Unleashing dynamic task scheduling at rack-scale

Magnus Norgren,
Andra Hugo (DDN Storage),
Stefanos Kaxiras, Konstantinos Sagonas
Task based programming on distributed shared memory

• Can we scale a task based system at cluster scale?
• ... on a Software DSM ?!
  → Yes, If we co-design scheduling & caching

Early results: Running tasks on remote memory at >90% of native execution on local memory
Distributed Shared Memory

- Coherent Distributed Shared Memory (DSM) gives us:
  - Transparent Caching (takes advantage of any locality)
    - Even if locality is tiny, its benefit is a function of latency!
    - Avoids hybrid SM (intranode)-MP (internode) programming
  - Freedom from data distribution
    - Demand fetching, prefetching can largely hide bad distribution
  - Natural and familiar synchronization constructs (locks)
  - Extend memory beyond one node

- But isn’t this quite costly?
  - Yes – Used to be, but times have changed
    - Latency only one magnitude worse than DRAM
    - ...And bandwidth is on par with DRAM
Trends: Latency
Memory vs. Network

From 3 magnitudes to 1 magnitude difference
Trends: Bandwidth Memory vs. Network

Bandwidth on par with DRAM
→ We can trade bandwidth for latency, fetching data in large granularity, prefetching etc.
ARGO DSM
(www.argodsm.com)

• A DSM based on current trends [HPDC’15]:
  • User-space implementation
  • Page-based DSM (uses virtual memory faults for misses)
  • MPI is the “network layer” (but only need RDMA)
• Runs
  • Pthreads – Data race free programs
  • OpenMP - Manually ported (Minor modifications)
  • Compile and link with ArgoDSM library ➔ MPI program that implements DSM
ARGO DSM  
(www.argodsm.com)

• Under the hood [HPDC’15]
  • SC for DRF (Sequential Consistency for Data Race Free)
    • Full relaxation of memory ordering between synchronization
  • Builds on very simple, completely distributed coherence (VIPS) [PACT’15, ISCA’13]
  • Queue Delegation Locking primitives [IEEE TPDS’18]

• Take decisions locally & Trade bandwidth for latency
Benchmark scaling
Pthread / OpenMP on ArgoDSM

Parsec Blackscholes

NAS EP

NAS CG

N-body
Benchmark scaling
Pthread / OpenMP on ArgoDSM

Parallel Pthread/OpenMP programs scaled up to 2000 threads

More information, see our HPDC‘15 paper
Task based programming

• Parallel programming model
  • StarPU, OmpSs, OpenMP, etc

• Express parallelism in terms of task-graphs
  • Can target heterogeneous systems
    • Our use case only runs on CPUs
  • High performance
    • Use kernels from BLAS libraries

• Challenge: How to do scheduling?
  • Eager
  • Data aware
  • Custom...

Couple with ArgoDSM for cluster awareness!
Task schedulers

Eager Scheduling

Push

Pop

Try to minimize time to completion

Takes data transfer into consideration

Data Aware Scheduling

Push

CPU #1  CPU #2  CPU #3  CPU #4  CPU #5

CPU #1  CPU #2  CPU #3  CPU #4  CPU #5

= Tasks
Task schedulers

Eager Scheduling

Try to minimize
time
to completion

Takes data transfer
into consideration

Data Aware Scheduling

= Tasks
The perfect fit
ArgoDSM+Tasks

- Tasks over ArgoDSM Co-design
  - ArgoDSM can
    - Handle caching of remote data making it appear as local
The perfect fit
ArgoDSM+Tasks

• Tasks over ArgoDSM Co-design
  • ArgoDSM can
    • Handle caching of remote data making it appear as local
The perfect fit
Cluster level DSM+Tasks

• Tasks over ArgoDSM Co-design
  • ArgoDSM can
    • Handle caching of remote data making it appear as local
    • Exploit task information
      • Prefetch data ahead of time
      • Prevent eviction of useful data
  • StarPU: performance models
    • ArgoDSM (cluster) locality!
ArgoDSM aware scheduler

Eager Scheduling

Push

Pop

Data Aware Scheduling

Push

Try to minimize time to completion

Takes data transfer into consideration

= Tasks
ArgoDSM aware scheduler

Eager Scheduling

No opportunity to optimize eager scheduling for cluster memory
- Discarded in favor of data aware scheduling

Push

Data Aware Scheduling

Try to minimize time to completion
Takes data transfer into consideration

Push

= Tasks

CPU #1  #2  #3  #4  #5

CPU #1  #2  #3  #4  #5
ArgoDSM aware scheduler

Try to minimize time to completion
Takes data transfer into consideration

Data Aware Scheduling

Push

= Tasks
ArgoDSM aware scheduler
Data Aware
ArgoDSM aware scheduler
Data Aware

• Can we do something clever?

= Tasks

Different cores/devices in the same node
ArgoDSM aware scheduler
Data Aware

• Can we do something clever?
  • Take locality into account

Data Aware Scheduling

Tasks Contain Cached Data
→ No network overhead

Different cores/devices in the same node

= Tasks
= Cached
ArgoDSM aware scheduler
Data Aware

• Can we do something clever?
  • Take locality into account

Data Aware Scheduling

Push

Tasks Contain Cached Data → No network overhead

= Tasks  = Cached

Different cores/devices in the same node
ArgoDSM aware scheduler
Data Aware

• Can we do something clever?
  • Take locality into account

Tasks Contain Cached Data → No network overhead

Different cores/devices in the same node

= Tasks = Cached
ArgoDSM aware scheduler
Data Aware

• Can we do something clever?
  • Take locality into account

Data Aware Scheduling
Push

Tasks Contain Cached Data
→ No network overhead

Better runtime prediction adapted to cluster memory
→ Higher performance

= Tasks  = Cached

Different cores/devices in the same node
ArgoDSM aware scheduler
Local First
ArgoDSM aware scheduler
Local First

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier

= Tasks  = Cached

Different cores/devices in the same node
ArgoDSM aware scheduler
Local First

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier

= Tasks  = Cached

Different cores/devices in the same node
ArgoDSM aware scheduler
Sorted Insert
ArgoDSM aware scheduler

Sorted

• Can we do something clever?
  • Take locality into account
  • Schedule cached tasks earlier

Data Aware Scheduling

Push

Only partially cached

Different cores/devices in the same node

= Tasks  = Cached  = Partially Cached
ArgoDSM aware scheduler

• Can we do something clever?
  • Take locality into account
  • Schedule cached tasks earlier
  • Sorted insert
    • Favor fully cached tasks

• Behind the scenes
  • Prefetch tasks
  • Separate thread iterating the queues

不同内核/设备在同一个节点

Data Aware Scheduling

Push

CPU #1
CPU #2
CPU #3
CPU #4
CPU #5

Different cores/devices in the same node

= Tasks  = Cached  = Partially Cached
ArgoDSM aware scheduler

- Can we do something clever?
  - Take locality into account
  - Schedule cached tasks earlier
  - Sorted insert
    - Favor fully cached tasks

- Behind the scenes
  - Prefetch tasks
  - Separate thread iterating the queues

= Tasks = Cached = Partially Cached

Data Aware Scheduling
Push

Different cores/devices in the same node
Cholesky

- Coarse grain data transfer make use of BW better
  - 1800 pages per task buffer
  - >32 pages data transfer unit to get performance

Cholesky 67k (36GB), Blocksize: 960, Cachesize: 10GB
**Cholesky**

- Data aware – increases performance by 10%

![Graph showing performance normalization for Cholesky 67k (36GB), Blocksize: 960, Cachesize: 10GB](image)

- Eager
- Data Aware

Normalized to native performance vs Data Transfer Size (Pages)
Cholesky

• *Sorted* - Inserts tasks based on how much is cached
  • Increase performance another ~10%

![Graph showing performance normalized to native performance vs data transfer size.](image-url)
Cholesky

• *Local First* - Any cached task is always placed first
  • Small chance of data evicted when executed
  • Increases performance up 90%+ of performance

Cholesky 67k (36GB), Blocksize: 960, Cachesize: 10GB
Cholesky

- Coarse grain data transfer make use of BW better
  - 1800 pages per task buffer
  - 32 pages data transfer unit get performance

Cholesky 67k (36GB), Blocksize: 960, Cachesize: 10GB
LU 76k (45GB), Blocksize 960, Data transfer size 256 pages

Scaling cachesize

- Data Aware
- Local First
- Sorted
- Eager

Normalized to native performance

Scaling from 40% - 60%
Small difference between schedulers

What about prefetching?
LU

LU 76k (45GB), Blocksize 960, Data transfer size 256 pages
Scaling cachesize

Normalized to native performance

- Data Aware Prefetch
- Data Aware
- Local First Prefetch
- Local First
- Sorted Prefetch
- Sorted
- Eager

Prefetching really does well!
Up to 90% of native performance

Prefetching

No Prefetching

30%
The next step – Scale computation

• Using our method we have 2 degrees of freedom
  • Move data or tasks (EPEEC – Horizon 2020)
• Optimize at cluster level
  • How to divide up work?
    • Cluster global work queues?
    • Open question: When do we synchronize?
  • Give the scheduler locality information to improve scheduling
  • Give the DSM task information
    • Similar to this talk, memory is the important part
Conclusions

• Task-based systems are a great fit for DSMs
Conclusions

• DSMs are a great fit for a Task based systems
  • ArgoDSM can make use of efficient caching and prefetching based on scheduler information
  • Scheduling policies can make use of DSM information to schedule for locality

• As advertised:
  • Up to 90%+ of native performance operating on remote memory using a co-design of scheduler and DSM caching techniques
Thank you!
Questions?

Magnus.Norgren@it.uu.se
www.argodsm.com
Resources and References

• ArgoDSM
  • Project site (and the HPDC’15 paper)
    • http://www.it.uu.se/research/project/argo
    • http://www.argodsm.com
    • https://dl.acm.org/citation.cfm?id=2749250

• VIPS Coherence
  • https://dl.acm.org/citation.cfm?doid=2370816.2370853
  • http://ieeexplore.ieee.org/document/6398353/?reload=true

• Queue-Delegation Locking
  • http://www.it.uu.se/research/group/languages/software/qd_lock_lib

• StarPU
  • http://starpu.gforge.inria.fr
Extra Slides –
Scaling ONLY MEMORY
Python example

• Python-based spectral filter application from meteorology (very large data set using SciPy)
  • 8 Nodes for memory, one node for compute
  • Outperforms swap

50 GB Spectral Filtering

no more RAM, swap begins
Extra Slides - Bandwidth

ArgoDSM memcpy cs MPI-RMA

Bytes

MB/s
Extra Slides – HW Specs

• Compute per node
  • Two Intel Xeon E5 2630 v4, 2.20Ghz (10 cores, 20 threads)

• Network
  • Infiniband FDR
    • Latency 0.7us
    • Bandwidth 56GB/s
Extra Slide - Eviction Policies

- Eviction Policy - Direct Mapped, Random, ’least recently mapped’=FIFO

Cholesky 67k (36GB), Blocksize: 960
256 Cache Entry Granularity