Grid Computing

11M37152
Takafumi Saito

Reference

- Title:“Supporting GPU Sharing in Cloud Environments with a Transparent Runtime Consolidation Framework”
- Writer : Vignesh T. Ravi, Michela Becchi, Gagan Agrawal, Srimat Chakradhar
- Conference : HPDC ’11 Best Paper Award

Outline

- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

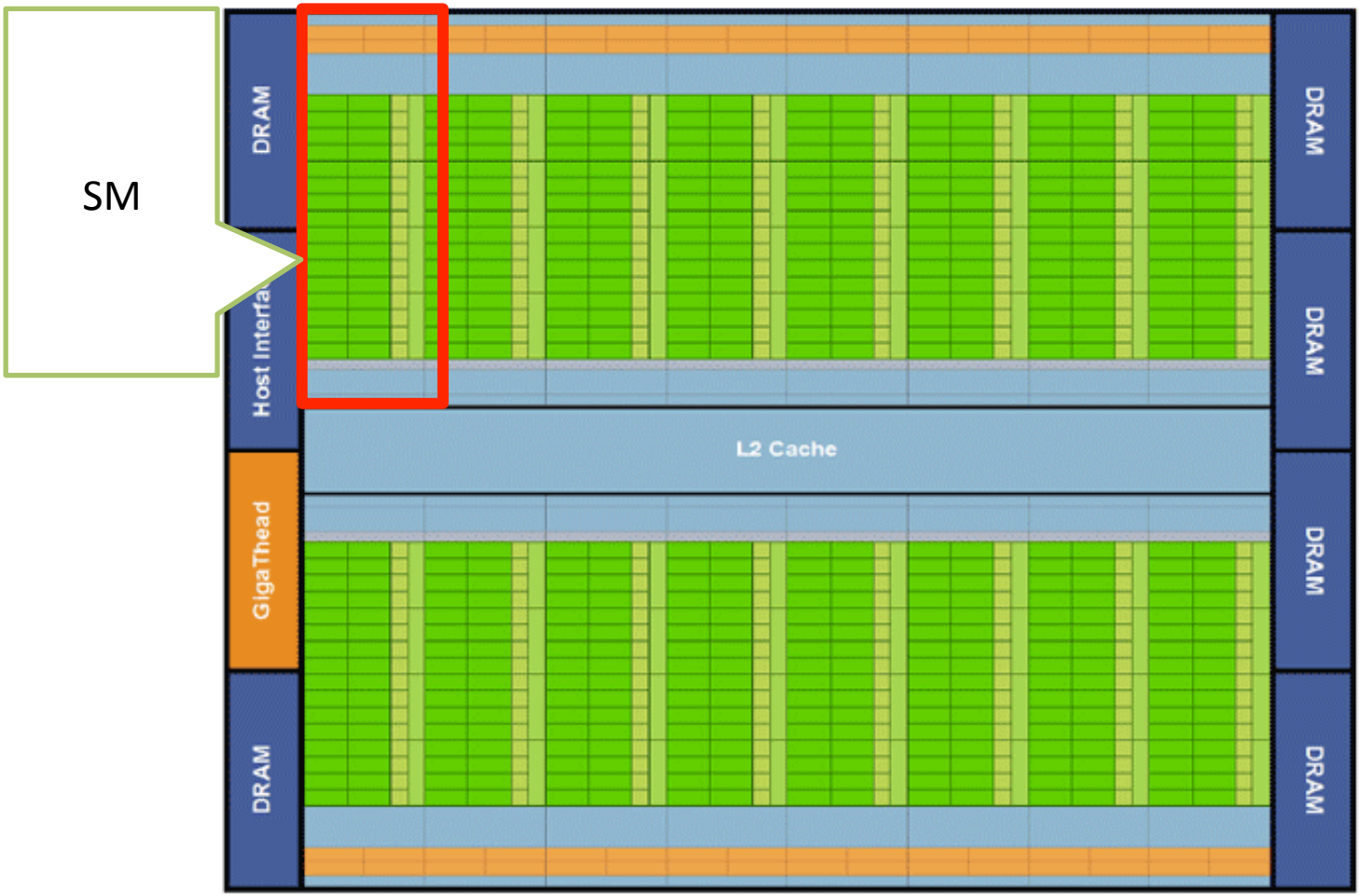
Outline

- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

GPU

- Graphics Processing Units
- GPGPU(General-Purpose GPU)
  - Popular in high performance computing
  - Advantage
    - Extreme-scale
    - Cost-effective
    - Power-effective

GPU architecture



Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).

Background

- Cloud Environments
  - “pay-as-you-go”
  - Easy to use up-to-data resources
  - no need to maintenance for users

→ GPU on cloud will be important in HPC

Purpose

- Make GPU shared resource in the cloud
- Motivation
  - Multiple VMs sharing single GPU on multi-core node
  - Good cost performance
  - Utilization of high degree of parallelism

Outline

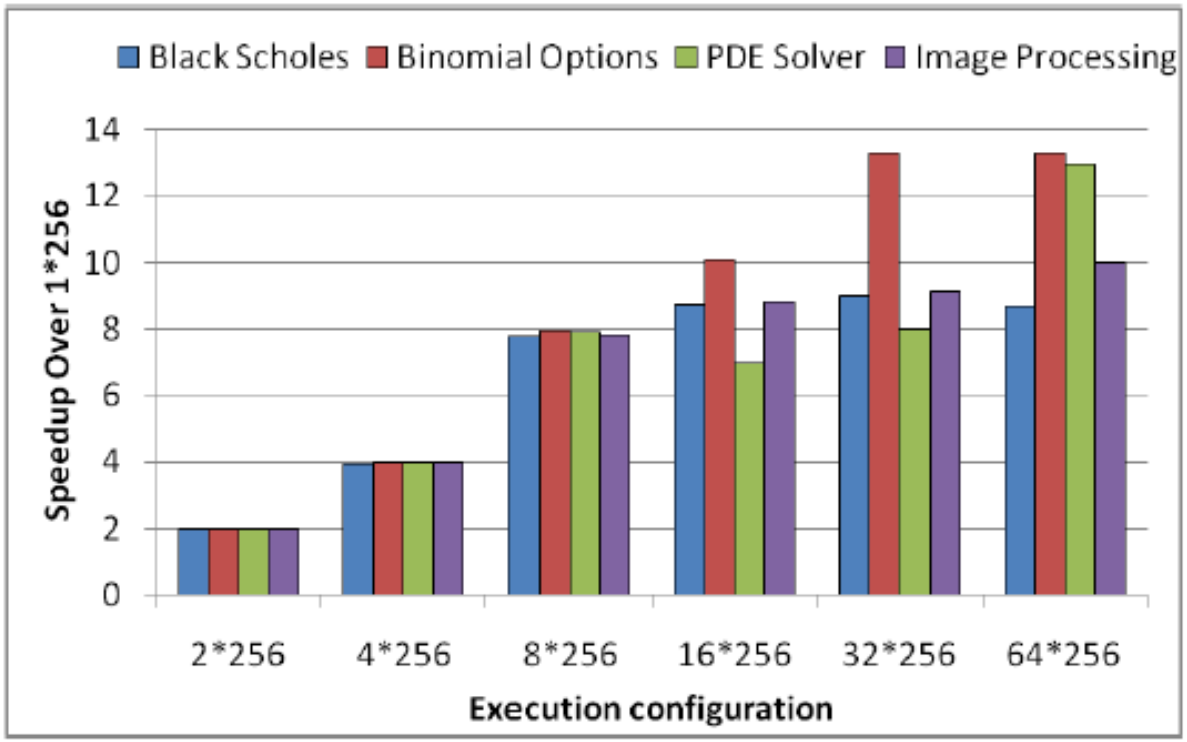
- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

Approach

- Framework to enable applications to share GPUs
- Contributions
  - Extensions of GPU virtual software for consolidation
  - Solutions to the conceptual problem of consolidation
    - Affinity score
    - molding

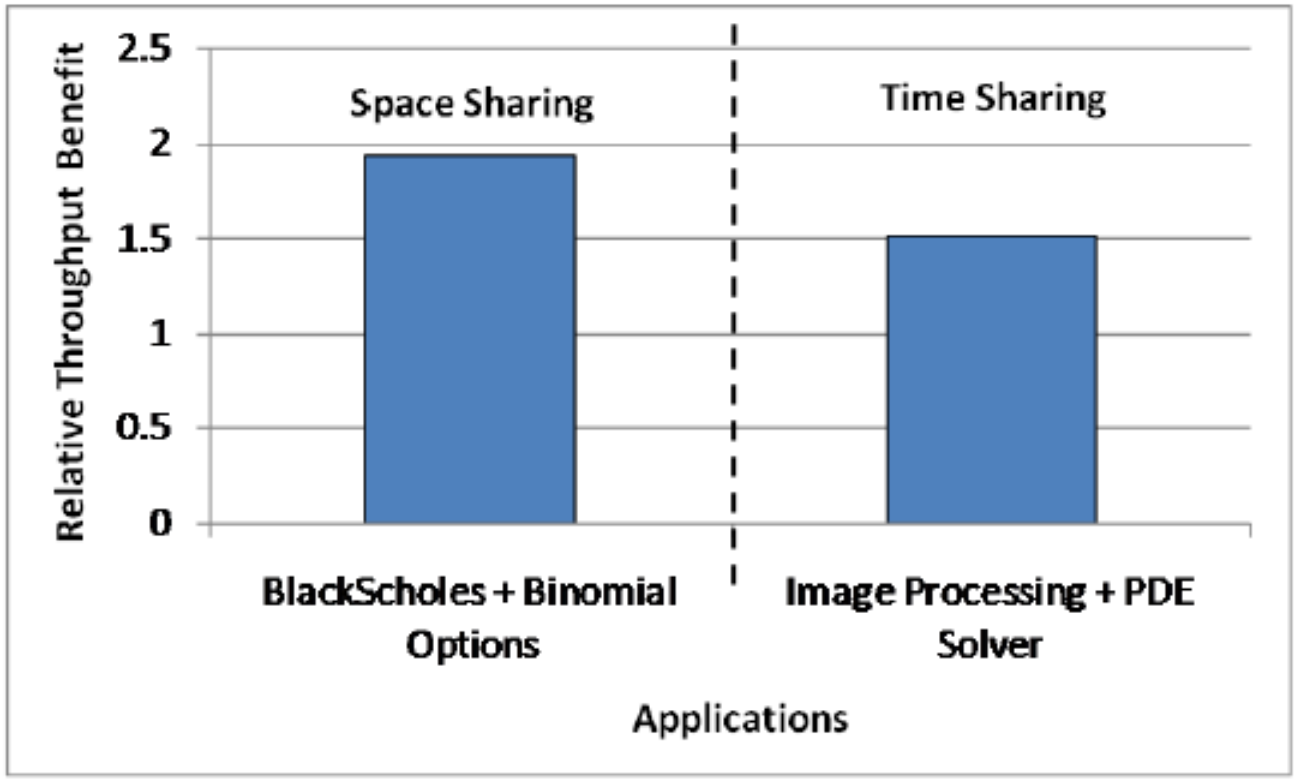
Relationship between resource utilization and performance

- Three applications scale until 8 thread blocks
  - Because of the number of thread blocks exceeds the number of SMs
- Image Processing gets better performance over 8 thread blocks
  - Low-overhead context switching mask memory latencies



Evaluation experiment in consolidation

- GPU Sharing
  - Space Sharing : assign subset of SMs to each of kernels
  - Time Sharing : time-share SMs among kernels



Outline

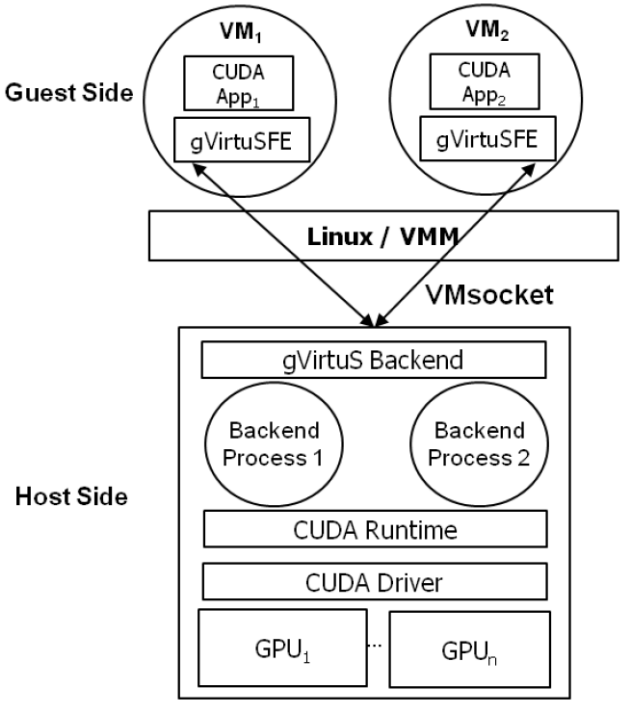
- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

Design Challenge

- How to Enable Sharing of GPU(s) across Different Applications?
  - create a virtual process context
- What and How to Consolidate?
  - Use information for consolidation decision
- How to Achieve a Low Overhead Design?
  - Overhead of virtual machine on GPU is low
  - Overhead of virtual process context and consolidation decisions must be kept low

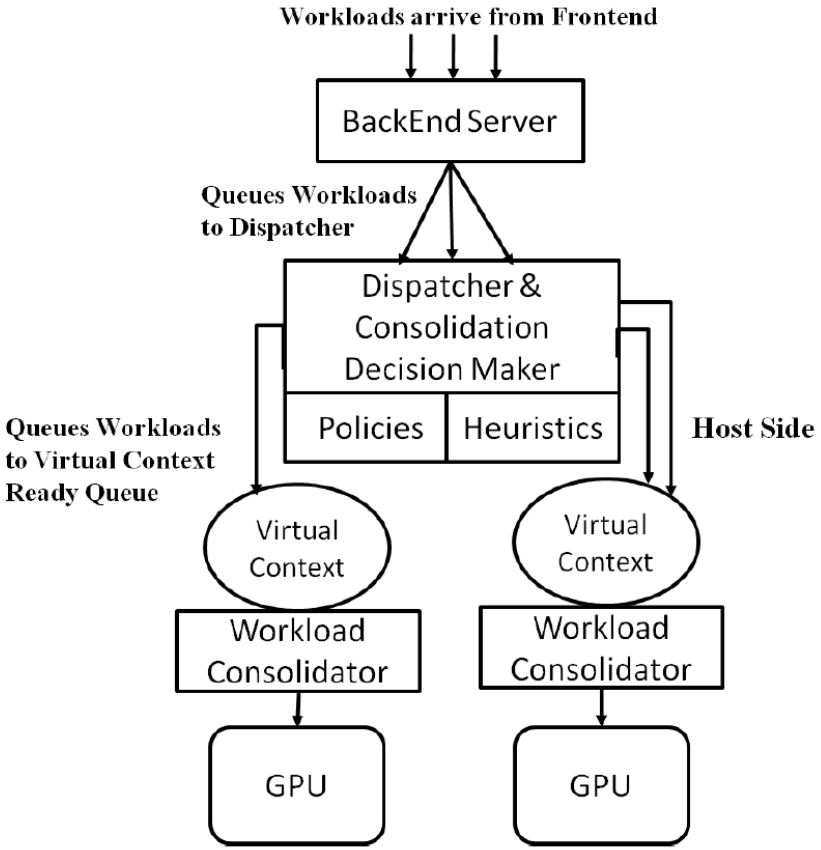
gVirtuS Current Design

- gVirtuS
  - Open source virtual machine to run CUDA-enabled applications
- gVirtus compornent
  - Frontend library : intercept CUDA call and redirect to backend
  - Backend daemon : transfar



Runtime Consolidation Framework

- Improved Backend
  - DCDM(Dispatcher and Consolidation Decision Maker)
  - VCC(Virtual Context with Consolidation)



DCDM & VCC

- DCDM
  - Reads execution configuration of the kernels
  - Changes into information to decide consolidation benefits
- VCC
  - Creates a virtual context for each GPUs

Design Issues and Limitations

- No use constant memory and texture memory

Outline

- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

Causes of Resource Contention

- SMs
  - If the sum of thread blocks is more than the number of available SMs
- Shared Memory
  - If aggregated shared memory requirements exceed the amount of available shared memory

→ mold the number of thread blocks and/or the number of threads to avoid contentions

Policies of Molding

- Forced Space Sharing
  - Reduce the number of thread blocks
  - Increase the number of threads of each block
- Time Sharing with Reduced Threads
  - Reduce the number of threads of each block

Consolidation Algorithm

- Consolidate N given kernels on 2 GPUs

Algorithm 1 Runtime Consolidation Scheduling Algorithm

```
1:  Configuration List of all N Kernels
2:  WQ1 = φ, WQ2 = φ
3:  A[][] = GeneratePairwiseAffinity(K)
4:  [ki,kj] = FindMinAffinityPair(A[][])
5:  Push ki into WQ1
6:  Push kj into WQ2
7:  for all Kernels K - ki and kj do
8:      kl = GetNextKernel()
9:      a1 = GetAffinityForList(ConfigList(kl, WQ1))
10:     a2 = GetAffinityForList(ConfigList(kl, WQ2))
11:     [a3,NewConfigSet] = GetAffinityByMolding(ConfigList(kl,
        WQ1))
12:     [a4,NewConfigSet] = GetAffinityByMolding(ConfigList(kl,
        WQ2))
13:     if MaxAffinity(a1) then
14:         Push kl into WQ1
15:     else if MaxAffinity(a2) then
16:         Push kl into WQ2
17:     else if MaxAffinity(a3) then
18:         Apply NewConfigSet to kl and WQ1
19:         Push kl into WQ1
20:     else
21:         Apply NewConfigSet to kl and WQ2
22:         Push kl into WQ2
23:     end if
24: end for
25: Dispatch WQ1 to Virtual Context1 for consolidation
26: Dispatch WQ2 to Virtual Context2 for consolidation
```

Pair-wise Affinity Score

- Compute affinity score for set of kernels

Algorithm 2 Generate Pairwise Affinity

```
1:  Input: Kernel Configuration set K_1...K_N
2:  for i = 1 to N do
3:      for j = 1 to N do
4:          if i ≠ j and j > i then
5:              if SpaceSharing(k_i, k_j) then
6:                  Affinity[i,j] = 1
7:              else
8:                  if SHMEM_i + SHMEM_j > MAXSHMEM then
9:                      Affinity[i,j] = 0
10:                 else
11:                     Affinity[i,j]=1-(THREADS_i+THREADS_j)/1000
12:                 end if
13:             end if
14:         end if
15:     end for
16: end for
17: Return Affinity[i,j]
```

Affinity Score with WQ

- there is shared memory contention
  - Convert time sharing to space sharing
- the sum of threads of kernels is large
  - Change to suitable molding configuration

Algorithm 3 Get Affinity By Molding

```
1:  Input: Configuration of Next Kernel to Schedule, k_l
           Configuration of list of kernels in WQ
2:  V[] = FindKernelSetViolatingSHMEM(k_l, WQ)
3:  if nonempty(V[]) then
4:      NewConfigList.Append(ConvertToSpaceSharing(V[]))
5:  end if
6:  for each remaining Time Sharing kernel k in WQ do
7:      if TotThreads is large then
8:          NewConfigList.Append(FindConfigForMold(k))
9:      end if
10: end for
11: Affinity = GetAffinityForList(NewConfigList)
12: Return [Affinity, NewConfigSet]
```

Outline

- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

Evaluation

- Setup
  - CPU : 2 Intel Xeon E5520
  - Memory : 4GB
  - GPU : 2 Nvidia Tesla C2050
    - 14 SMs (shared memory : 48KB)
    - 32 cores (1.15GHz) per SM
    - Device memory : 3GB
  - gVirtuS : version 2.0

Evaluation

- Benchmarks
  - 8 benchmark applications

- Low Shared Memory : ~3KB
- Median Shared Memory : ~16KB
- Heavy Shared Memory : ~48KB

Benchmarks	Memory characteristics	Data Set Description
Image Processing(IP)	No Shared Memory	2 * 3584 * 3584 points
PDE Solvers (PDE)	No Shared Memory	2 * 3584 * 3584 points
BlackScholes(BS)	No Shared Memory	1.000.000 options
Binomial Options(BO)	Low Shared Memory	256 options, 2048 steps
K-means Clustering(KM)	Median Shared Memory	4.194.304 points
K-Nearest Neighbor (KNN)	Median Shared Memory	4.194.304 points
Euler(EU)	Heavy Shared Memory	10.000 nodes, 60.000 edges
Molecular Dynamics(MD)	Heavy Shared Memory	130.000 nodes, 16.200.000edges

Evaluation of basic policy

- Evaluations of space sharing and time sharing
  - Blind consolidation policy
  - Applications mapped to GPU in round robin fashion

Space Sharing

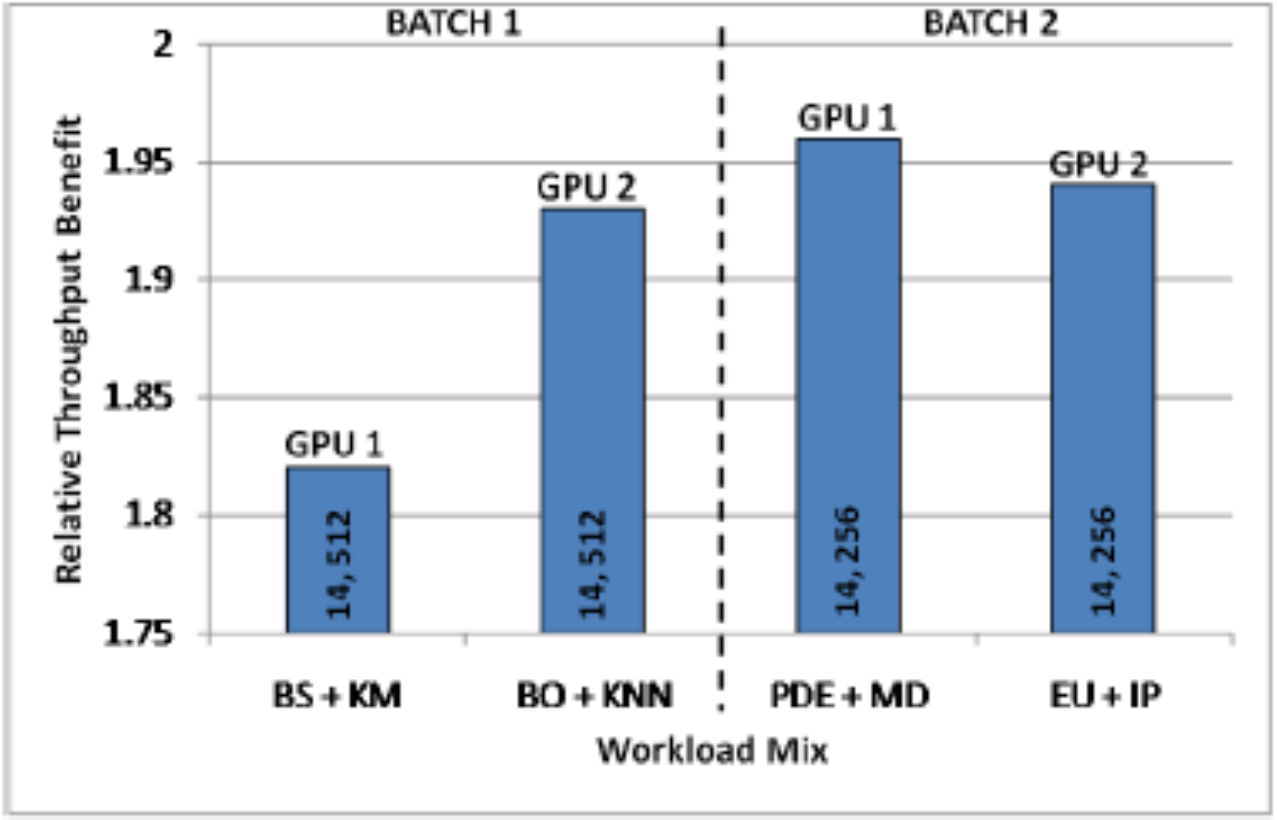


Figure 7: Throughput Benefits from Space Sharing

Time Sharing

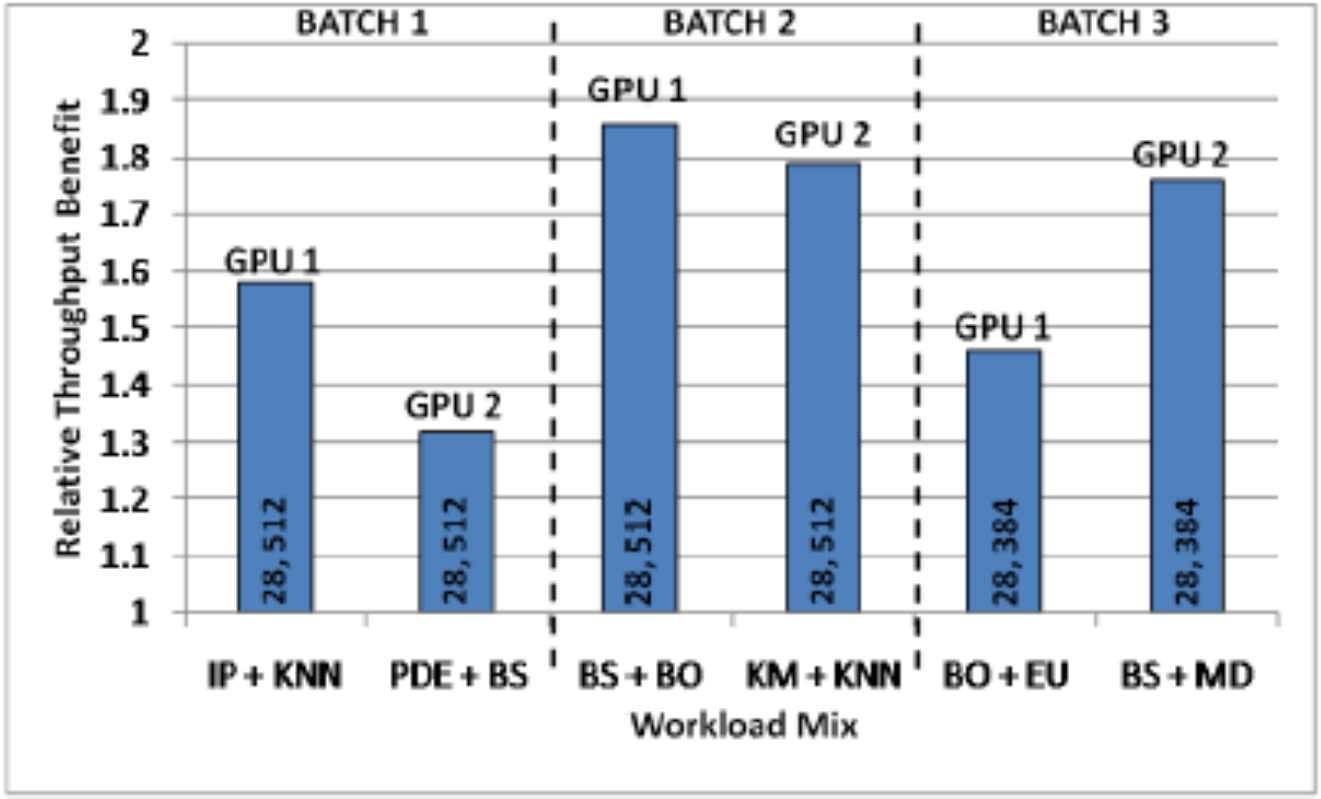


Figure 8: Throughput Benefits from Time Sharing

Impact of Contention on Basic Policies

- Contention occur on basic policies when …
  - the number of threads per SM is large
    - batch1 and batch2
  - resource requirements exceed the SM availability
    - batch3

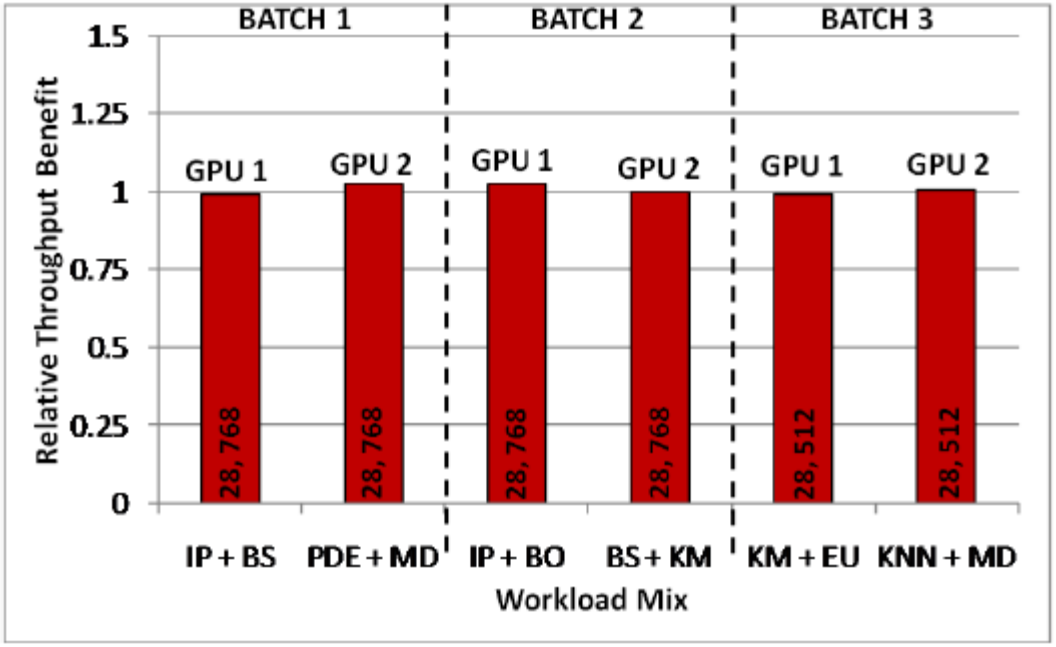


Figure 9: Drawbacks of Basic Policies

Impact of Contention on Basic Policies

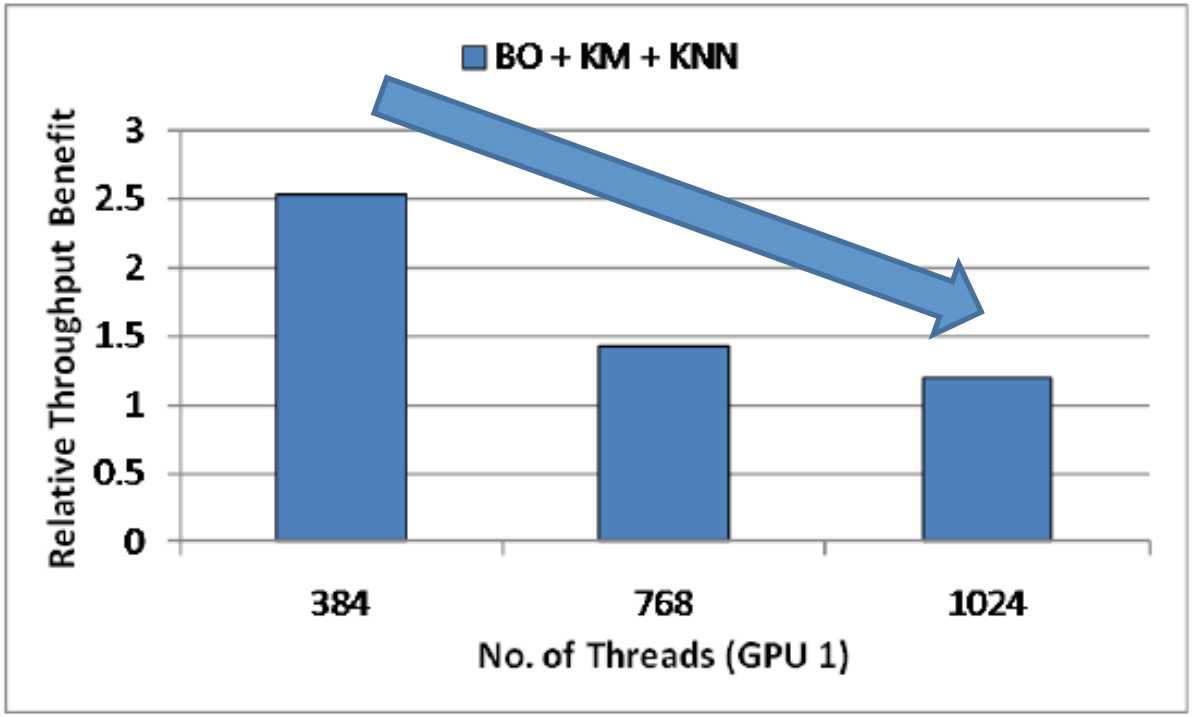


Figure 10: Impact of Large Threads on Throughput

Impact of affinity Score

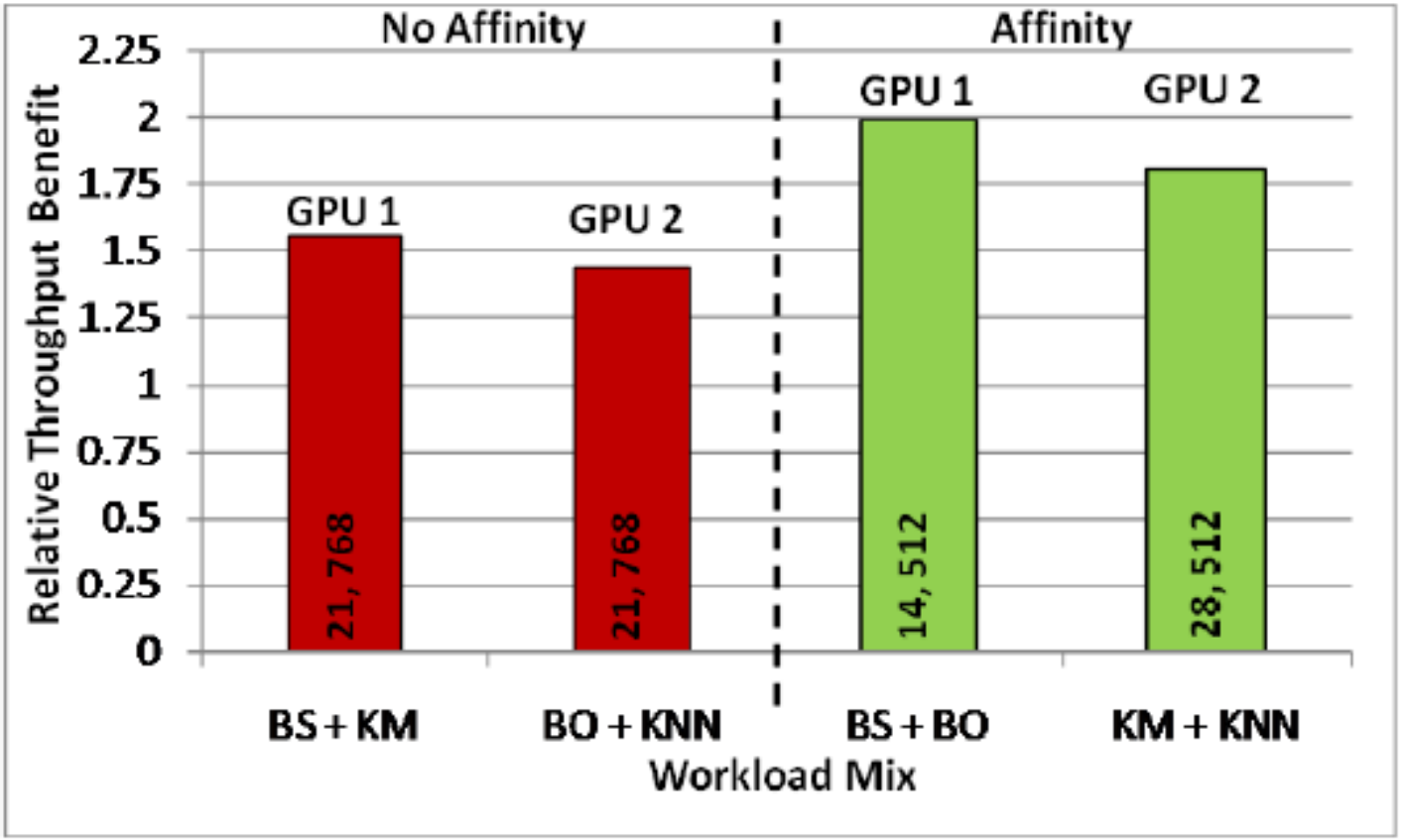
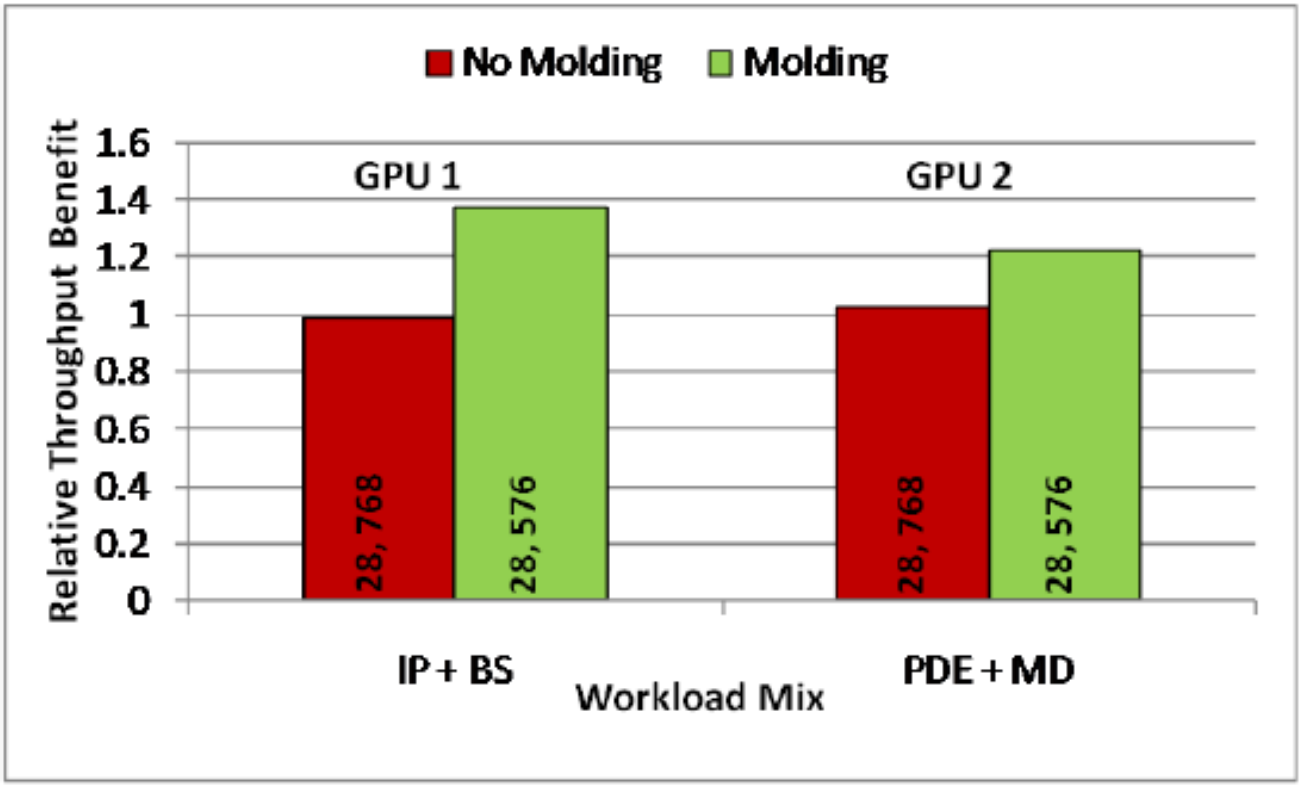


Figure 11: Effect of Affinity Scores on Throughput

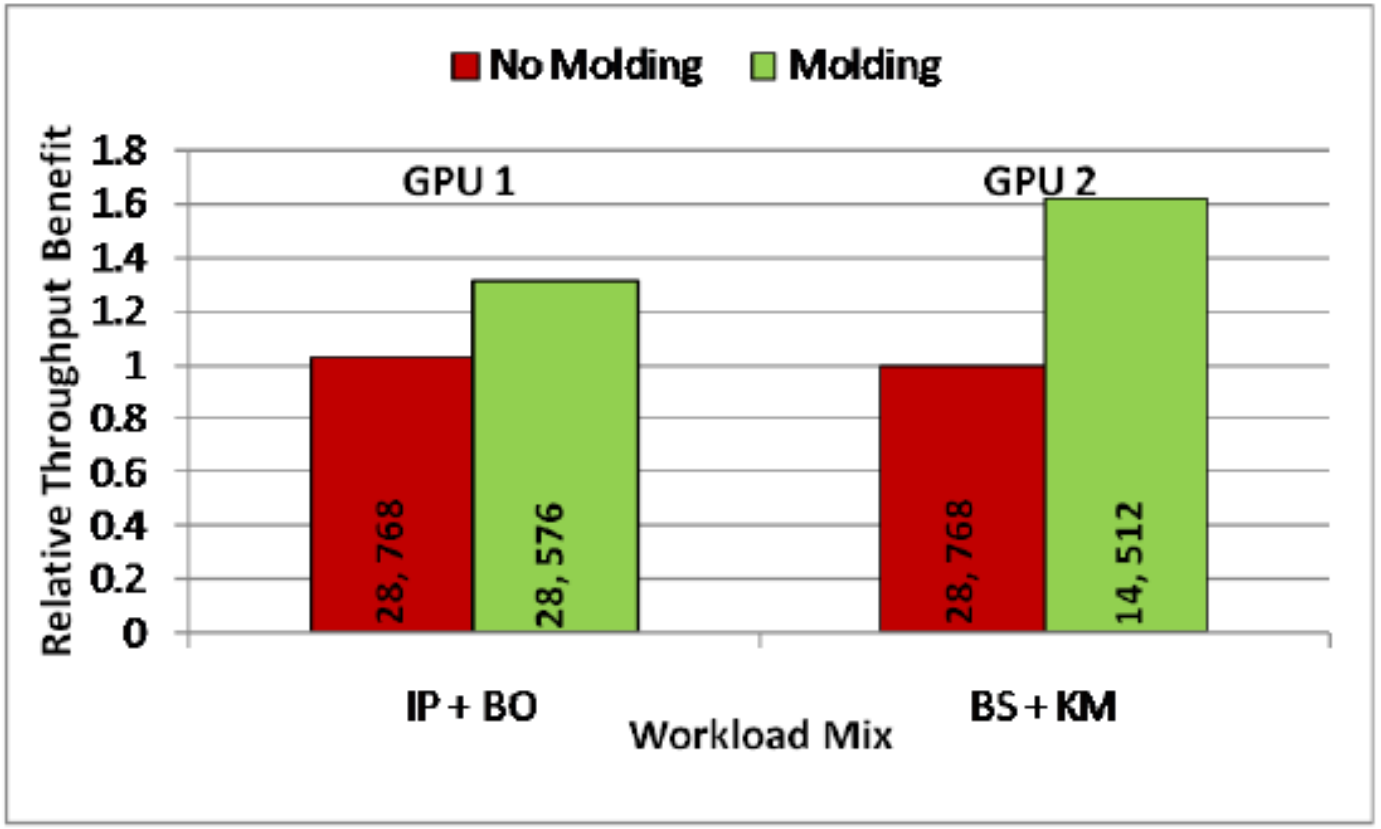
Impact of molding for threads

- One application per GPU is molded
  - IP and PDE



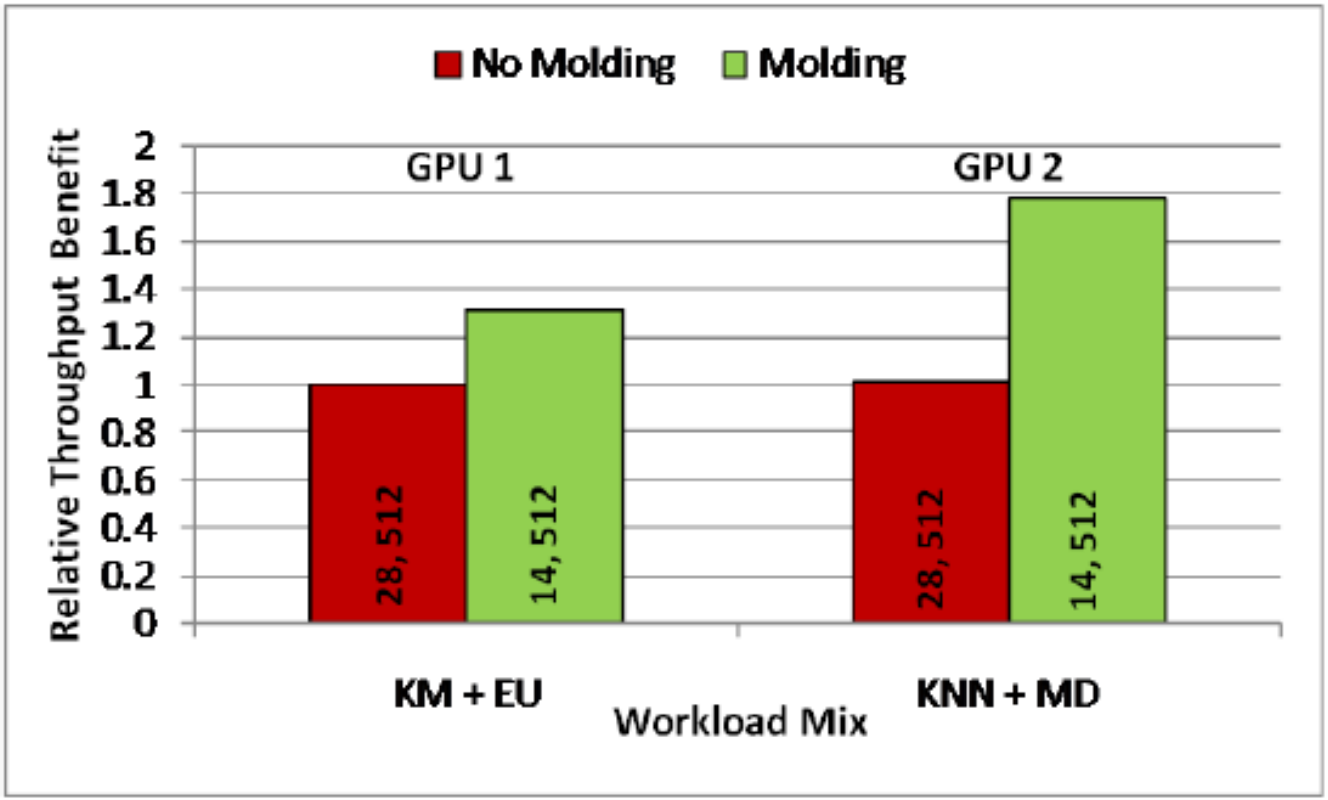
Impact of molding for shared memory

- **Time Sharing with Reduced Threads** is performed on GPU1
- **Forced Space Sharing** is performed on GPU2



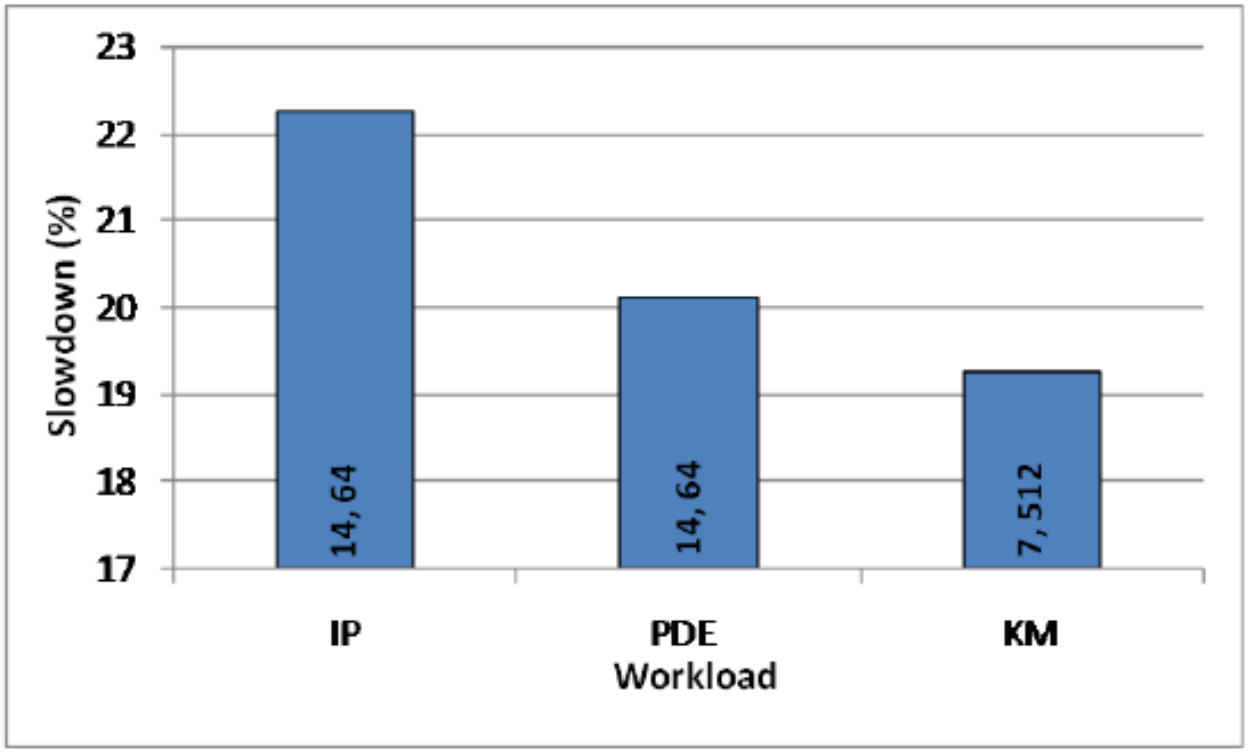
Molding

- Using Forced Space Sharing to avoid shared memory contention

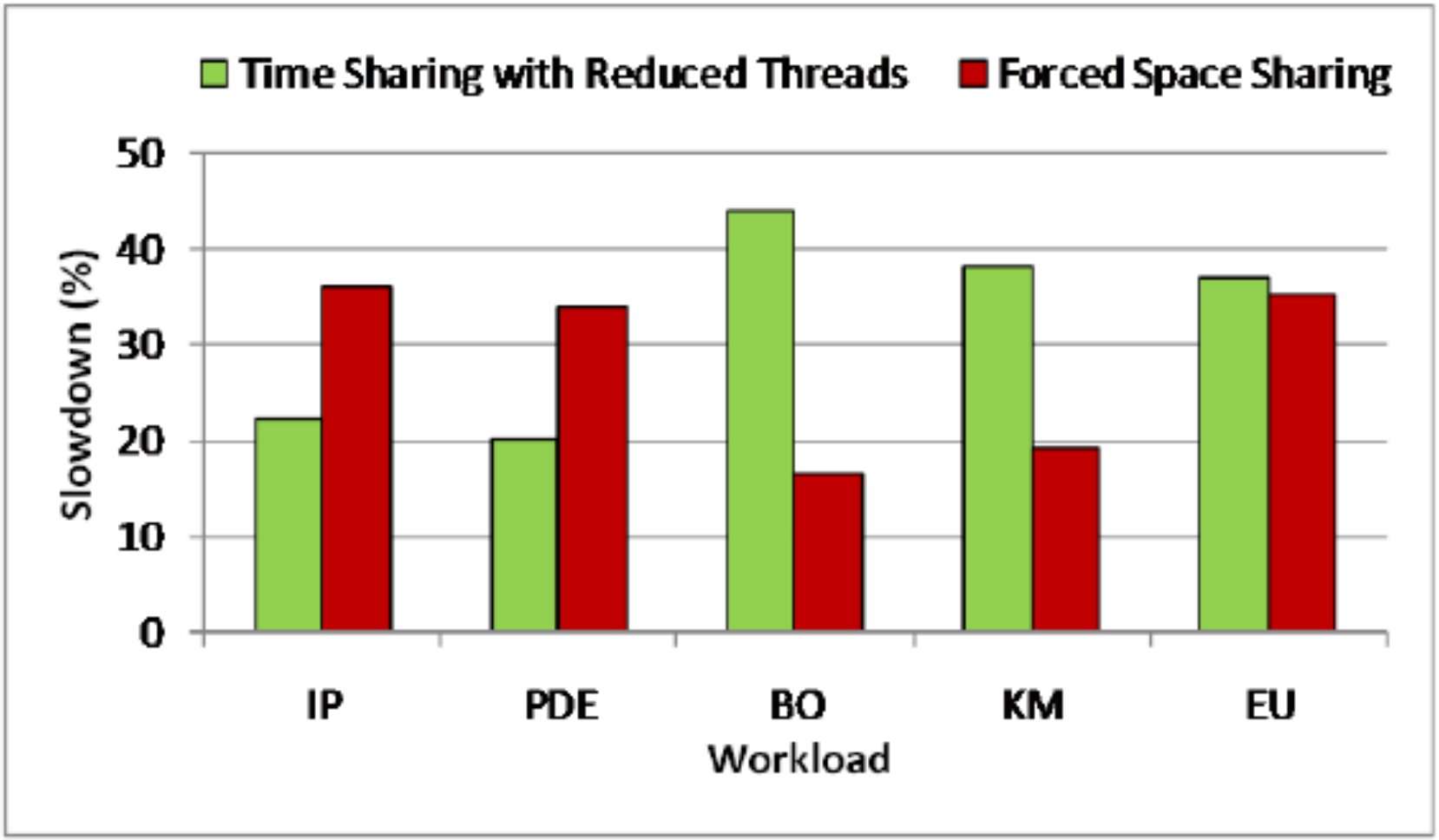


Impact of Molding on Applications

- Case of reducing the number of thread
  - IP, PDE, and KM
  - They have noticeable loss in performance

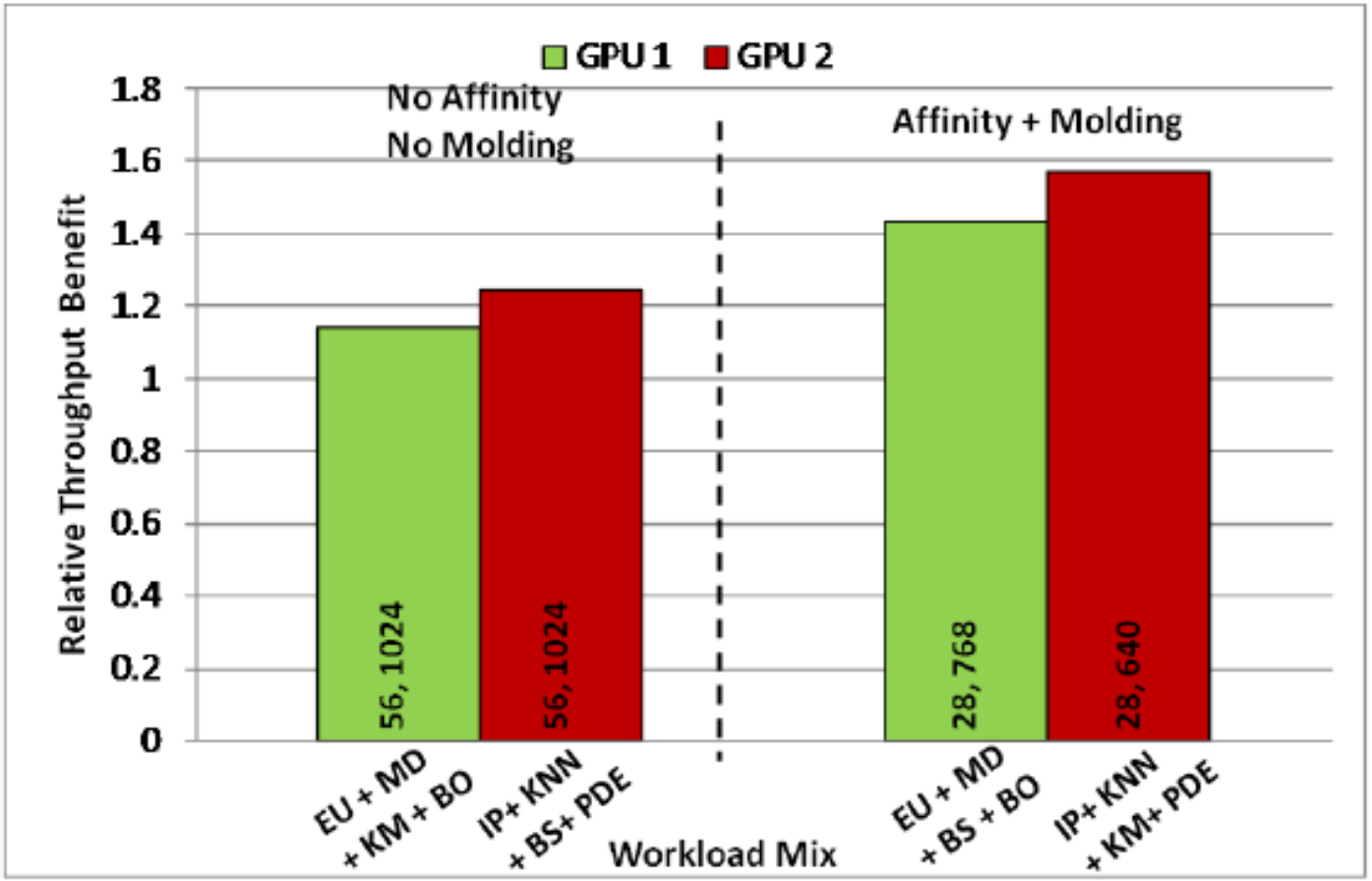


Choice of Molding Type



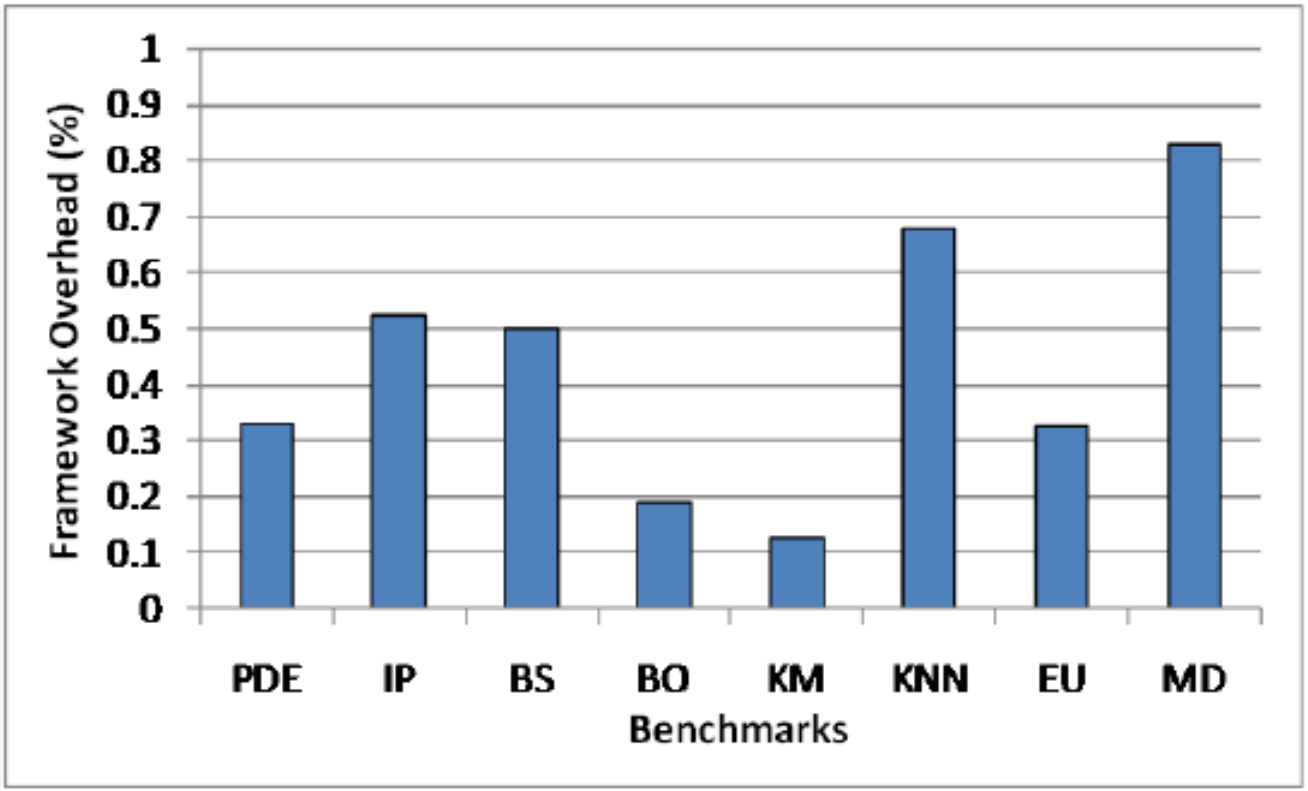
Impact of High Contention

- Eight applications are consolidated on two GPU



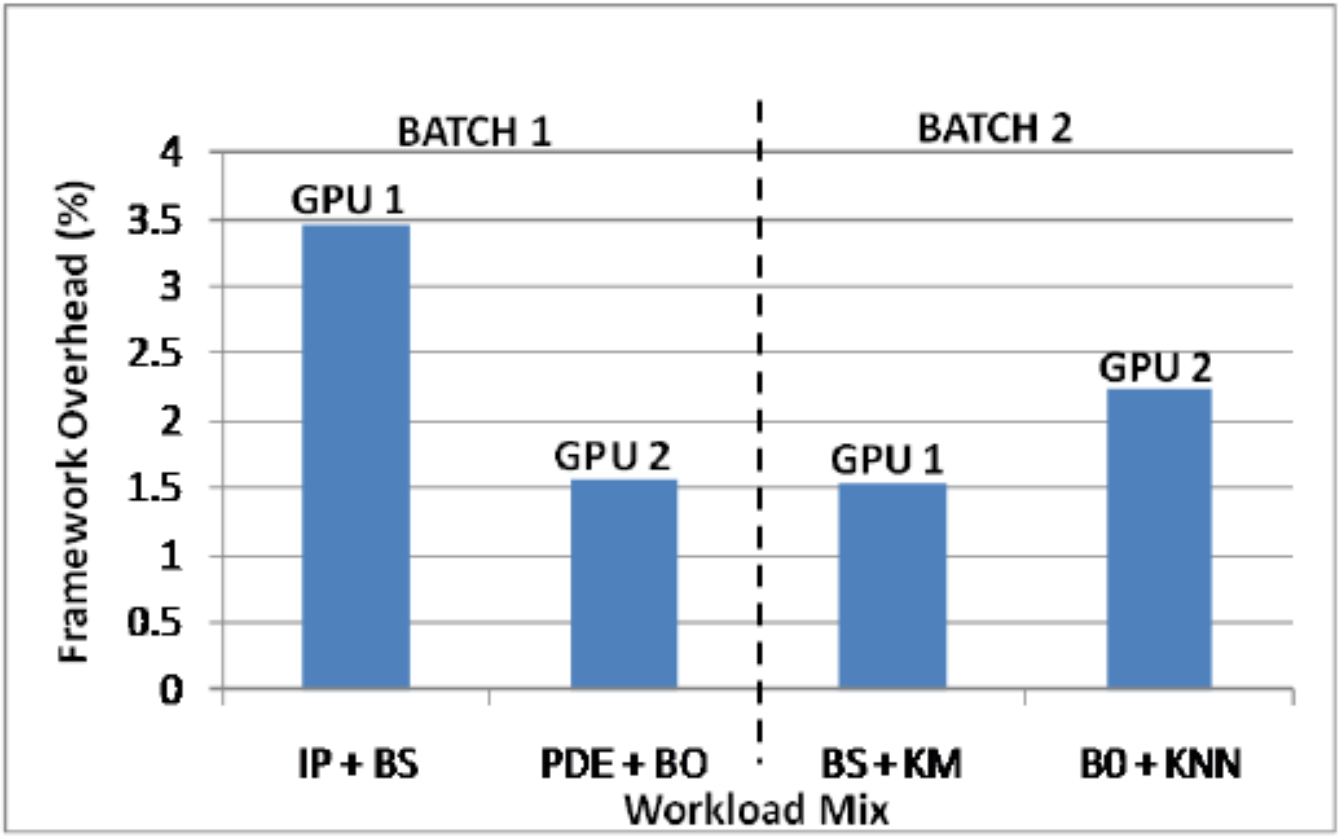
Framework Overhead Analysis

- Framework overhead when no consolidation



Framework Overhead Analysis

- Framework Overhead when scheduling than the available GPUs



Outline

- Background
- Approach
- Software Design
- Policies and Algorithm
- Evaluation
- Conclusion

Conclusion

- Present framework to enables applications to share GPUs
  - Extend gVirtuS for enabling consolidation
- Evaluate framework
  - Guarantee performance over sequential execution
  - Improve throughput using consolidation algorithm when contention occur

Comments

- No evaluation in cloud environment