Matching with Non-Uniformity: A Key to Efficiency on Multicore and GPU

Xipeng Shen

Eddy Z Zhang, Yunlian Jiang, Ziyu Guo, Kai Tian

Computer Science Department

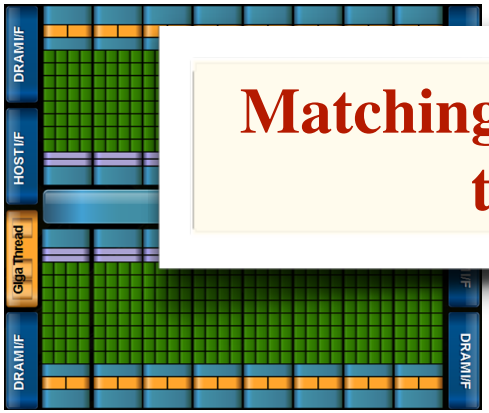
The College of William and Mary

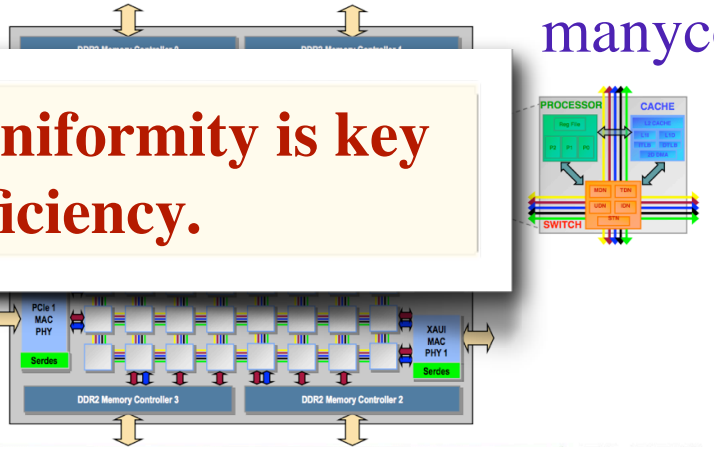
Appearance of Non-Uniformity

non-uniform execution model

non-uniform communication

GPU



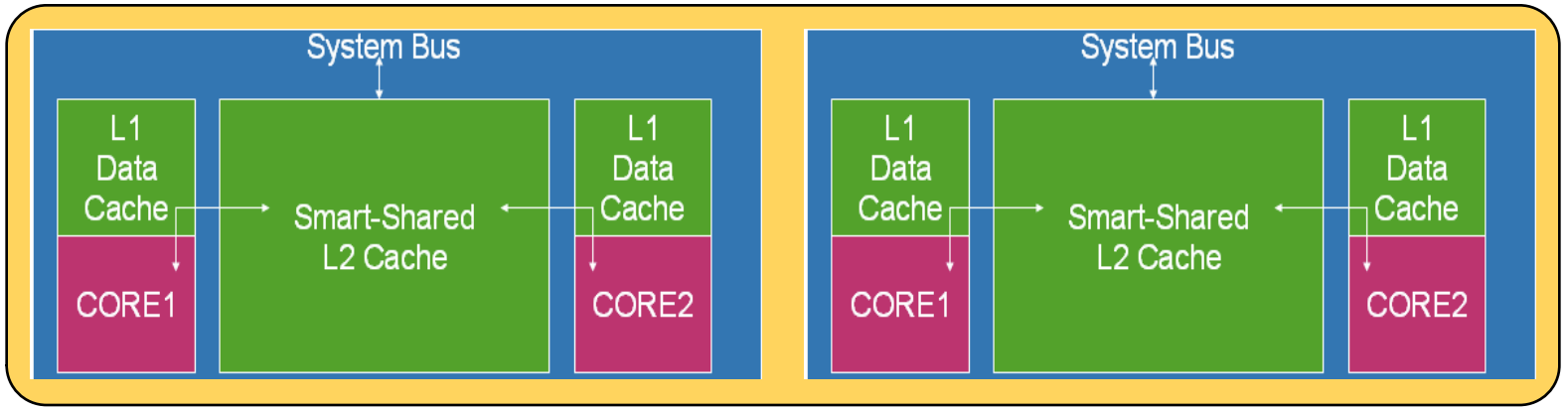


manycore

Matching with the non-uniformity is key to computing efficiency.

non-uniform resource sharing

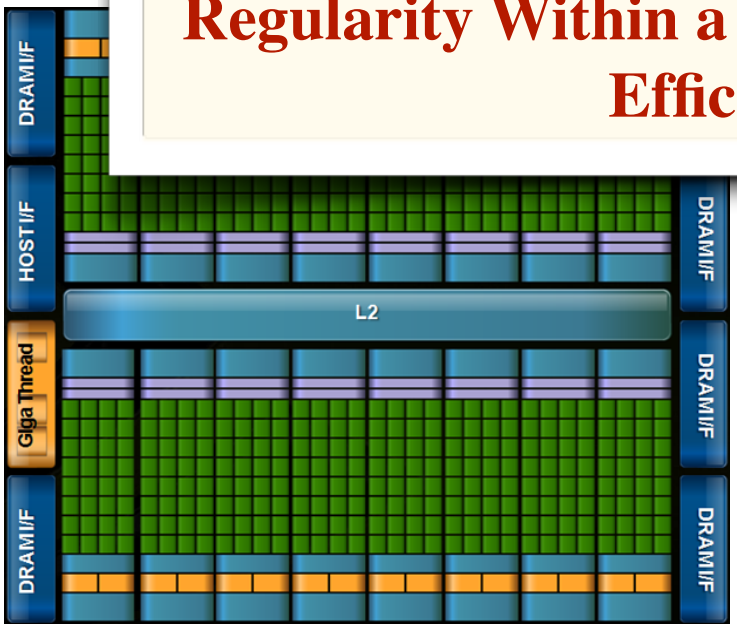
multicore

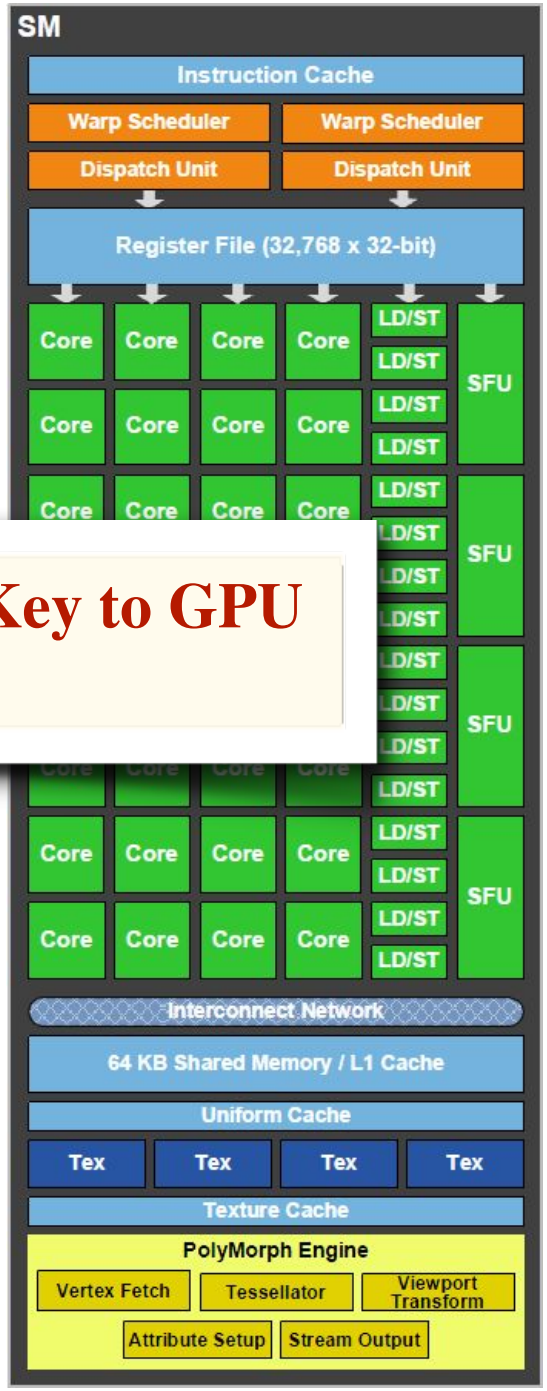


GPU

a SIMD group (warp)

Regularity Within a Warp is Key to GPU Efficiency

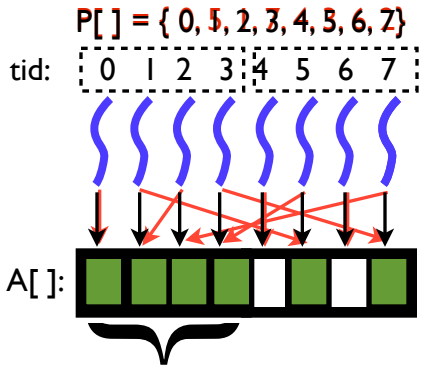




Dynamic Irregularities

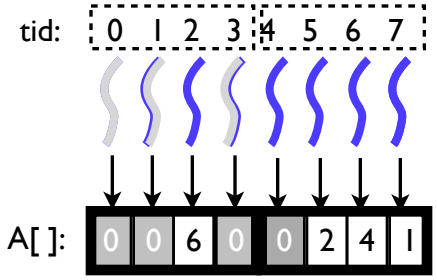
memory

... = A[P[tid]];



a mem seg.

control flow (thread divergence)



if (A[tid]) {...}

for (i=0;i<A[tid]; i++) {...}

Degrade throughput by up to (warp size - 1) times.

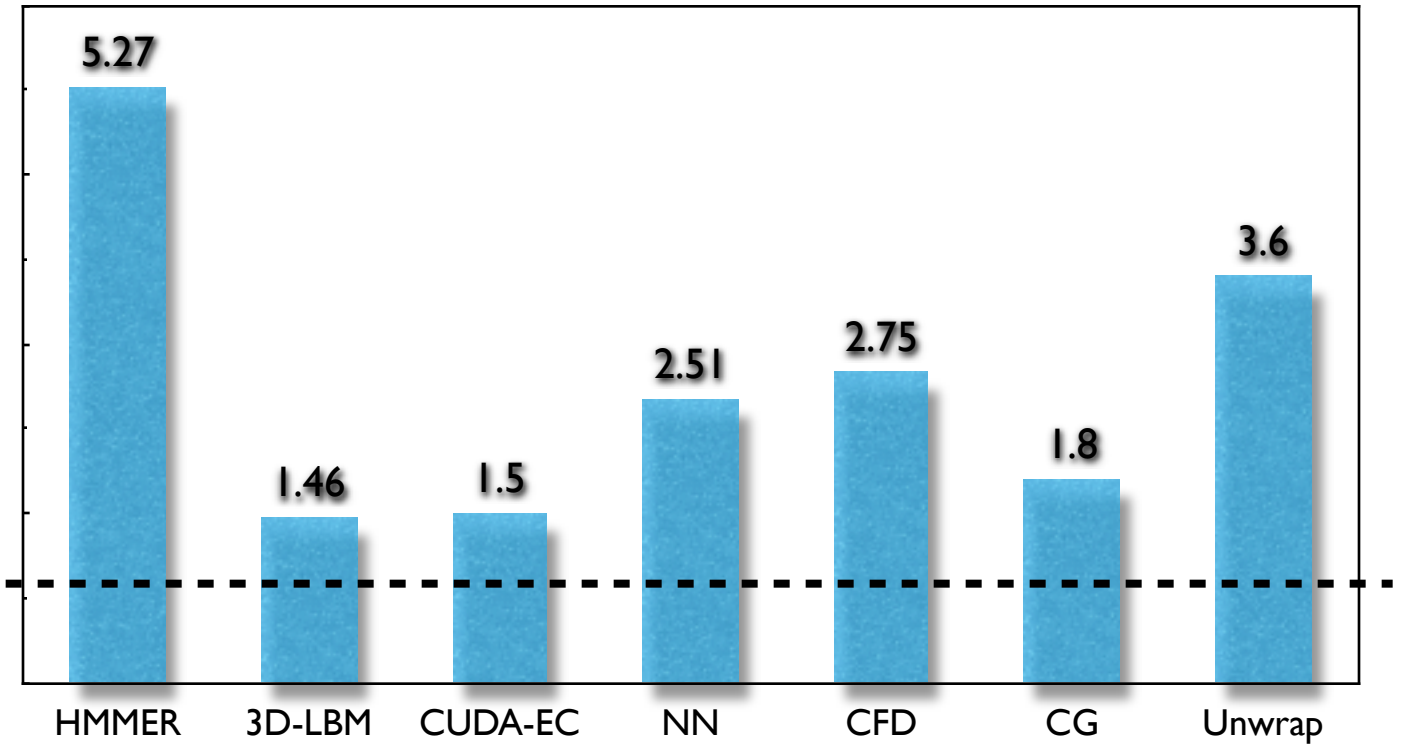
(warp size = 32 in modern GPUs)

Performance Impact

- Applications: dynamic programming, fluid simulation, image reconstruction, data mining, ...

Host: Xeon 5540.

Device: Tesla 1060.



Previous Work on GPU

- Mostly on Static Memory Irregularities
  - [Baskaran+, ICS'08], [Lee+, PPoPP'09],[Yang+, PLDI'10], etc.
- Dynamic Irregularity
  - Remain unknown until runtime
  - Mostly through hardware extensions.
    - [Meng+, ISCA'10], [Tarjan+, SC'09], [Fung+, MICRO'07], etc.

Open Questions for Dyn. Irreg. Removal

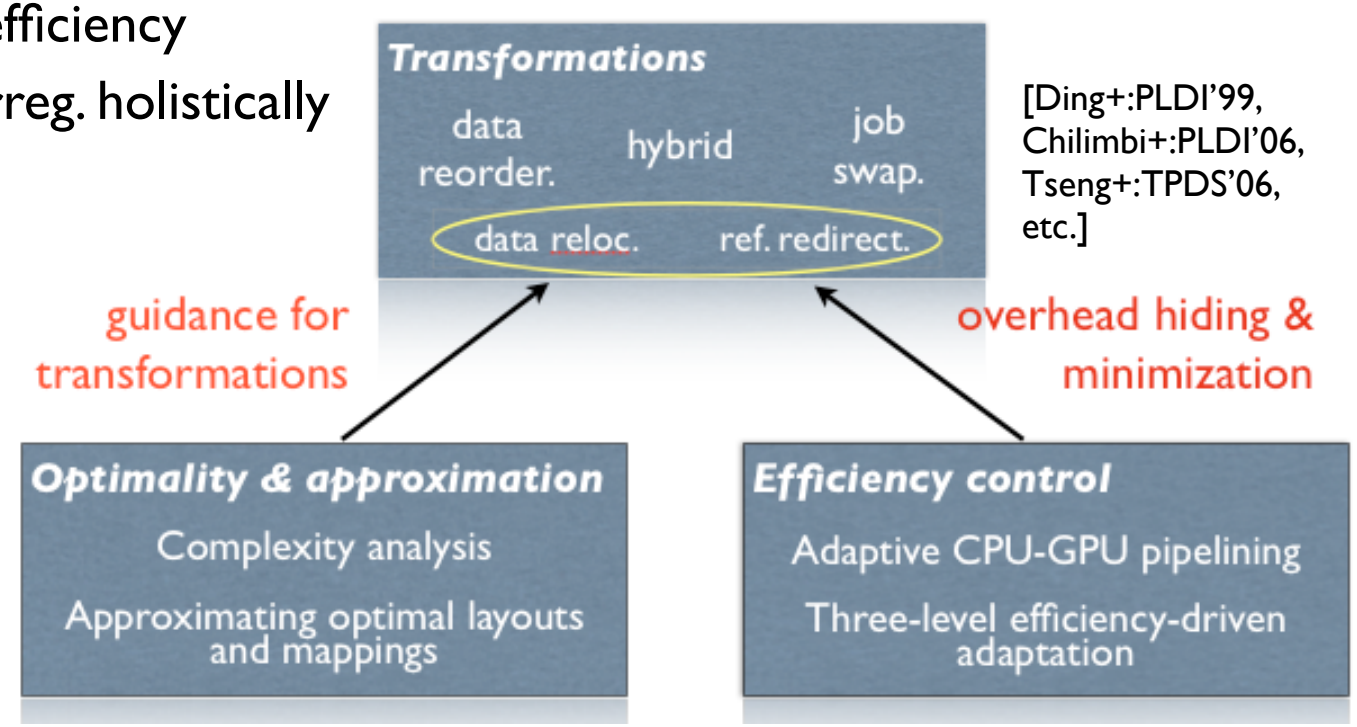
- Is it possible w/o hardware extentions?
- More fundamentally
  - Relations among data, threads, & irregularities?
  - What layout or thread-data mappings minimize the irreg.? How to find them? Complexity?
  - How to resolve conflicts among irregularities? Dependences?

Streamlining On the Fly

[Zhang+:ASPLOS'11]

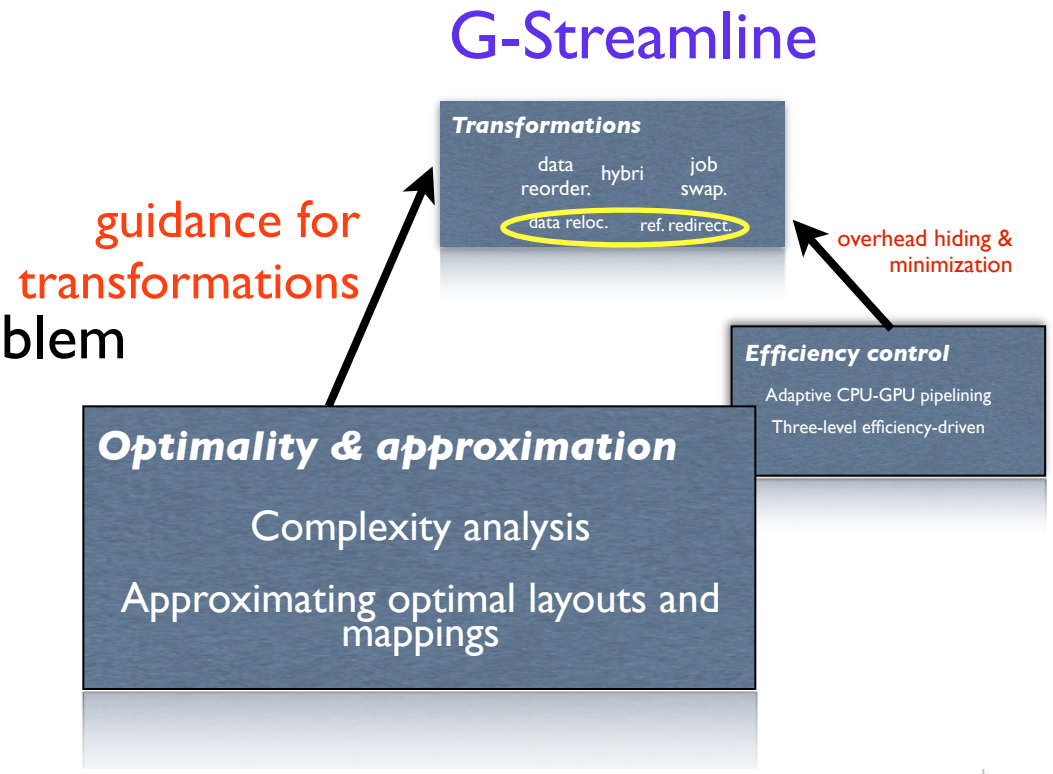
- No profiling or hw ext.
- Transparent removal on the fly
- Jeopardize no basic efficiency
- Treat both types of irreg. holistically

G-Streamline



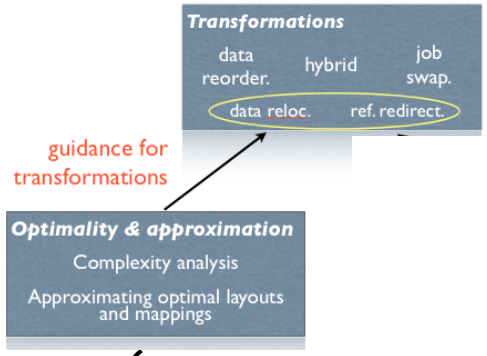
[Ding+:PLDI'99, Chilimbi+:PLDI'06, Tseng+:TPDS'06, etc.]

- NP-Complete
  - Layout: 3D matching
  - Mapping: Partition Problem
- Approx.
  - Duplication
  - Clustered sorting



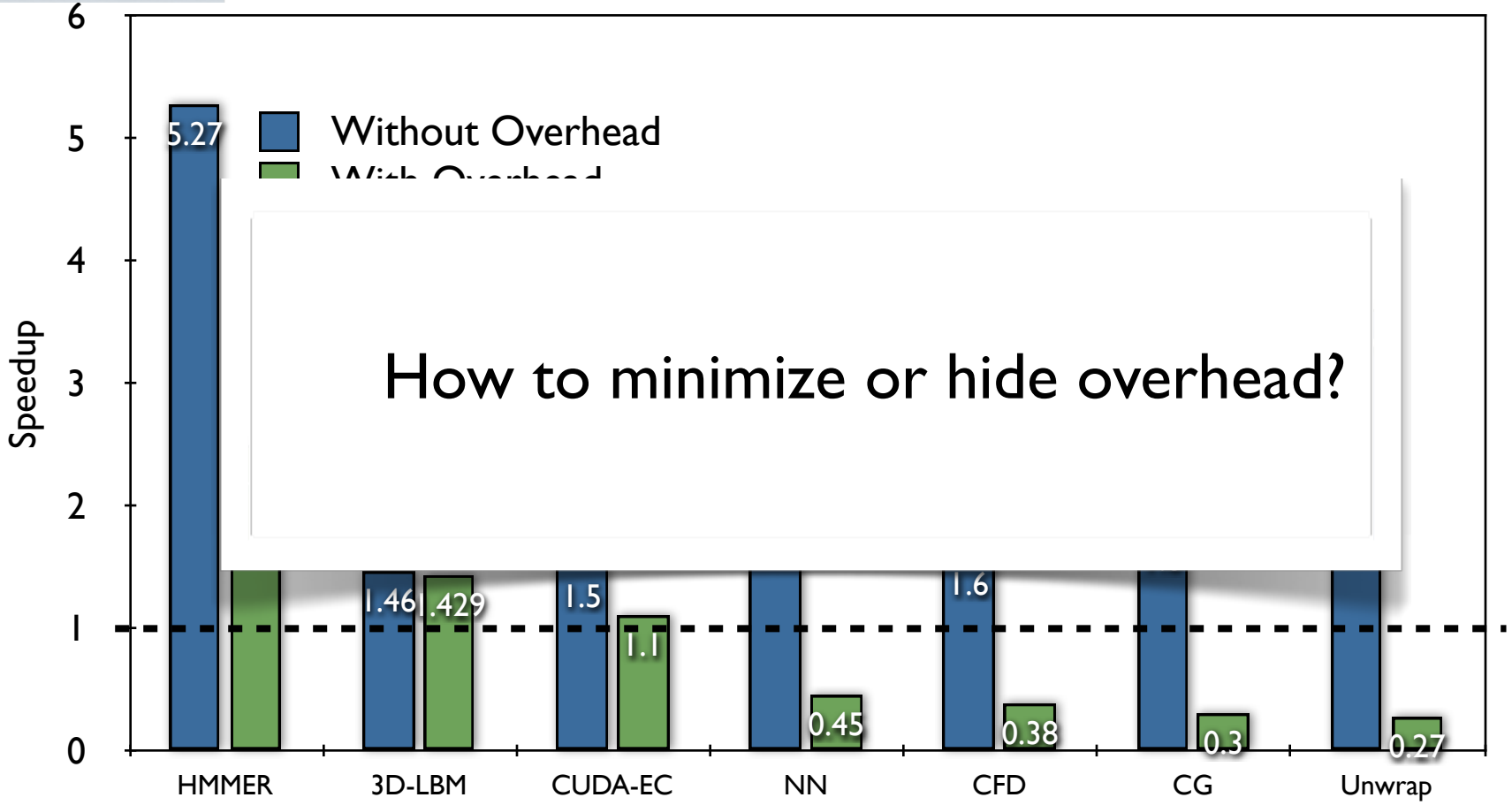
How to determine
optimal layouts / thread-data mapping?

See Zhang+:ASPLOS'11 for details.



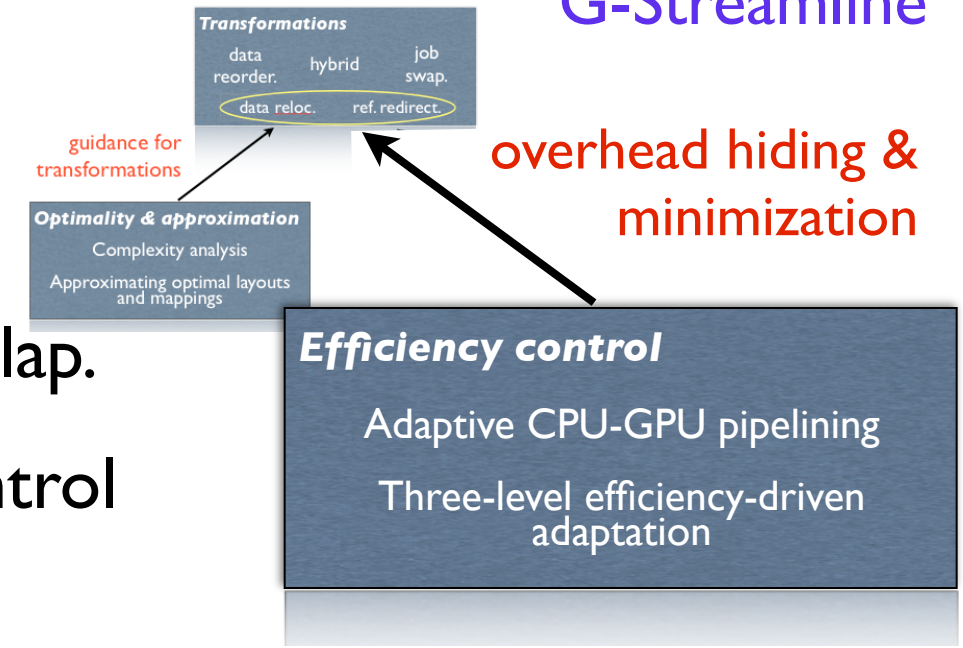
After Transformation

- *Benchmark Suites: Rodinia, Tesla Bio, and etc.*
- *Host: Xeon 5540. Device: Tesla 1060*



How to minimize or hide overhead?

- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control

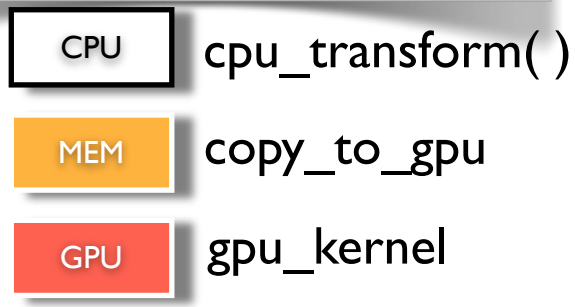


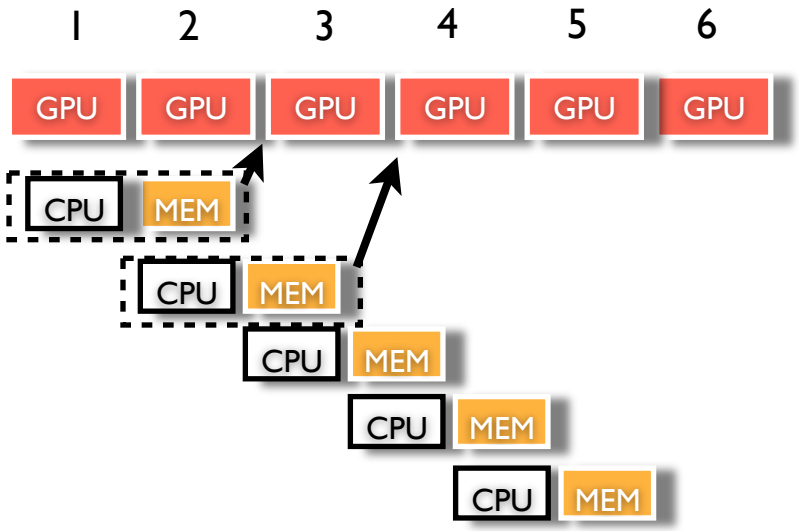
- ❖ Transparent, on-the-fly
- ❖ Adaptive to pattern changes
- ❖ No perf. degradation
- ❖ Resilient to dependence
- ❖ Automatically balance benefits and overhead

CPU-GPU Pipelining

- Utilize Idle CPU Time
  - ❖ Transform on CPU while computing on GPU
  - ❖ Automatic shutdown when necessary

```
for i=1:n
    async_transform (i+2);
    async_copy (i+2);
    gpu_kernel(i);
end
```





Kernel Splitting

```
gpuKernel_org<<<...>>>(pData,...);
```

split

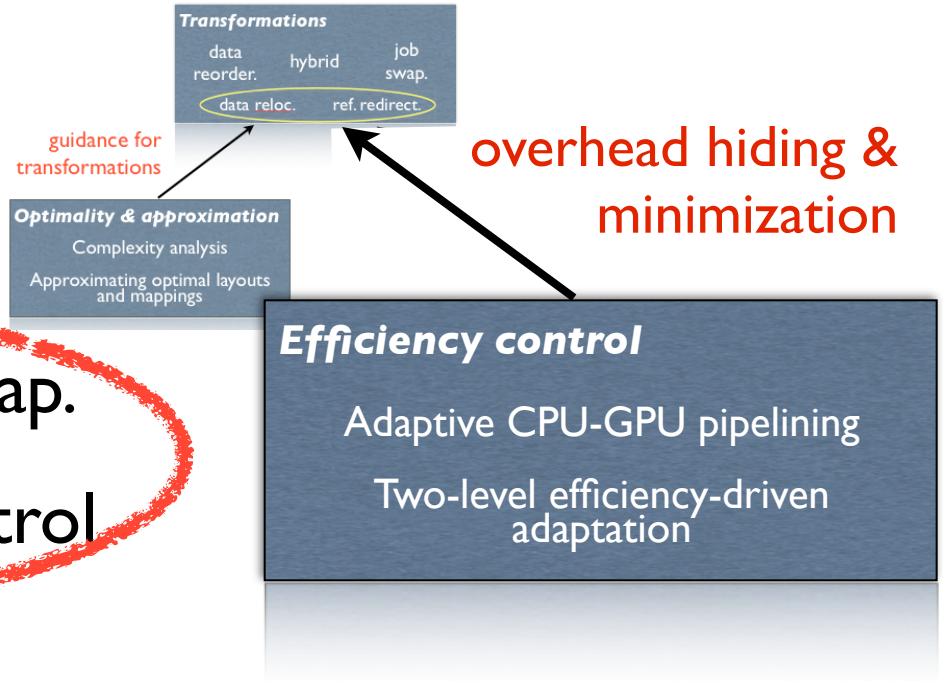
pipeline

```
gpuKernel_org_sub<<<...>>>(pData,0, (1-r)*len, ...);

gpuKernel_opt_sub<<<...>>>(pData,(1-r)*len+1, len, ...);
```

- Also useful for loops with no dependences
  - Enable partial transformation

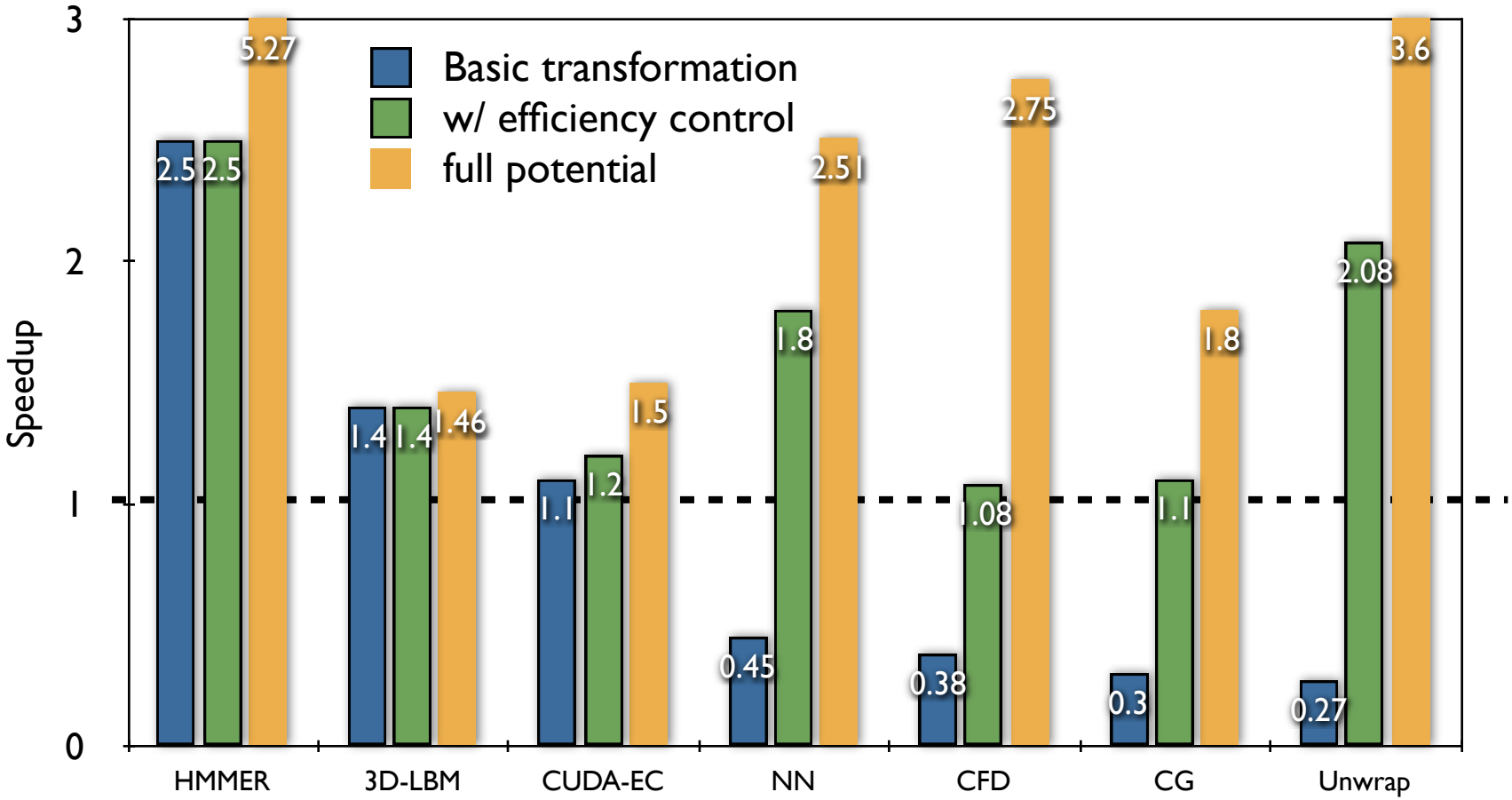
- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control



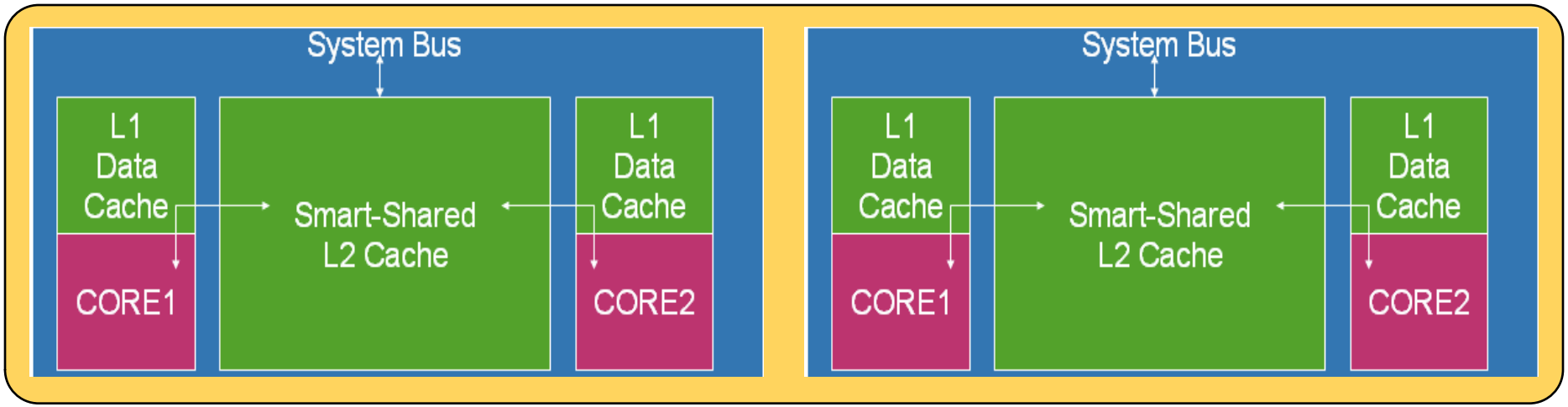
- ❖ Transparent, on-the-fly
- ❖ Adaptive to pattern changes
- ❖ No perf. degradation
- ❖ Resilient to dependence
- ❖ Automatically balance benefits and overhead

See Zhang+:ASPLOS'11 for details.

Final Speedup



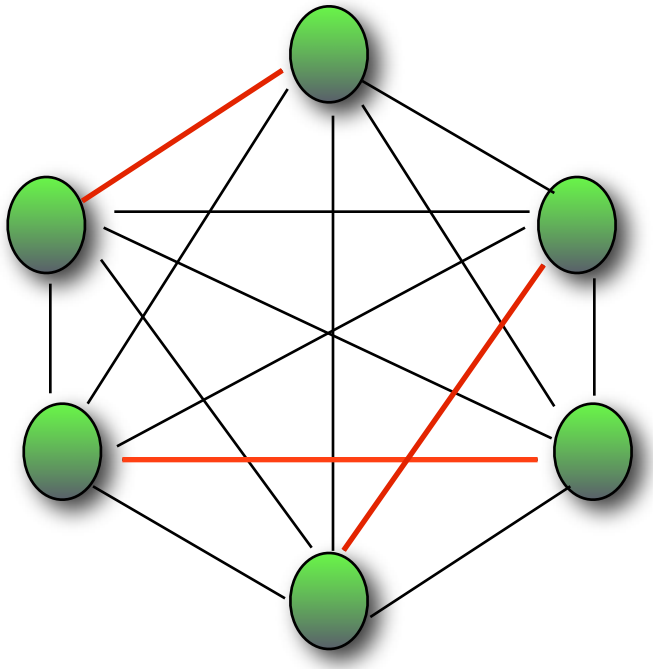
Non-Uniform Cache Sharing on Multicore

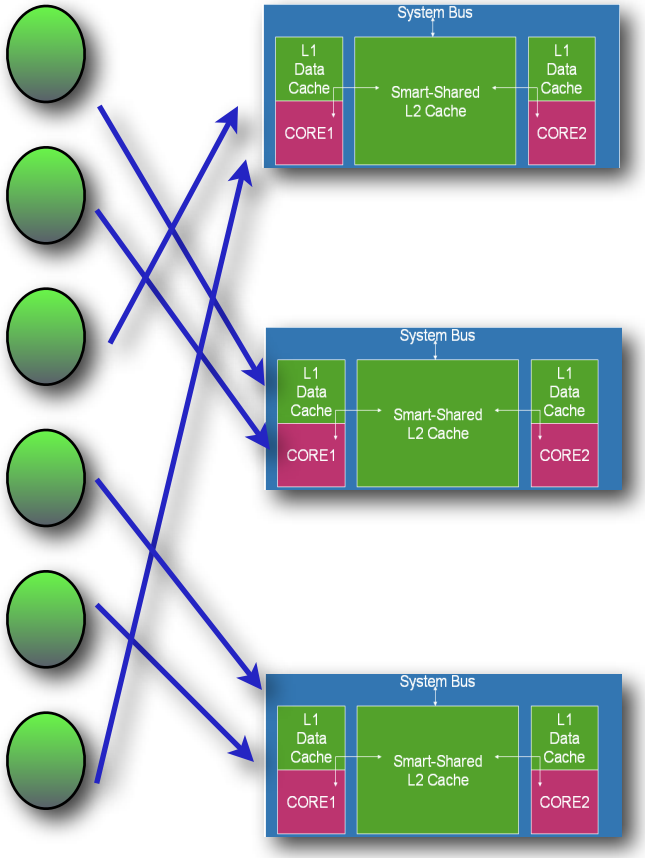


- Cache sharing is a double-edged sword
  - Reduces communication latency
  - But causes conflicts & contention

Optimal Job Co-Scheduling

[Jiang+: PACT'08, Jiang+:TPDS'11]



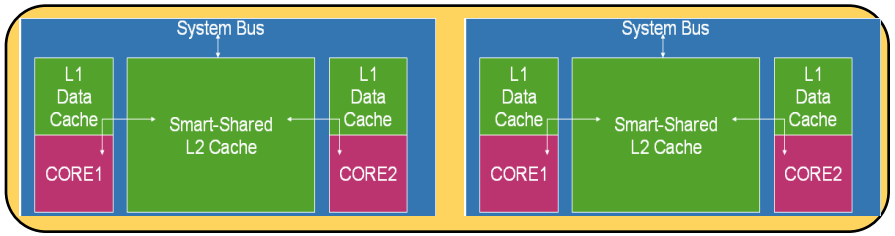


- Minimum-weight perfect matching problem & a $O(N^4)$ solution.
- NP-completeness for K-core (K>2) & some approximation algorithms.

On Threads of a Multithreading Application

[Zhang+:PPoPP'10]

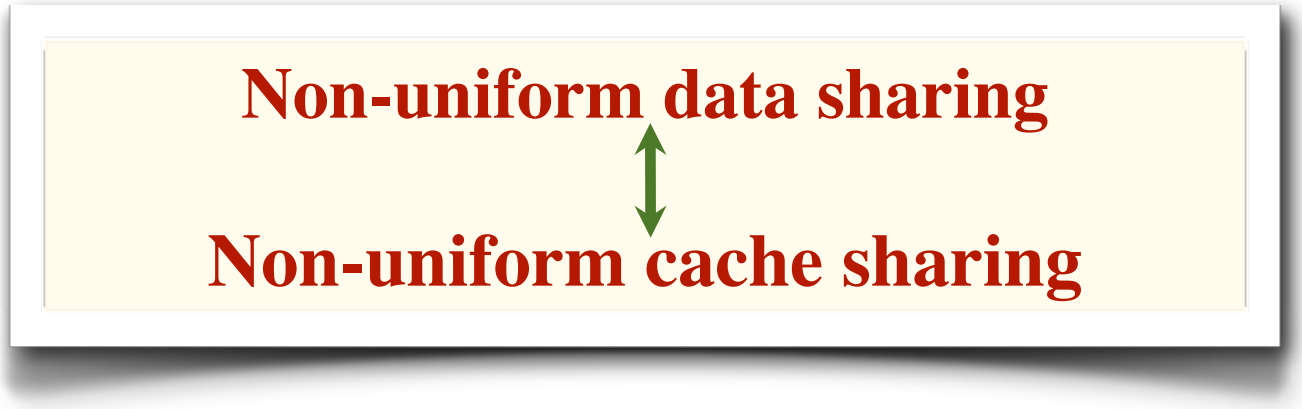
- Observation
  - Insignificant effects from shared cache (PARSEC)
- Reasons
  - Three mismatches with non-uniform shared cache
    - Limited data sharing among threads
    - Large working set size
    - Uniform inter-thread relations



Cache-Sharing-Aware Transformations

[Zhang+:PPoPP'10]

- Increase data sharing among siblings
- Decrease data sharing otherwise

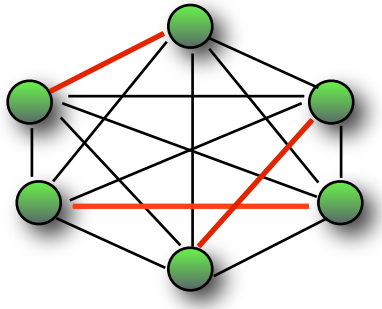


- >50% cache miss reduction.
- 5-33% speedup.

Summary

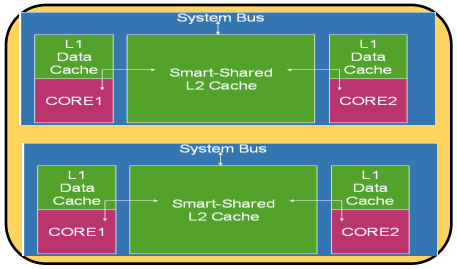
Matching with non-uniformity is the key.

Job co-sch.



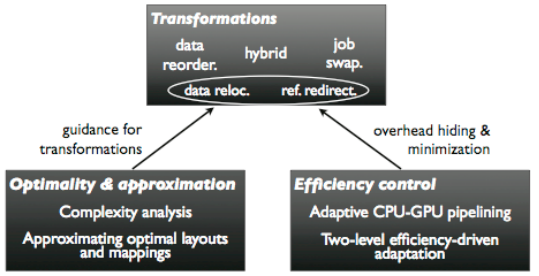
Exploiting job non-uniformity

Cache-sharing-aware transformation



Creating thread non-uniformity

G-Streamline



Eliminating warp non-uniformity

Shared cache on multicore

Execution model on GPU

Thanks!