A Memory Management System Optimized for BDMPI's Memory and Execution Model

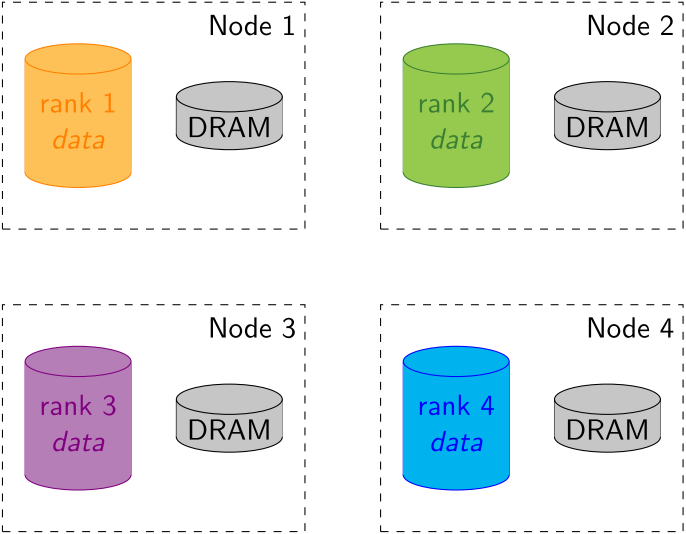
Jeremy Iverson[1] George Karypis[1]

[1]University of Minnesota, Minneapolis, MN, USA

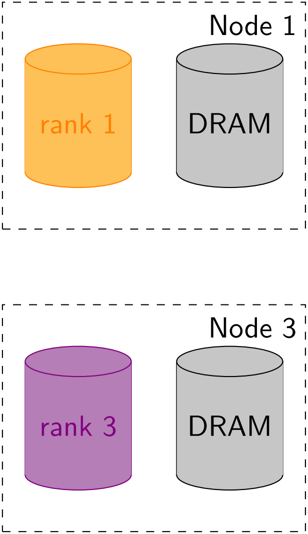
EuroMPI 2015

October 4, 2015

Consider this parallel application

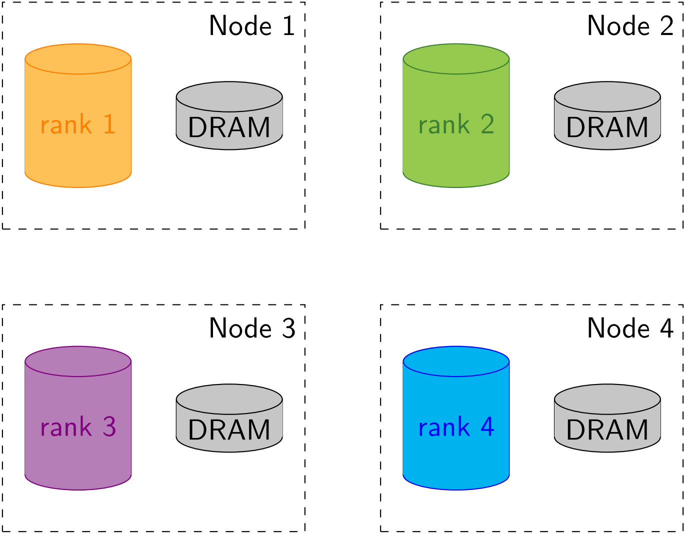


One *simple* solution

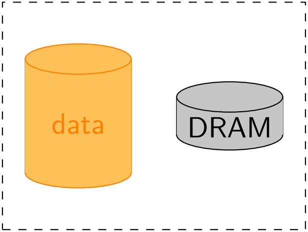


A *more realistic* solution

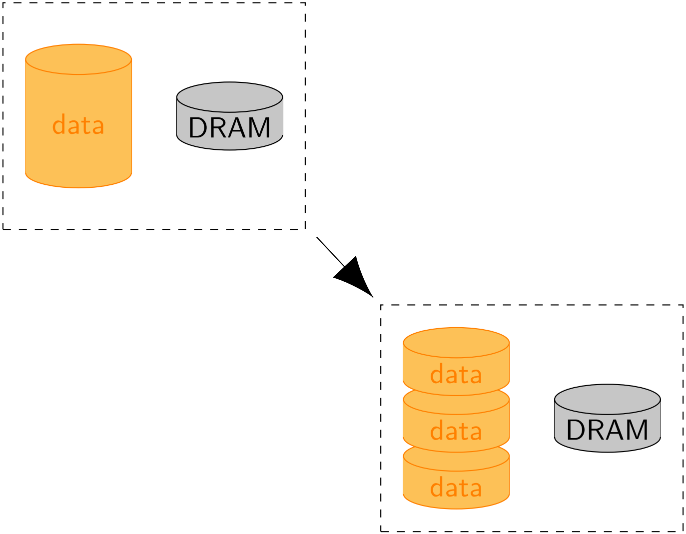
But what if hardware is fixed?



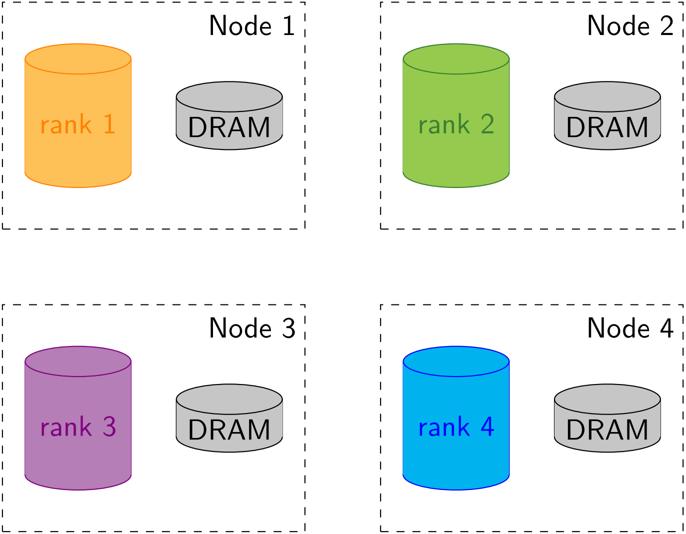
Let's look at a serial application



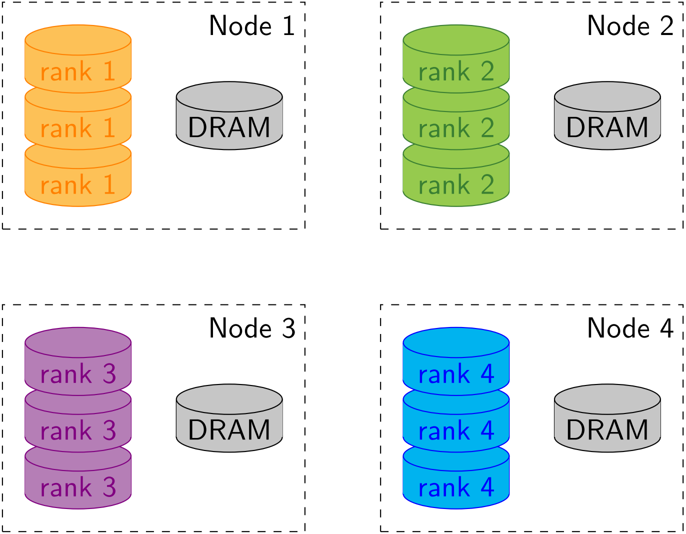
Let's look at a serial application



Now recall the parallel application...



. . . and apply the serial solution



Remember the *more realistic* solution?

What if we could just...

Enter BDMPI

BigData MPI (BDMPI)

- Transparent layer between an MPI application and an MPI runtime

Node-level co-operative multi-tasking (execution model)

- MPI process will run until it blocks for a communication operation (collective, recv)
- Cost of loading data from disk is amortized over large segments of computation

Constrained memory over-subscription (memory model)

- Assumes the problem is decomposed s.t. each MPI process can fit its working set in memory
- Manages the scheduling of MPI processes per compute node to reduce pressure on OS swapping mechanism

BigData MPI (BDMPI)

- Transparent layer between an MPI application and an MPI runtime

Node-level co-operative multi-tasking (execution model)

- MPI process will run until it blocks for a communication operation (collective, recv)
- Cost of loading data from disk is amortized over large segments of computation

Constrained memory over-subscription (memory model)

- Assumes the problem is decomposed s.t. each MPI process can fit its working set in memory
- Manages the scheduling of MPI processes per compute node to reduce pressure on OS swapping mechanism

Enter BDMPI

BigData MPI (BDMPI)

- Transparent layer between an MPI application and an MPI runtime

Node-level co-operative multi-tasking (execution model)

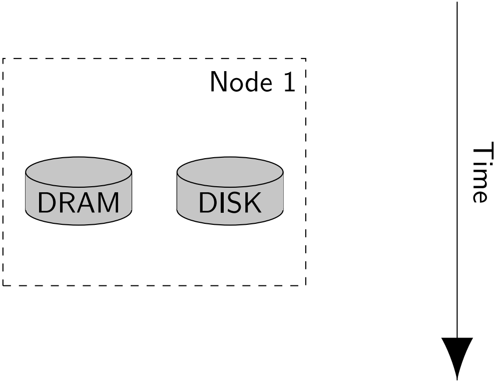
- MPI process will run until it blocks for a communication operation (collective, recv)
- Cost of loading data from disk is amortized over large segments of computation

Constrained memory over-subscription (memory model)

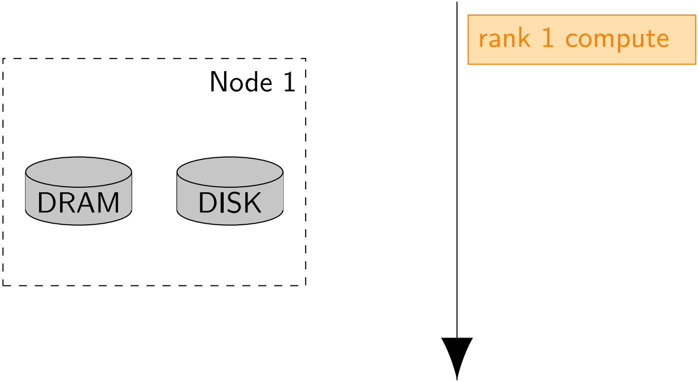
- Assumes the problem is decomposed s.t. each MPI process can fit its working set in memory
- Manages the scheduling of MPI processes per compute node to reduce pressure on OS swapping mechanism

Overview

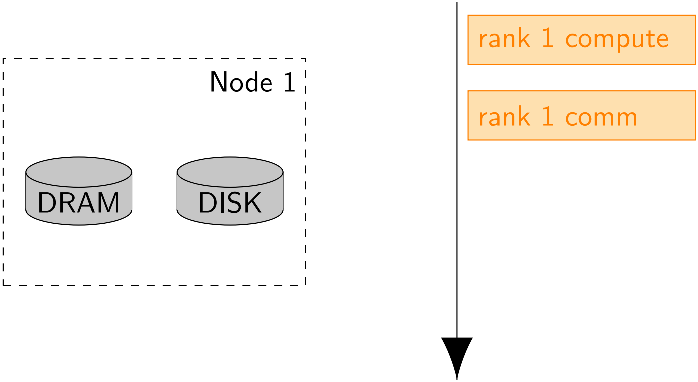
Pitfalls of OS swapping in BDMPI



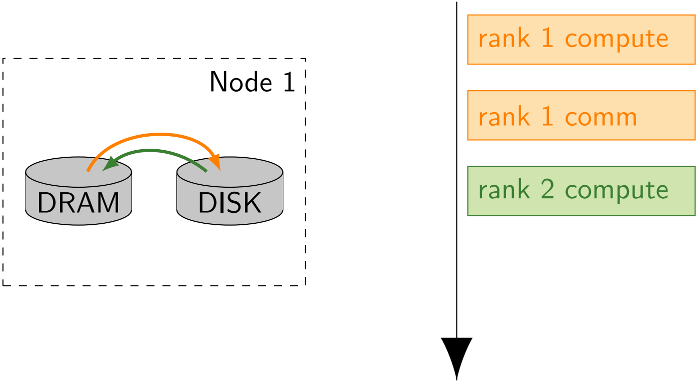
Pitfalls of OS swapping in BDMPI



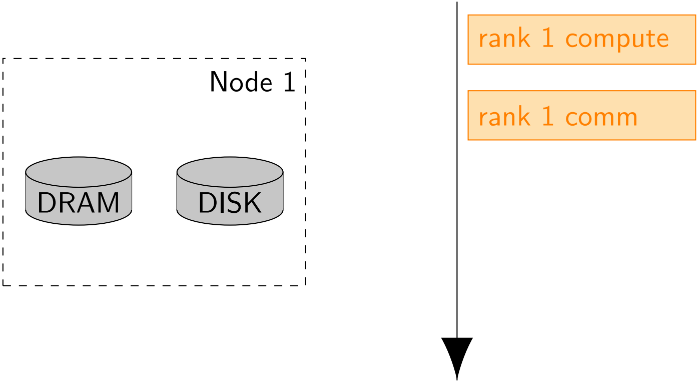
Pitfalls of OS swapping in BDMPI



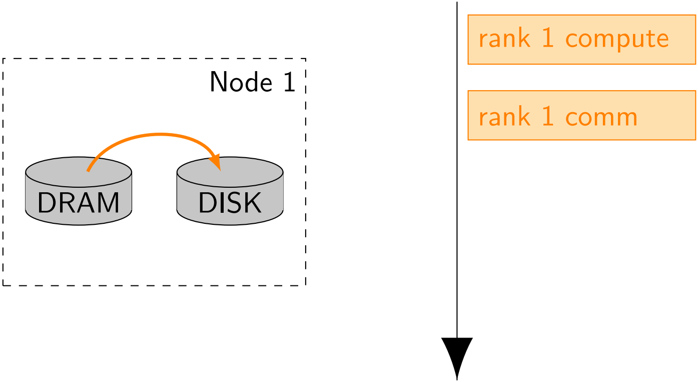
Pitfalls of OS swapping in BDMPI



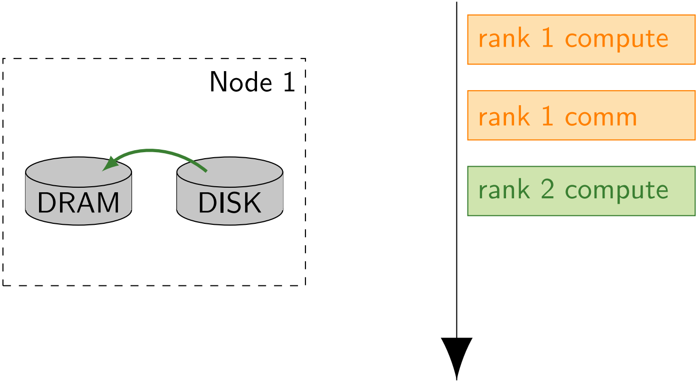
Let's back up. . .



Let's back up...



. . . and reduce disk contention



Important perspective

Hypothesis

- Exploiting the BDMPI memory and execution models will lead to reduced disk contention compared with deferring to the OS VMM

Key question

- How aggressively should a process' virtual address space be exchanged between physical memory and disk to maintain to prevent memory over-subscription?

Important perspective

Hypothesis

- Exploiting the BDMPI memory and execution models will lead to reduced disk contention compared with deferring to the OS VMM

Key question

- How aggressively should a process' virtual address space be exchanged between physical memory and disk to maintain to prevent memory over-subscription?

SBMA Overview

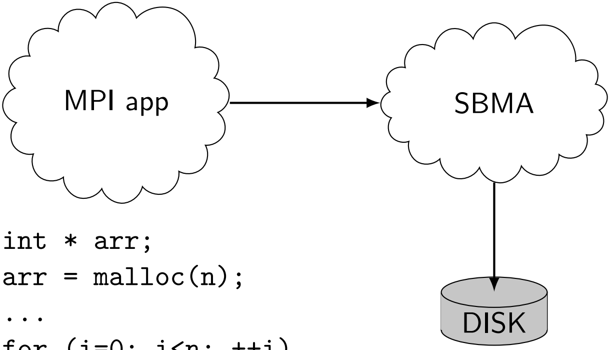
What it is...

- *Storage-Backed Memory Allocation (SBMA)*
- Built as part of the BDMPI library
- User space virtual memory manager

How it works...

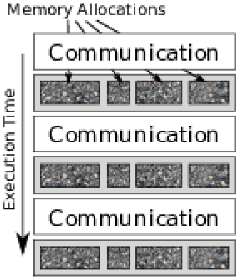
- Uses C interposition to fulfill applications' memory allocation requests
- Relies on memory protection and signal handling to track status of allocated pages

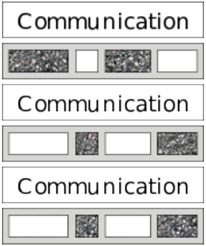
An illustrative example

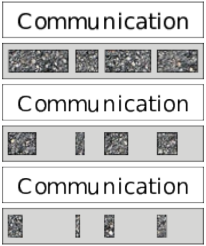


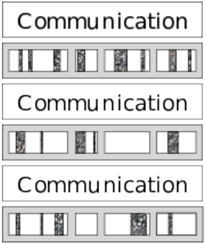

```
int * arr;
arr = malloc(n);
...
for (i=0; i<n; ++i)
  if (!arr[i])
    arr[i] = 1;
...
free(arr);
```

Memory access patterns









Benchmarks

Synthetic

- Sequence of reads and writes
- Used to quantify the overhead introduced by the SBMA library

PageRank

- Memory footprint fixed
- Multiplying a sparse matrix by a vector

ParMetis

- Memory footprint changes throughout execution
- Recursively contracting a graph

SPLATT

- Memory footprint fixed, but has different phases requiring different amounts of memory
- Multiplying a sparse tensor and dense matrices

Experimental setup

System

- Four machine cluster with an aggregate 16GB DRAM and 1.2TB swap

Datasets

- Synthetic - dynamically generated random data (4GB in memory)
- PageRank - 6.6B edges, ordered randomly (35GB in memory)
- ParMetis - 760M edges (13GB in memory)
- SPLATT - 2.9M$\times$2.1M$\times$25.5M with 143.6M non-zeros (26GB in memory)

Synthetic benchmark

	Read (x == y)		Write (x = y)		Read/Write (x += y)	
	OS	SBMA	OS	SBMA	OS	SBMA
AI	1195	1194	514	373	472	352
LI	1195	927	514	325	472	310
AR	28	28	514	373	28	28
LR	30	30	514	325	30	30

Throughput (system pages/sec)

- A Aggressive
- L Lazy
- I In-memory
- R On disk

Synthetic benchmark

	Read (x == y)		Write (x = y)		Read/Write (x += y)	
	OS	SBMA	OS	SBMA	OS	SBMA
AI	**1195**	**1194**	**514**	**373**	**472**	**352**
LI	**1195**	**927**	**514**	**325**	**472**	**310**
AR	28	28	514	373	28	28
LR	30	30	514	325	30	30

Throughput (system pages/sec)

- **A** Aggressive
- **L** Lazy
- **I** In-memory
- **R** On disk

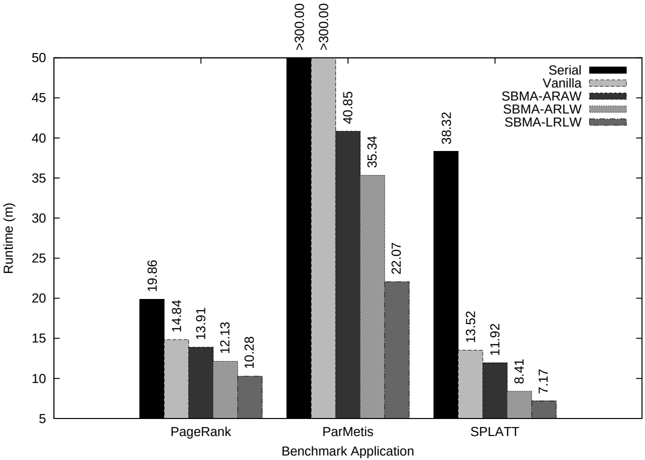
Synthetic benchmark

	Read (x == y)		Write (x = y)		Read/Write (x += y)	
	OS	SBMA	OS	SBMA	OS	SBMA
AI	1195	1194	514	373	472	352
LI	1195	927	514	325	472	310
AR	**28**	**28**	514	373	**28**	**28**
LR	**30**	**30**	514	325	**30**	**30**

Throughput (system pages/sec)

- A Aggressive
- L Lazy
- I In-memory
- R On disk

Real world benchmarks



Conclusions

What we've learned

- Possible to implement a user space virtual memory manager with less a $2\times$ slowdown in memory throughput
- Exploiting BDMPI's execution and memory models improves performance over OS VMM with speedups from $2\times$ to $12\times$

Conclusions

What we've learned

- Possible to implement a user space virtual memory manager with less a $2\times$ slowdown in memory throughput
- Exploiting BDMPI's execution and memory models improves performance over OS VMM with speedups from $2\times$ to $12\times$

Moving forward

- Add support for MPI+X
- Allow more than one process to run simultaneously on each compute node so long as memory constraint is not violated

Thank you

Questions?

jiverson@cs.umn.edu

http://glaros.dtc.umn.edu/gkhome/bdmpi/download