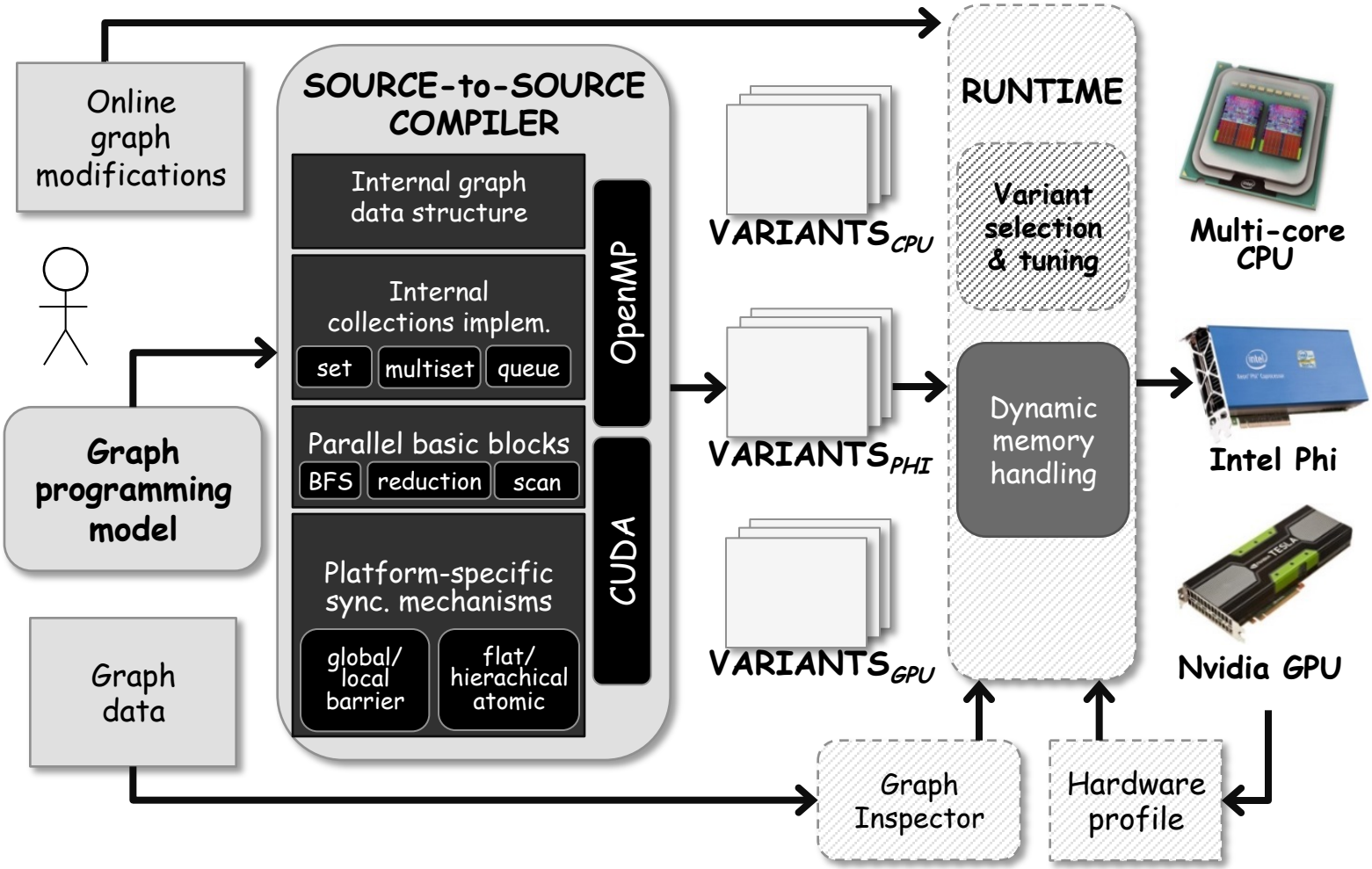
Facilitating Irregular Applications on Many-core Processors

Da Li

Advisor: Dr.Michela Becchi
Electrical and Computer Engineering
University of Missouri

http://nps.missouri.edu

Unifying Programming Interface



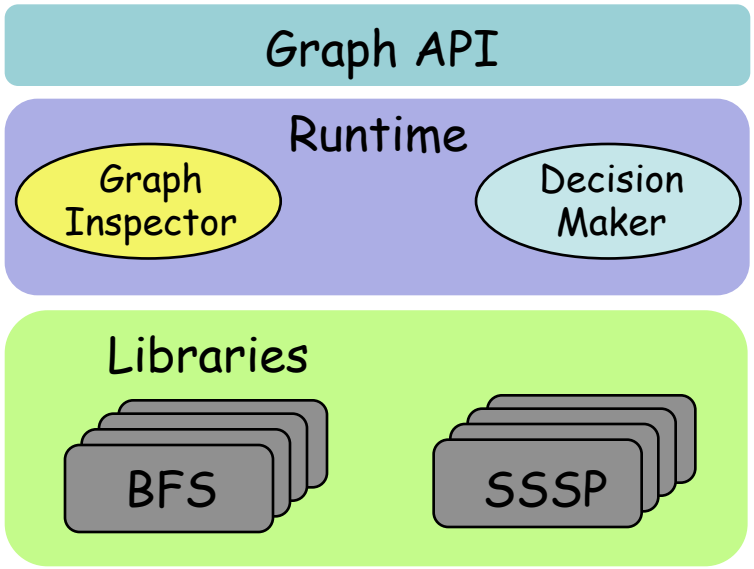
Results

Application	Application & Graph Type	Attributes	Comp. pattern & arith. intensity	Best Speedup		
				m-CPU	GPU	Phi
BFS	read-only static	Level (int)	Set level (simple& low)	2.5x	50x	3.5x
PageRank	read-only static/dynamic	Rank (double)	Calculate Rank (intermediate & intermediate)	6.3x	6.5x	9x
A-DFA	read-only static	Default Trans (int)	Compare trans. (complex, low)	8x	6x	45x
DFA construction	read-write dynamic	Trans Table (int)	Compare Subset (complex, low)	6.5x	5.5x	4.3x

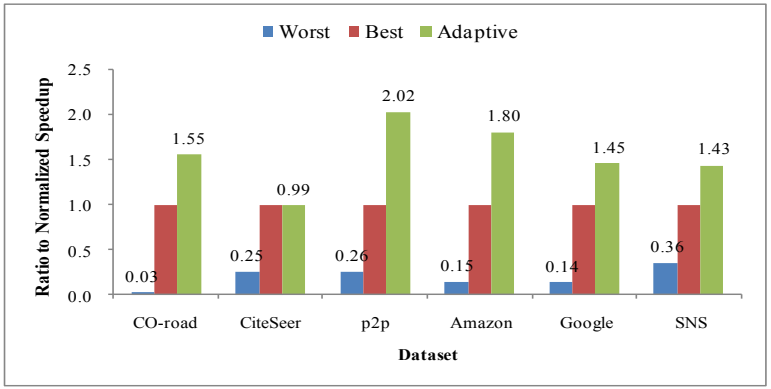
- Platform selection depends on application and graph type
- Unified programming interface facilitates fast-prototyping

Adaptive Computing

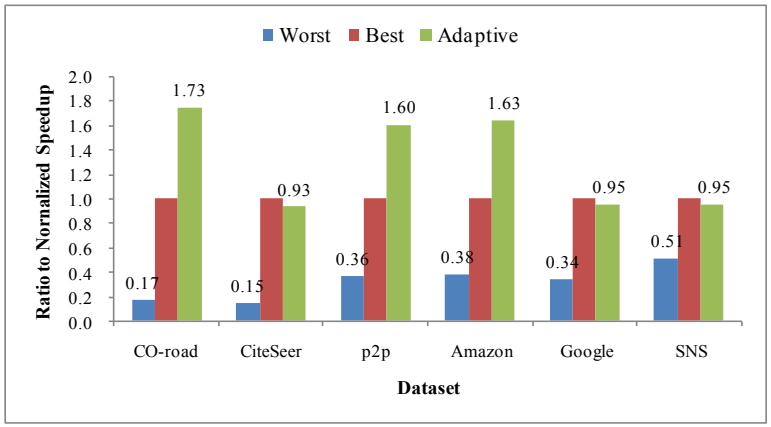
System overview



BFS

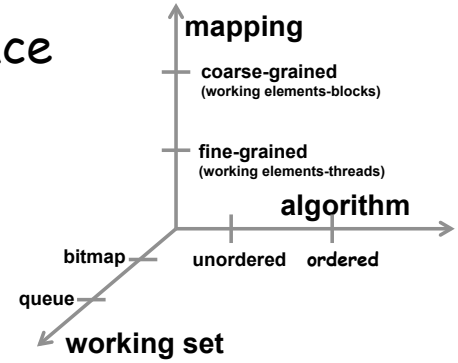


SSSP



3-d exploration space

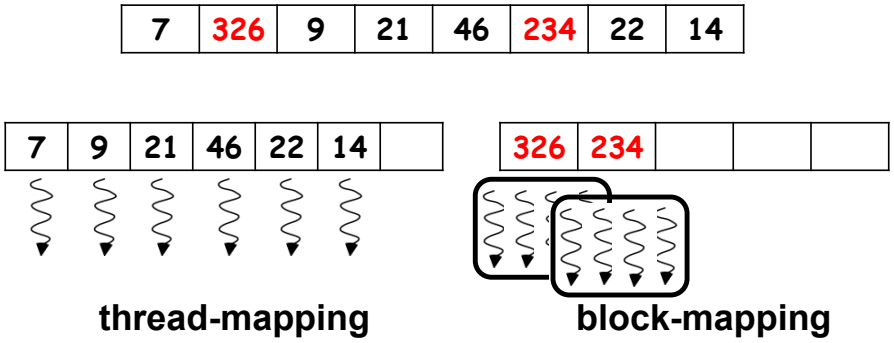
- working set representation
- algorithm
- mapping method



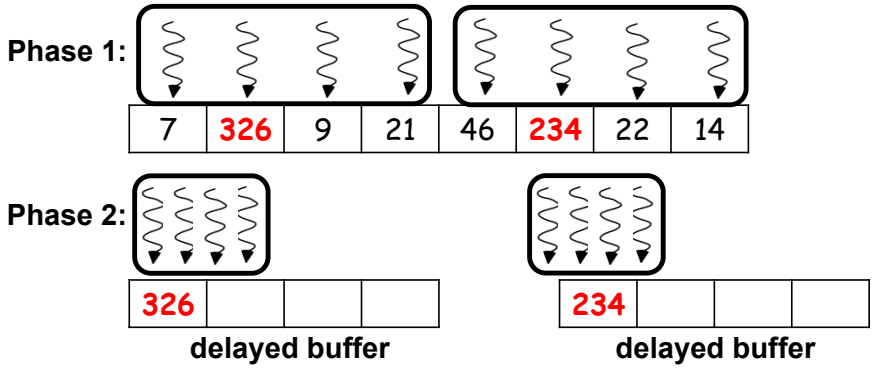
- Performance of static solution varies from dataset to dataset
- Adaptive solution outperforms static ones

Parallelization Templates

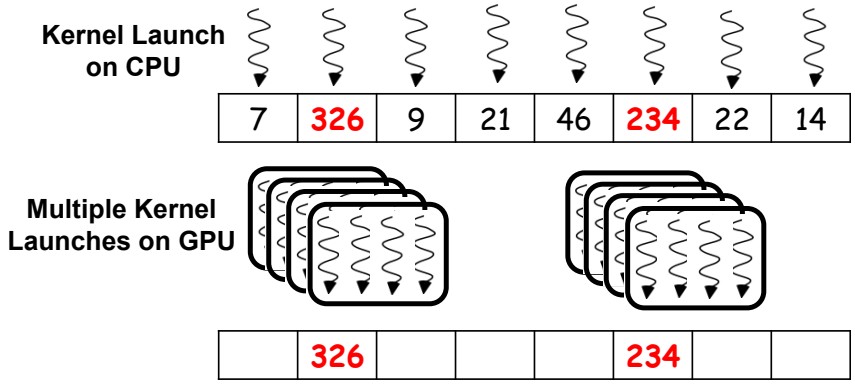
dual-queue



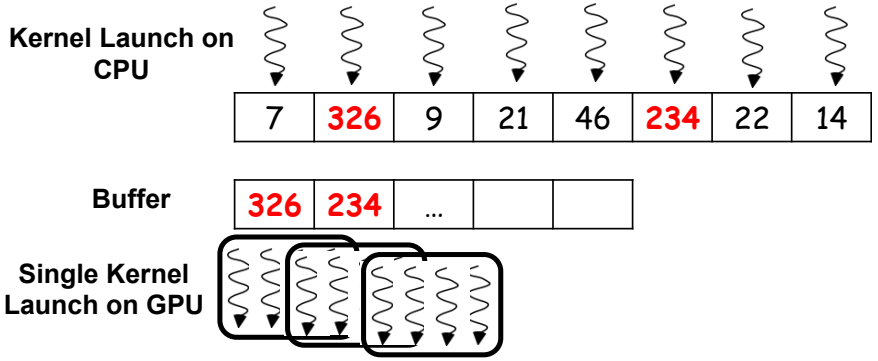
delayed-buffer(global/shared)



naïve-dynamic parallelism



optimized dynamic parallelism



Results

SSSP

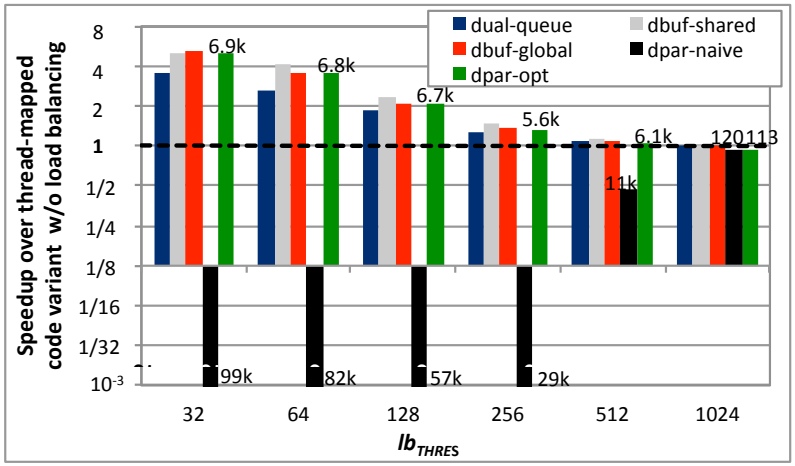


TABLE I. Profiling data collected on SSSP (lb_{THRES}=32)

Templates	Metrics		
	warp efficiency	***gld efficiency***	***gst efficiency***
baseline	35.6%	15.8%	3.2%
dual-queue	74.9%	79.1%	4.8%
dbuf-shared	75.7%	94.3%	50.4%
dbuf-global	72.3%	89.1%	8.5%
dpar-naïve	25.3%	45.5%	16.3%
dpar-opt	70.2%	63.2%	10.9%

- lb_{Thres} is the major factor affecting performance
- dpar-naïve degrades performance because of too many small kernel launches
- dual-queue, dbuf-shared, dbuf-global, dpar-opt lead to a better utilization of GPU cores and memory bandwidth
- dbuf-shared outperforms baseline and other code templates

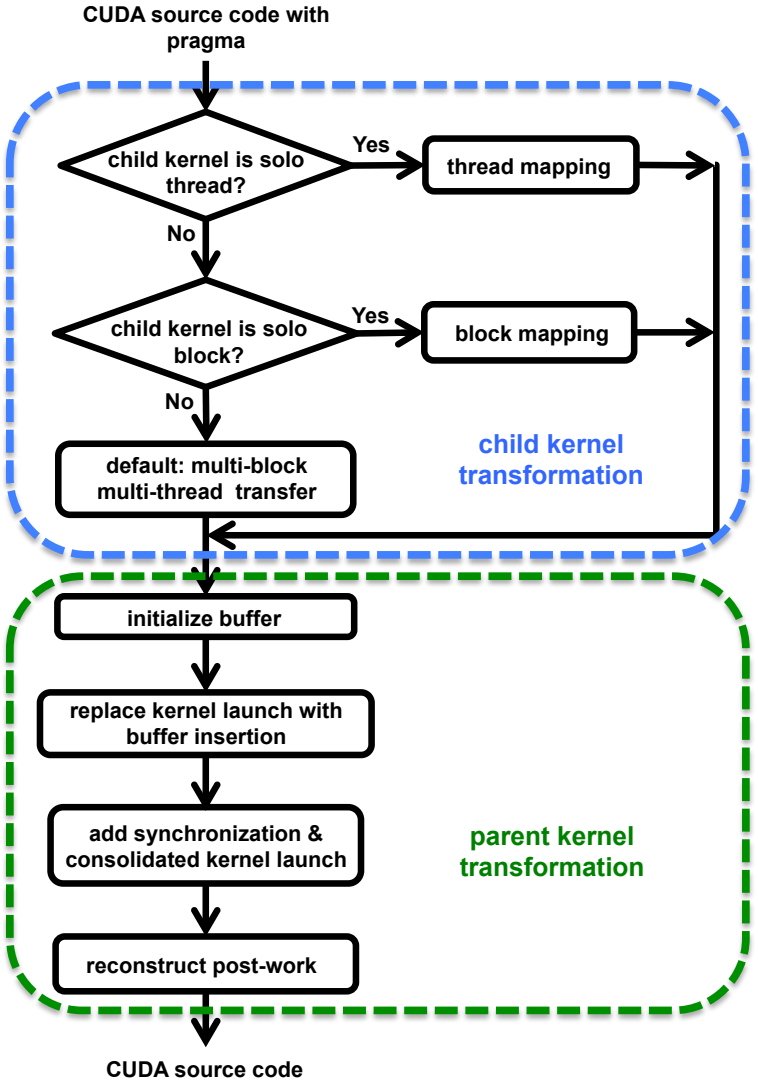
Workload Consolidation

Annotated CUDA source code

```
__global__ void parent_kernel() {
   work_item = get_work_item(…)
   prework(work_item)
   if (condition) {
  #pragma dp consldt(block) buffer(default, 256) work(work_item)
      child_kernel<<<block_dim, thread_dim>>>(…, work_item, …)
   } else  work(work_item)
   postwork(work_item)
}
```

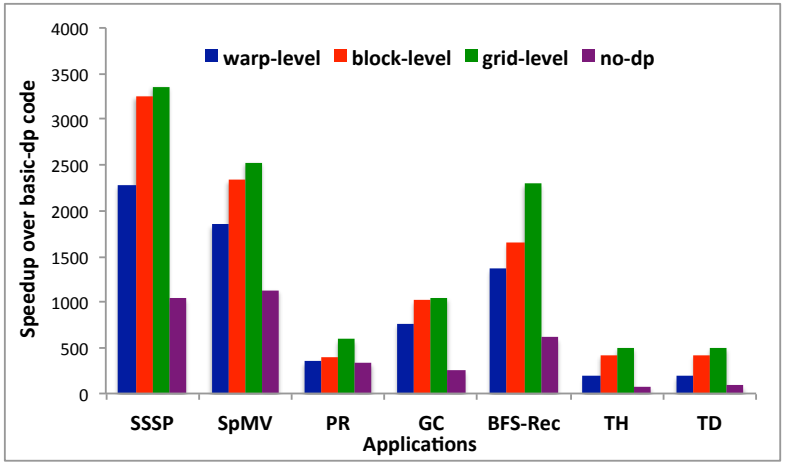
Generated CUDA kernel

```
__global__ void parent_kernel() {
   work_item = get_work_item(…);
   prework(work_item)
   if (condition)  insert_buffer(curr)
   else  work(work_item)
   synchronize
   if (thread_id==selected)
      child_kernel_consolidate<<<b_dim_con, t_dim_con>>>()
   synchronize
   postwork(work_item)
}
```

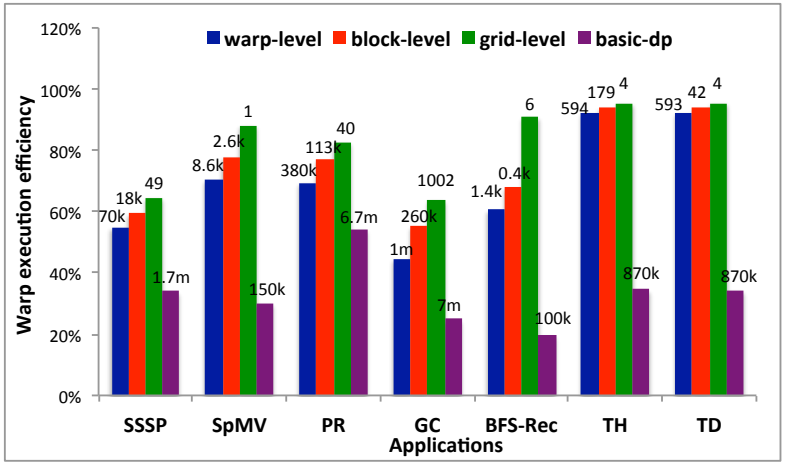


Results

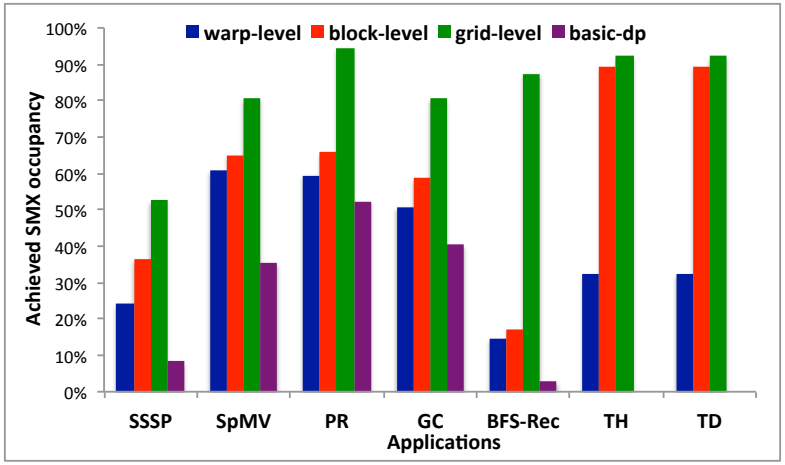
Overall Speedup



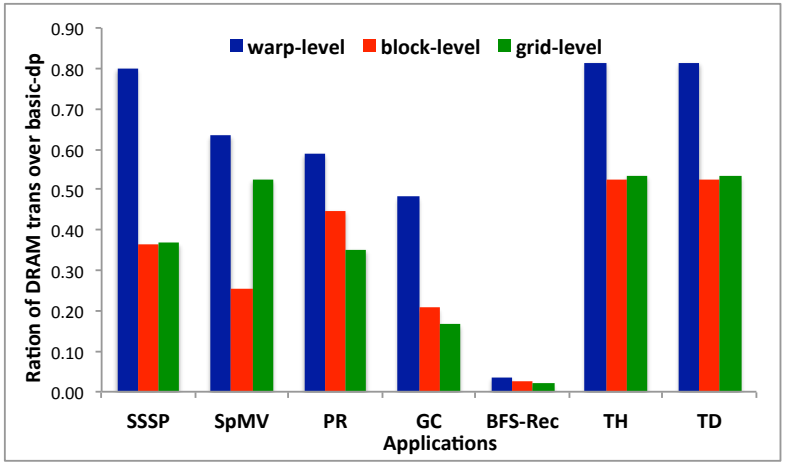
Warp Efficiency



SMX Occupancy



DRAM Transactions



Compiler-assisted workload consolidation can improve the performance