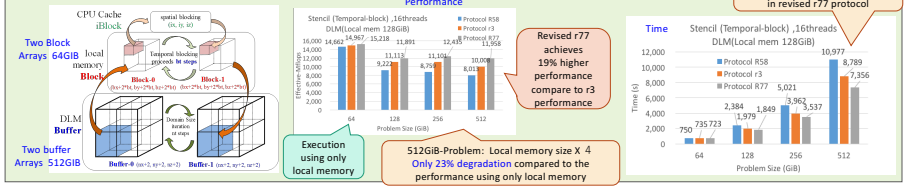
- Fixed Problem Size : (32 K x 32 K matrix : 24 GiB)
- Available Local memory size is reduced from 120 GiB to 3 GiB.
- Local node (128GiB. 24 cores, 16 threads) +



7-point Stencil computation with Temporal blocking [4]

- Stencil Problem Data Size : 64GiB ～512GiB, 256 time steps
- Local node(128GiB. 20 cores) + four memory server nodes (128GiB x 4), 16-threads

Performance



Result Summary

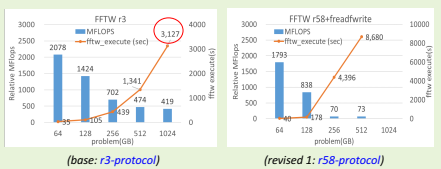
In all the cases, the proposed protocol r77 achieved a better performance by using two system threads that are separately responsible for page swap and user thread request management and communicate to the memory servers independently

Comparison between revised r77 protocol and base r3 protocol
- Micro Benchmark: Exec. time 43% - 25% reduction
- Stream Benchmark: Remote Memory bandwidth 61% increases
- Stencil Computing: Exec. time of 512GiB-problem (Local memory ratio 25%) 19% reduction
- Matrix Multiplication: Exec. time of 24GiB-Problem (Local memory ratio 25%) 12% reduction
- 3D-FFT: Exec. time of 1TiB-Problem (Local memory ratio 12.5%) 12% reduction

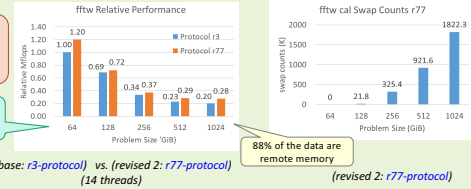
3D Fourier Transform using fftw library [5]

- Two 3D- arrays, src and dst, allocated with dlm_alloc are passed as to 3dfftw( ) function arguments .
- Local node(128GiB. 16 cores) + 7 memory server nodes (63GiB - 128GiB), 14-threads

fftw library (fftw3.3.6-pl1, libfftw3_omp.a)



(base: r3-protocol) *(revised 1: r58-protocol)* *(revised 2: r77-protocol)*

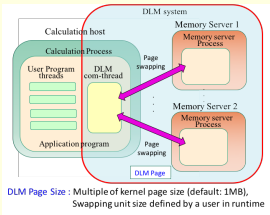


(base: r3-protocol) vs. (revised 2: r77-protocol) (14 threads)

(revised 2: r77-protocol)

DLM (Distributed Large Memory) [1][2] for Out-Of-Core Multi-thread Program Executions

A user-level remote memory paging system, DLM (Distributed Large Memory), offers virtual large memory using a cluster of distributed node memories [1][2]. It is highly portable to various clusters and is an easy-to-use remote memory for people without any knowledge of MPI. It was designed for users who need to solve large-size problems that require the processing of large amounts of data beyond the capacity of local memories, using existing algorithms and programs originally designed for shared-memory models. They prefer and accept the extra execution time caused by partially using remote memory instead of the local memory, because converting existing complex algorithms to parallel MPI programs is not an easy task and requires substantial costs. The DLM supports multi-thread programs and libraries written in OpenMP and pthread[1].



DLM Page Size : Multiple of kernel page size (default: 1MB), Swapping unit size defined by a user in runtime

Fig. 1 DLM

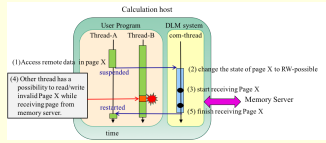


Fig.2 Difficulties in page swapping by multiple threads

DLM System :
- Virtually provides a single large memory using distributed memories in a cluster for multi-thread programs designed for shared-memory-models, such as OpenMP and pthread.
- A Transparent system to user applications : The segv-signal-handler invoked by each thread in an application fetches a remote page implicitly.
- Little modification for existing programs
- No need of MPI programming knowledge for using remote memories

Data inconsistency is generated when page fetch and data access are occurred simultaneously.
- Temporal suspension of all user threads in a program during copying a remote page to the user address space is employed.
  pthread_kill (pthread_t *thread*, int *sig*);
- *pthread_create()* is hooked by the DLM system to register thread IDs of user threads which are generated and terminated dynamically in program execution.

The Programming interface for DLM

- *Convert malloc to dlm_alloc for dynamic allocations*

DLM Multi-thread Program (malloc → dlm_alloc)

```
#define ENUM ((unsigned long ) (1L<<34))   //16G Elements
int main( )
{
dlm_startup(&argc, &argv);  // Initiate the DLM System
// 128 GiB array allocation
 array = (unsigned long *) dlm_alloc (sizeof(unsigned long) *
ENUM);
 // Initialization, array parallel write
 #pragma omp parallel for
   for ( i = 0; i < ENUM; i++ ) array[i] = i;
 // Array parallel read per 1MB (DLM pag
 #pragma omp parallel for
 for ( i = 0;  i < ENUM; i+=(1L<<17) )
     if (array[i] != i )  return 1;
dlm_shutdown();   // Finaliz
}
```

Fig. 3 The program using DLM library
Micro Benchmark for remote memory read and write

- *Use the dlmc translator for array-data [6]*

```
#define MAX 1000
dlm int a[MAX][MAX];
int main (int argc, char *argv[])
{
 int i, j;
 for ( i = 0; i < MAX; i++)
   for ( j = 0; j < MAX; j++)       a[i][j] = i*j;
 for ( i = 0; i < MAX; i++)
   for ( j = 0; j < MAX; j++)  printf("%d", a[i][j]);
 return 0;
}
```

Converted by dlmc translator

```
int (*dlm_a)[MAX];
int main (int argc, char *argv[])
{
  int i, j;
  dlm_startup( &argc,&argv);
  dlm_a = (int (*) [MAX]) dlm_alloc ( sizeof(int) *MAX*MAX );
  for ( i = 0; i < MAX; i++)
    for ( j = 0; j < MAX; j++)     dlm_a[i][j] = i*j;
  for ( i = 0; i < MAX; i++)
    for ( j = 0; j < MAX; j++)     printf("%d",dlm_a[i][j]);
  dlm_shutdown();
  return 0;
}
```

Fig. 4 DLMC Translation

Current and Future works

- Protocols designed for DLM cache-model (r3, r77 protocols were designed for DLM flat-model.)
- Protocols designed for m-DSM (Distributed Shared Memory for multithread programs) (r58 is modified version of the current protocol used in m-DSM.)

Experimental Environment

Server Name	Sandy Bridge Server crest2,3,4	Haswell Server crest5,6,7,8	Broadwell Server crest9,10
Network	Infiniband single FDRx4 (56Gbps)		
Node CPU	Xeon E5-2687W (3.10GHz) 2 CPU x 8 Core	Xeon E5-2687W v3 (3.10GHz) 2 CPU x 10 Core	Xeon E5-2687W v4 (3.00GHz) 2 CPU x 12 Core
Node Memory	DDR3-1600, crest4: 8GiBx8 (64 GiB) crest3: 16GiB x8 or x12 (128 /196 GiB), crest2: 8GiB x 16 (128 GiB)	DDR4-2133 16GiB x 8 (128 GiB)	DDR4-2400 16GiB x 8 (128 GiB)
L1 cache	32 KiB	32 KiB	32 KiB
L2 cache	256 KiB	256 KiB	256 KiB
L3 cache	20 MiB	25 MiB	30 MiB
OS	CentOS 7.1.1503 (x86_64), kernel 3.19.5	CentOS 7.1.1503 (x86_64), kernel 3.19.5	CentOS 7.2 (x86_64), kernel 3.19.5
Compiler	gcc version 4.8.3 -O3	gcc version 4.8.3 -O3	gcc version 4.8.3 -O3
MPI	MVAPICH2 2.0.1_mrail_OFED2.4-1.0.4		

References

[1] H. Midorikawa, Y. Suzuki, M.Iwaida: "User-level Remote Memory Paging for Multithreaded Applications", IEEE/ACM International Symp. on Cluster, Cloud and the Grid Computing CCGrid2013, pp.196-197, 2013
[2] H. Midorikawa, K.Saito, M.Sato, T.Boku: "Using a Cluster as a Memory Resource: A Fast and Large Virtual Memory on MPI", IEEE International Conf. of Cluster Computing, Cluster 2009, pp.1-10, 2009
[3]Stream Benchmark https://www.cs.virginia.edu/stream/
[4] H. Midorikawa, H.Tan, T.Endo,"An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations", IEEE The 12th International Conf. on High Performance Computing & Simulation
[5] fftw library http://www.fftw.org/
[6] S. Yoshimura, H.Midorikawa; "A C Compiler for Large Data Sequential Processing using Remote Memory", proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.198-202 , 2011-8 (DOI: 10.1109/PACRIM.2011.6032892)