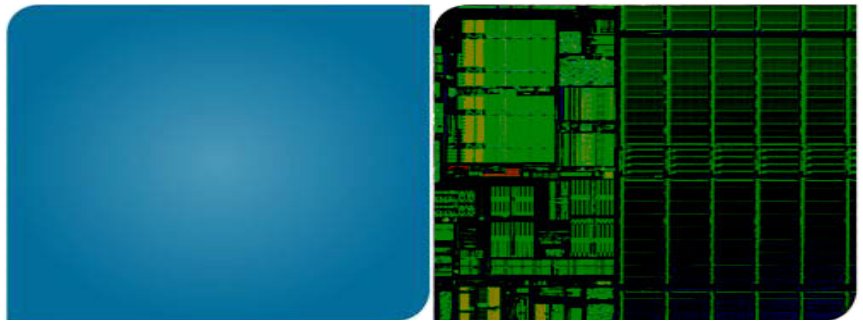
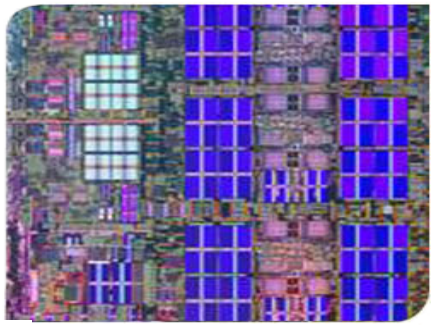
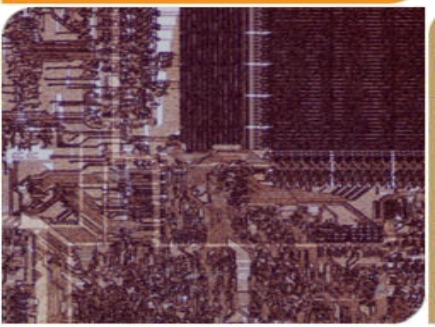




Modeling of Heterogeneous Computing Systems

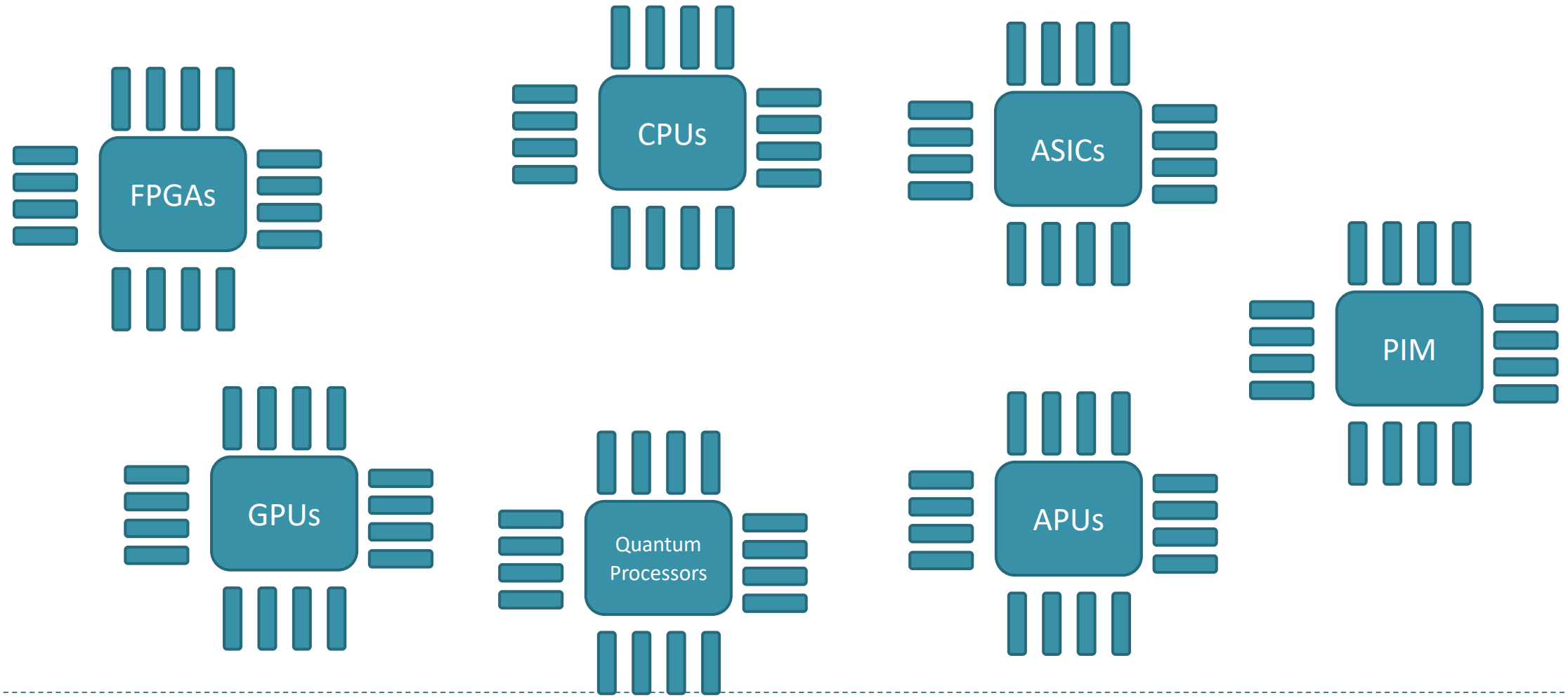



Hyesoon Kim
hyesoon@cc.gatech.edu

Georgia Tech

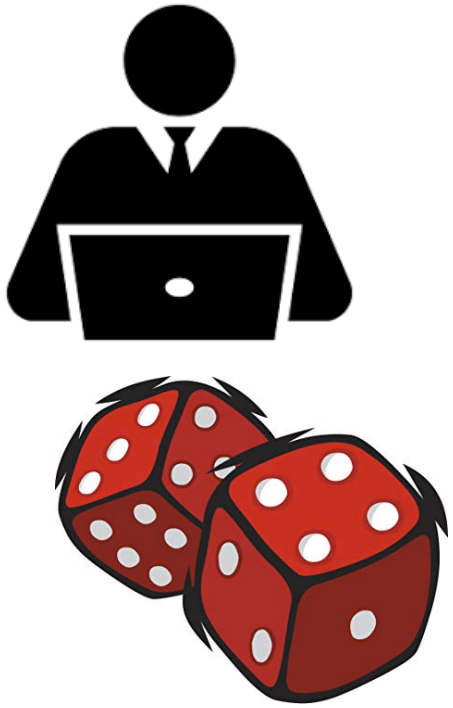
comparch

Special Hardware



Where to execute? And When to execute?

Software



FPGAs

CPUs

ASICs

GPUs

APUs

Quantum Processors

PIM

Georgia Tech

comparch



Case study 1: CPU vs. GPU

Georgia Tech comparch

Motivating problem

- We have a CPU code.
  - Should we convert the code for GPUs or not?
  - Can we know in advance before converting the code for GPUs?

Round 1: CPU vs. GPU compute power

Image source: street fighter

What if I have

- GPU codes

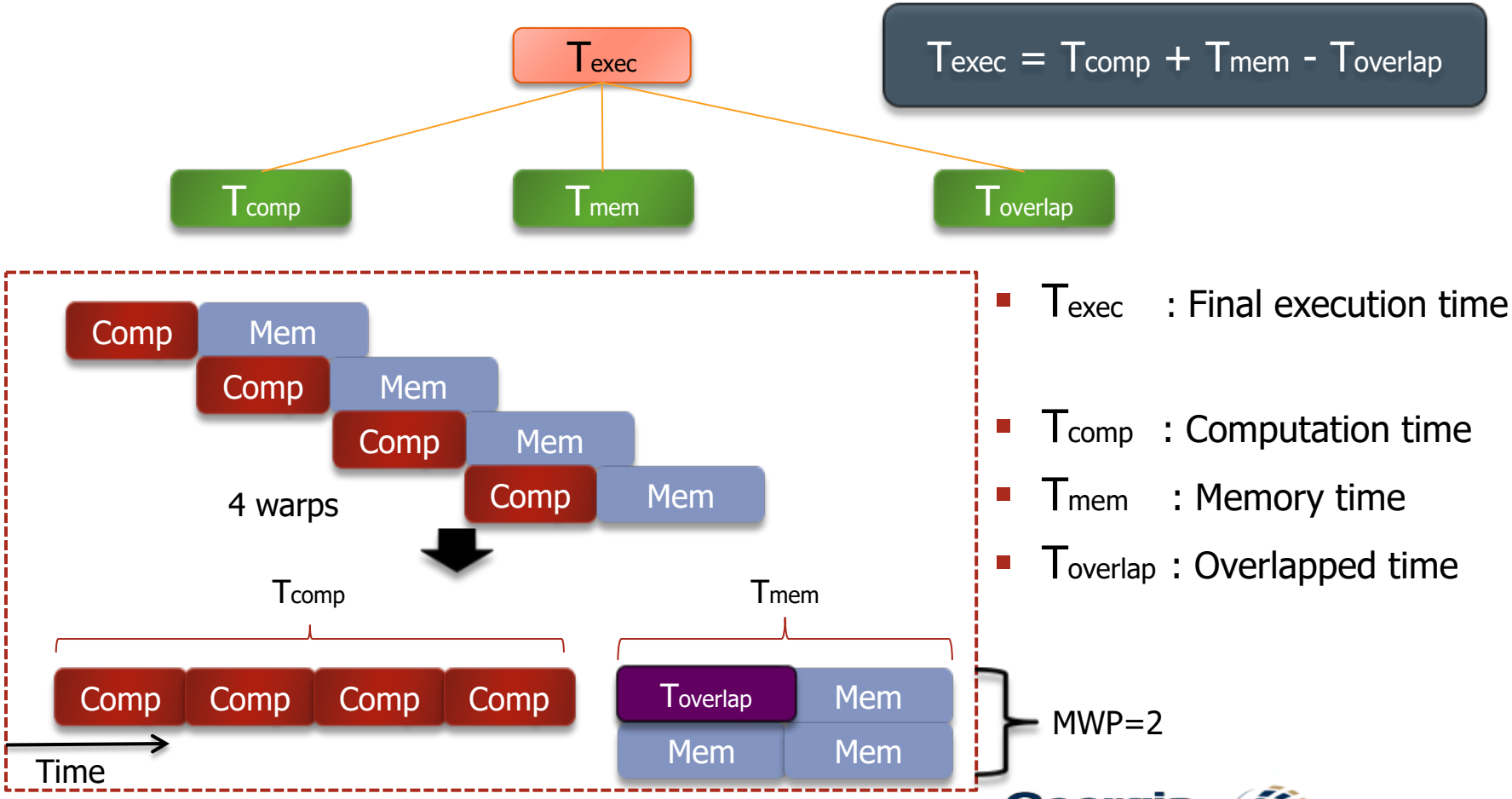
```
__global__
void kernel(uint32_t * vplist, cudaGraph graph, unsigned curr, bool *changed) {
        uint64_t tid = blockIdx.x * blockDim.x + threadIdx.x;
    uint64_t lane_id = tid % WARP_SZ;
    uint64_t warp_id = tid / WARP_SZ;
    uint64_t v1= warp_id * CHUNK_SZ;
    uint64_t chk_sz=CHUNK_SZ;

    if((v1+CHUNK_SZ)>graph.vertex_cnt)
    {
        if ( graph.vertex_cnt>v1 )
            chk_sz =  graph.vertex_cnt-v1;
        else
            return;
    }

    for(int v=v1; v< chk_sz+v1; v++)
    {
        if(vplist[v] == curr)
        {
            uint64_t nbr_off = graph.get_firstedge_index(v);
            uint64_t num_nbr = graph.get_edge_index_end(v) - nbr_off;
            for(uint64_t i=lane_id; i<num_nbr; i+=WARP_SZ)
            {
               uint64_t vid = graph.get_edge_dest(i + nbr_off);
               if(vplist[vid]==MY_INFINITY)
               {
                    vplist[vid] = curr + 1;
                    *changed = true;
               }
            }
        }
    }
}
```

Execution time is...

- Use an analytical model to follow a top-down approach

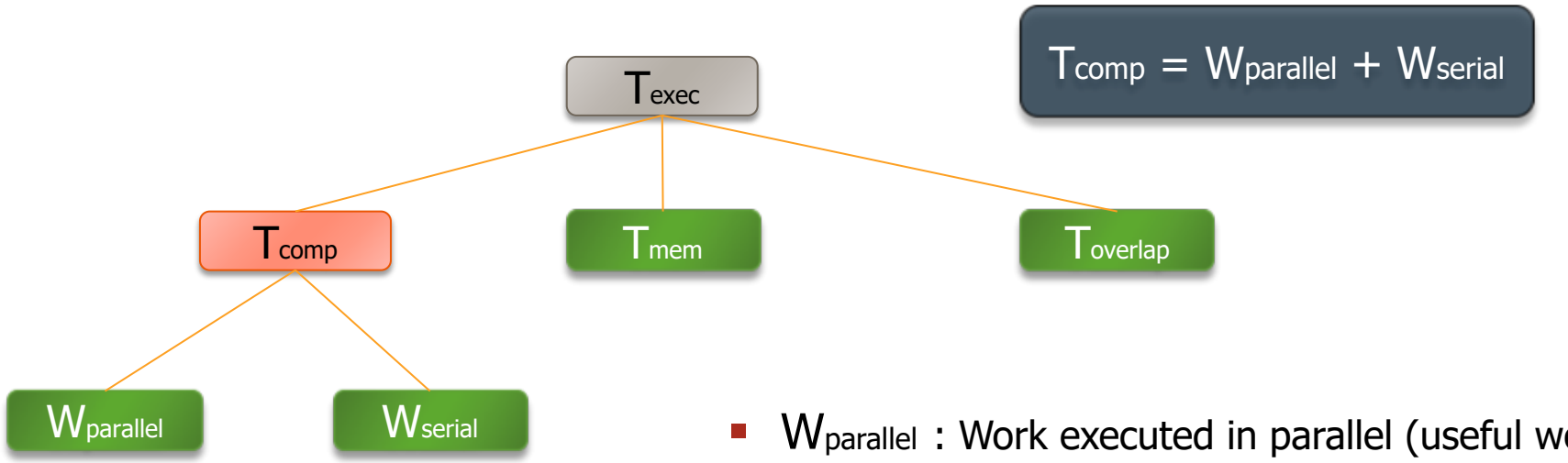


- T_{exec} : Final execution time
- T_{comp} : Computation time
- T_{mem} : Memory time
- $T_{overlap}$: Overlapped time

Sim et al. (PPoP'12)

Analytical Model

- T_{comp} is the amount of time to execute compute instructions



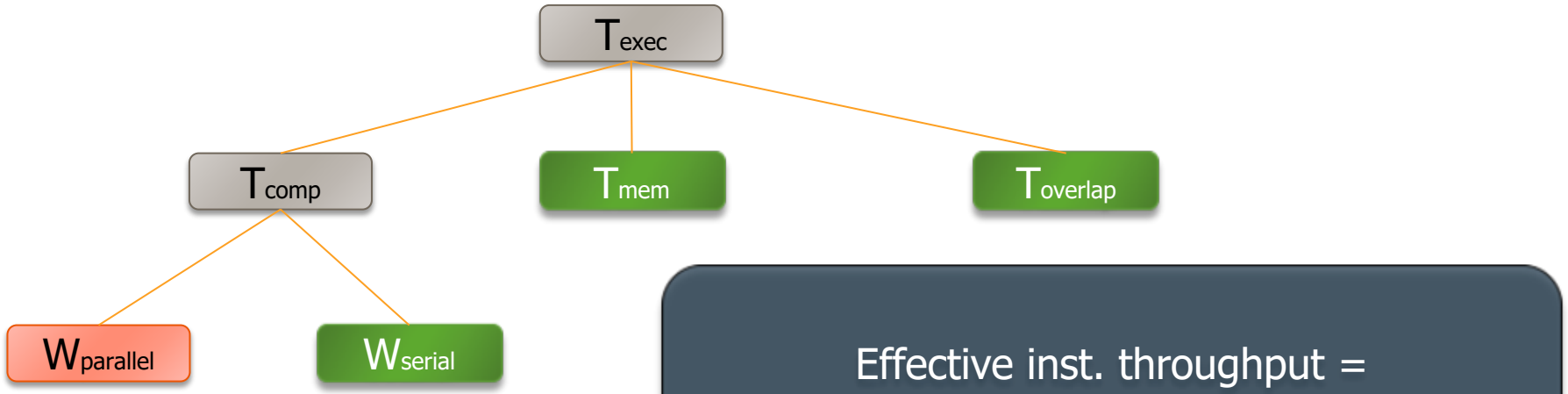
- $W_{parallel}$: Work executed in parallel (useful work)
- W_{serial} : Overhead due to serialization effects

Sim et al. (PPoP'12)

Georgia Tech comparch

Analytical Model

- $W_{parallel}$ is the amount of work that can be executed in parallel



Effective inst. throughput = f(warp_size, SIMD_width, # pipeline stages)

ITILP represents the number of instructions that can be parallely executed in the pipeline.

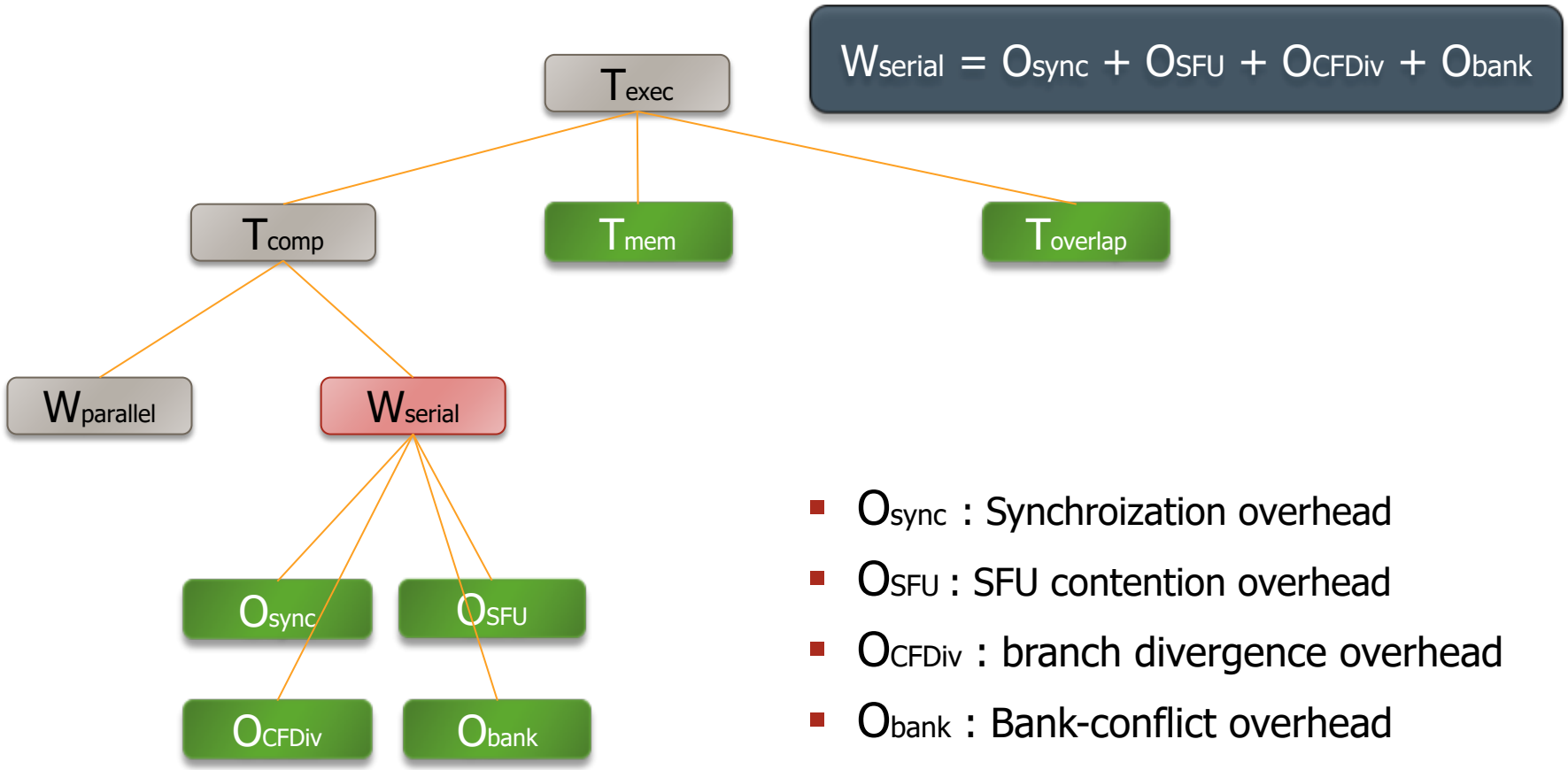
- $W_{parallel}$ = Total insts × Effective inst. throughput
- $\text{Effective Inst. throughput} = \frac{\text{average\_instruction\_latency}}{\text{ITILP}}$

Sim et al. (PPoP'12)

Georgia Tech comparch

Analytical Model

- W_{serial} represents the overhead due to serialization effects



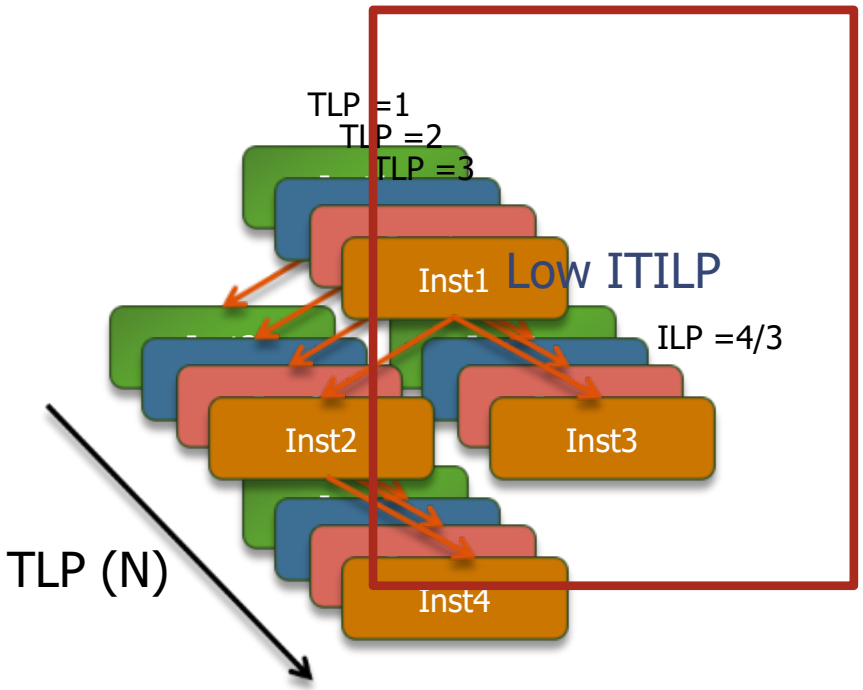
- O_{sync} : Synchroization overhead
- O_{SFU} : SFU contention overhead
- O_{CFDiv} : branch divergence overhead
- O_{bank} : Bank-conflict overhead

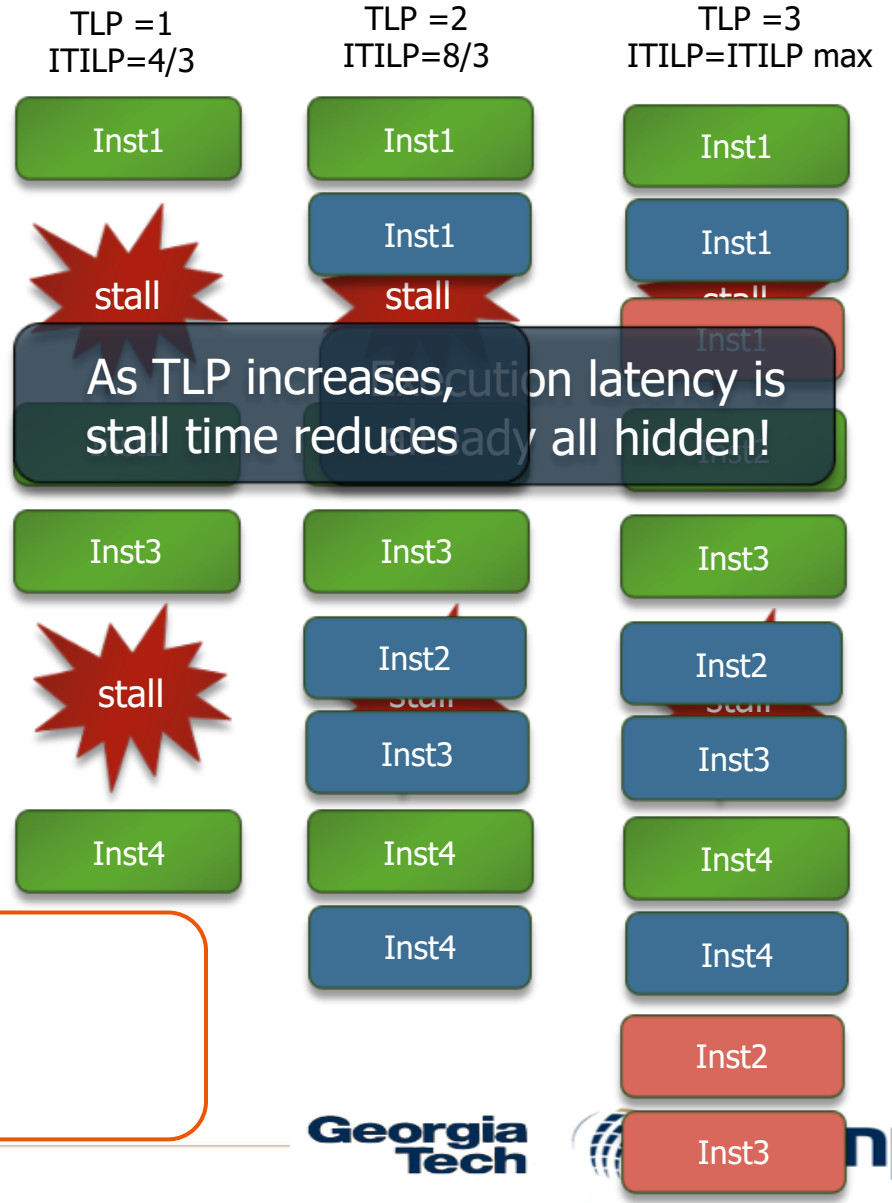
Sim et al. (PPoP'12)

Georgia Tech comparch

Analytical Model
ITILP (Inter-thread ILP)

- ITILP is inter-thread ILP.





$$ITILP = MIN(ILP \times N, ITILP_{max})$$

$$ITILP_{max} = avg\_inst\_lat / \frac{warp\_size}{SIMD\_width}$$

Sim et al. (PPoP'12)

Georgia Tech

nparch

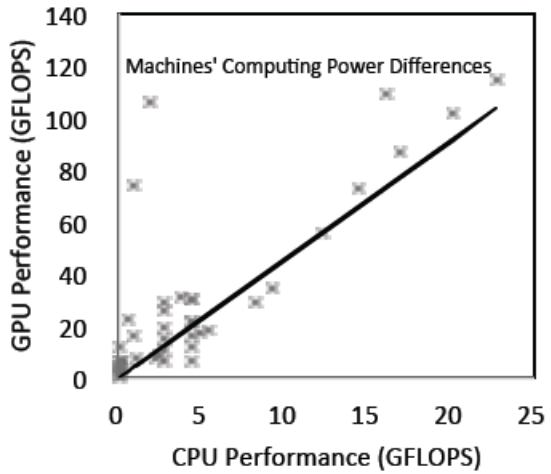
GPU Performance Predictor

- But, we don't have a GPU code yet.

$$\text{Machine\_Scale} = \frac{Peak\_GPU\_FLops}{Peak\_CPU\_FLops}$$

$$\text{Machine\_Scale\_SIMD} = \begin{cases} \frac{Machine\_Scale}{SIMD\_width} & vectorized \\ Machine\_Scale & otherwise \end{cases}$$

$$\text{GPU\_time} = \frac{CPU\_time}{Machine\_Scale\_SIMD}$$



Simply estimate the optimistic GPU performance

The performance benefits are heavily dependent on vectorizations

Consider Data Movements

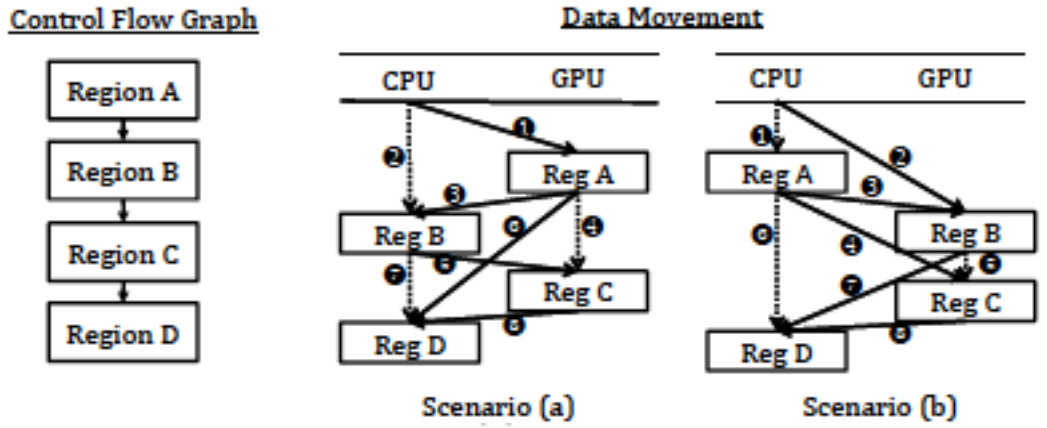
Image source: Ultra street fighter 2

Data transfer from CPU to GPU

- Tc= CompTime_CPU vs. Tg=CompTime_GPU + TransferTime_GPU
- If Tc<<Tg or Tc>>Tg, either CPU or GPU
  - Not: now let's worry about caches

Data Movements and Caching Effects

Region D's cache behavior



Data from	Baseline (ALL-CPU)	Scenario (a)	Performance
Region A (❻)	Hit if B & C don't evict data from A	Miss	Degrade
	Miss if B & C evict data from A	Miss	Same
Region B (❼)	Hit if C doesn't evict data from B	Hit	Same
	Miss if C evicts data from B	**Hit**	**Improve**
Region C (❽)	Hit	Miss	Degrade

Comparisons with all-CPU

Data from	Baseline (ALL-CPU)	Scenario (b)	Performance
Region A (❻)	Hit if B & C don't evict data from A	Hit	Same
	Miss if B & C evict data from A	**Hit**	**Improve**
Region B (❼)	Hit if C doesn't evict data from B	Miss	Degrade
	Miss if C evicts data from B	Miss	Same
Region C (❽)	Hit	Miss	Degrade

Offloading scenarios could change cache hits & miss behavior because dynamic working set size is changed

Georgia Tech comparch

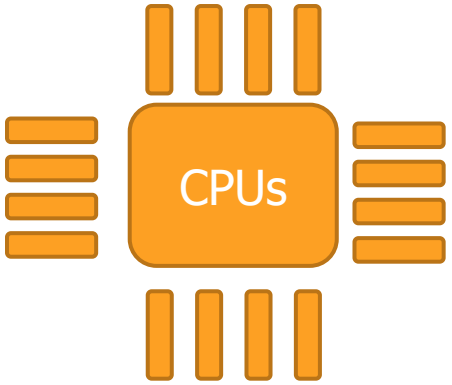
How to predict memory behavior?

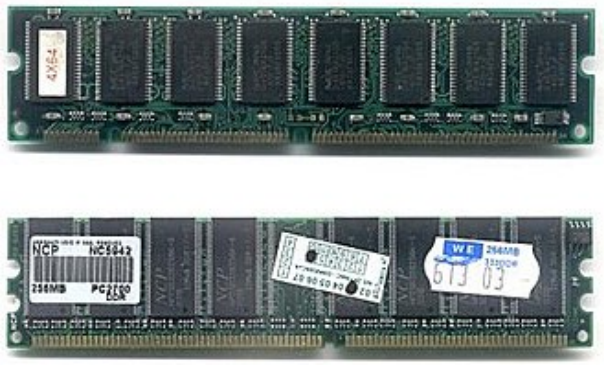
- Run-time profiling.
  - Use CPU code for the memory behavior
    - Critical component is measuring working set size and data sharing patterns
    - These are compute platform neutral
  - Different offloading scenarios to emulate



Case study 2: CPU vs. PIM

Execution Models of PIM



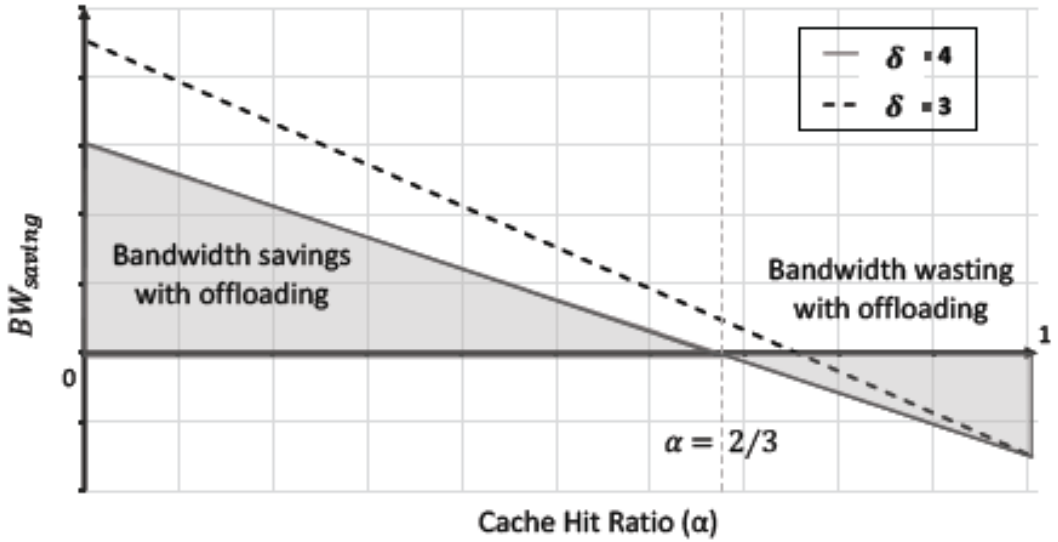


Almost no separate buffer:
only computation
(offloading instructions)

Full Processor
(offloading compute
kernels)

Instruction Offloading Benefit Modeling

BW not sensitive areas



$$BW_{reg} = \left(2N_c(1-\alpha)\times 6\right) + \left(N_{reg}(1-\beta)\times 6\right) \text{ flits.} \quad (1)$$

BW_{reg}: Regular bandwidth usage (without offloading)
N_c: Number of the candidate execution
N_{reg}: Number of the regular memory requests execution
α: Average LLC hit ratio of the candidate
β: Average LLC hit ratio of regular memory requests

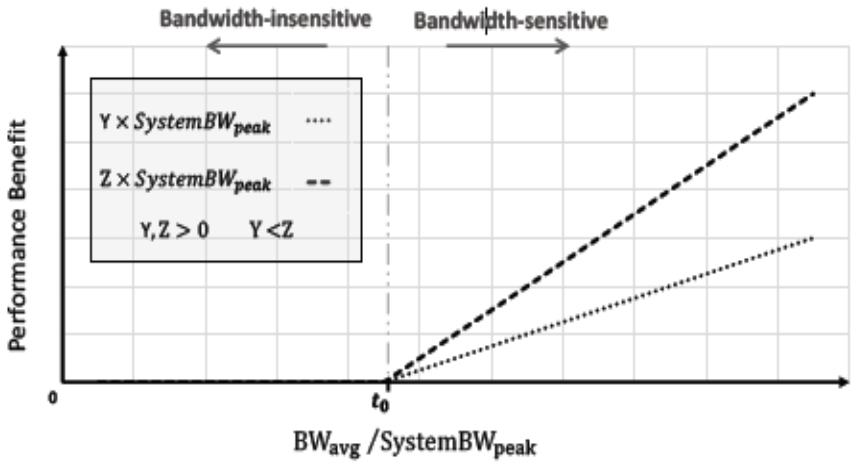
$$BW_{offload} = \left(N_c \times \delta\right) + \left(N_{reg}(1-\beta)\times 6\right) \text{ flits.} \quad (2)$$

$BW_{offload}$: Bandwidth usage with offloading
δ: Total packet size of request and response for offloading a candidate in flits

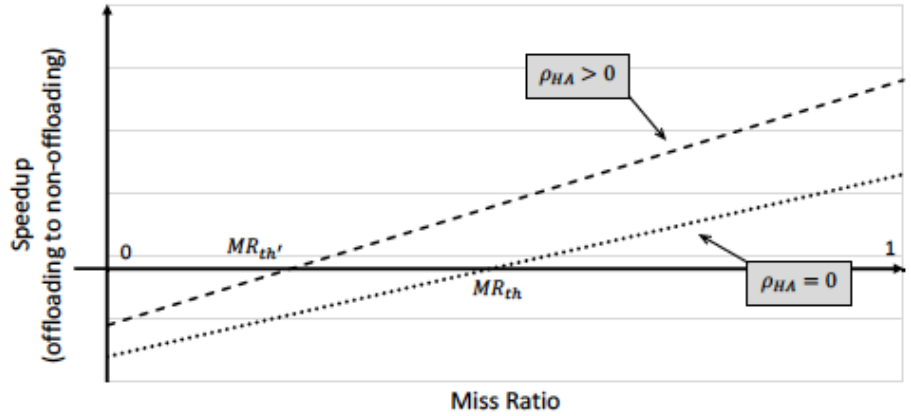
$$\begin{aligned} BW_{saving} &= BW_{reg} - BW_{offload} \\ &= 12N_c(1-\alpha) - \delta N_c = N_c\left(12(1-\alpha)-\delta\right) \text{ flits} \\ &= \begin{cases} 3N_c(3-4\alpha) \text{ flits}, & \text{if } \delta = 3 \\ 4N_c(2-3\alpha) \text{ flits}, & \text{if } \delta = 4. \end{cases} \end{aligned}$$

from: Hadidi'17

BW saving benefits & Cache behavior changes



BW insensitive applications



Heavily dependent on atomic instruction's overhead

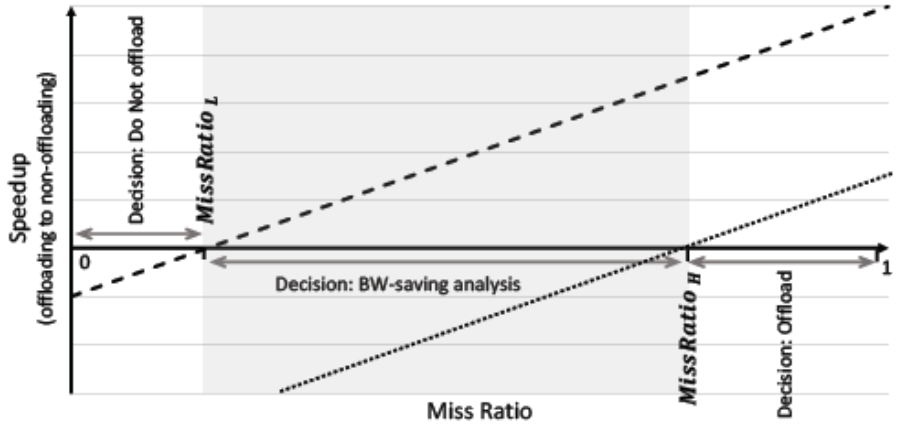
Working set size changes. Cache hit rate changes

Not all BW savings are useful

$$BW'_{saving} = (BW_{reg} - BW_{offload}) / BW_{reg} = BW_{saving} / BW_{reg},$$

BW'_{saving}: Normalized bandwidth savings
BW_{reg}: Regular bandwidth usage (without offloading)
$BW_{offload}$: Bandwidth usage with offloading

BW insensitive applications



from: Hadidi'17

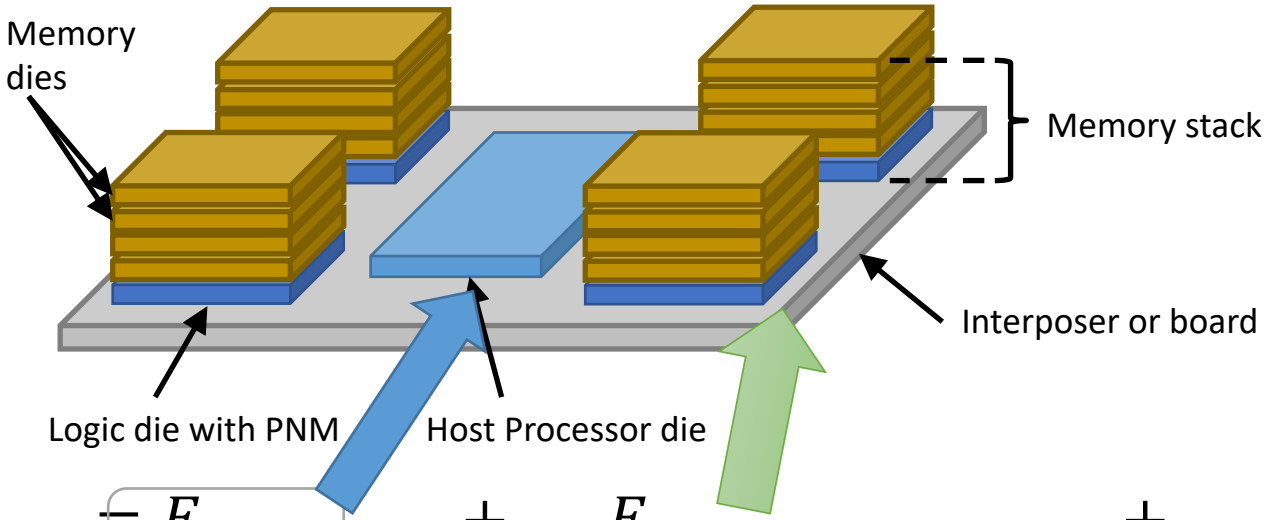
Kernel Offloading Cases

- Performance gain cases
  - Bandwidth effect
  - Memory access latency
  - Shallow cache hierarchy
  - Reduced coherence traffic
- Performance slowdown cases
  - Low performance of PNM cores
  - Small cache sizes in PNM
  - NUMA effects

Computing power effects were the most significant factor

from: Hadidi'17

Energy Model



$$E_{total} = E_{Host} + E_{PNM-Memory} + E_{transfer}$$

$$E_{core} + E_{uncore} + E_{cache}$$

$$P_{host}^{uncore} \times N_{channel} \times t_{total}$$

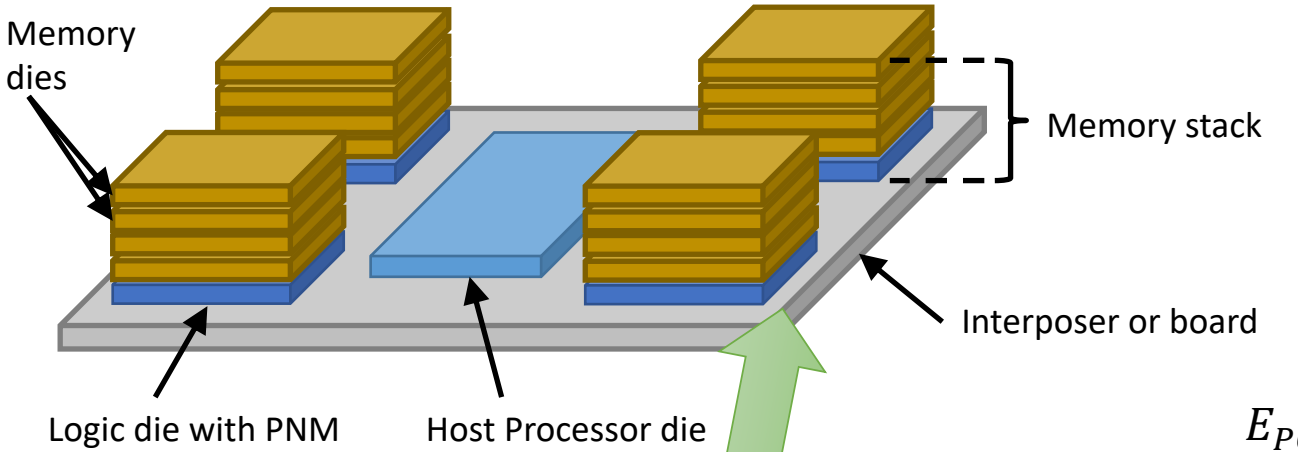
$$P_{host}^{active} \sum t_{active} + P_{host}^{idle} \sum t_{idle}$$

$$E_{cache}^{static} + E_{cache}^{dynamic}$$

$$\sum_L P_{SRAM}^{leakage} \times t_{total} \times N_L^{SRAM} \qquad \sum_L E_L^{access} \times N_L^{access}$$

Source:kim'15

Energy Model



$$E_{total} = E_{Host} + E_{PNM-Memory} + E_{transfer}$$

$$E_{transfer}: \quad E_{PCB} \times N_{PCB}^{transfer}$$

$$E_{PNM-Memory} = E_{PNM-core} + E_{PNM-uncore} + E_{PNM-cache} + E_{memory}$$

$$E_{PNM-uncore} \rightarrow (N_{link} \times P_{link} + P_{misc}) \times t_{total}$$

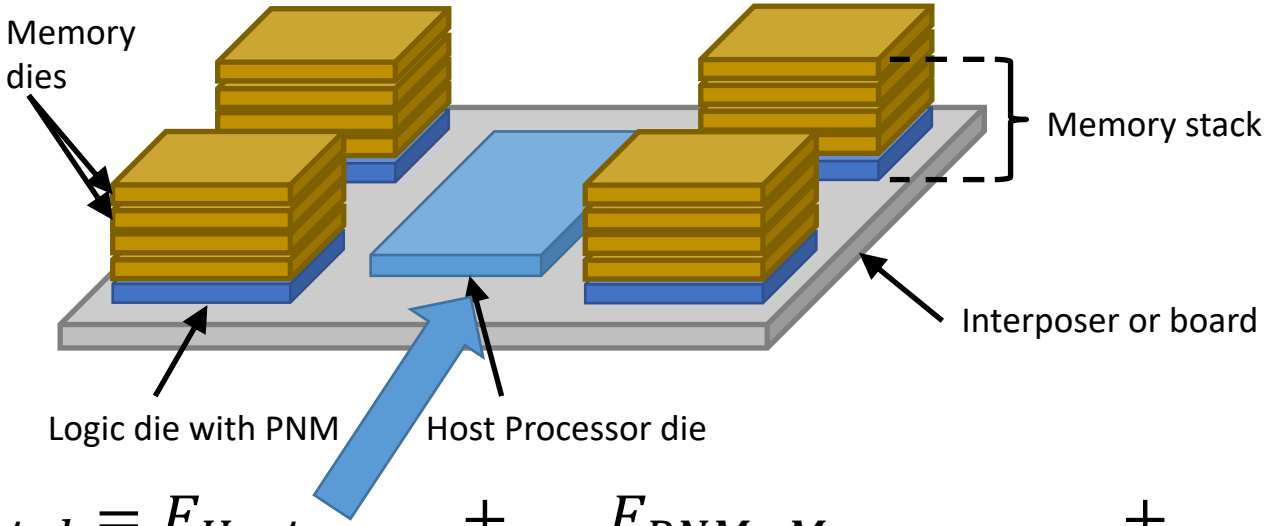
$$E_{memory} \rightarrow E_{static} + E_{dynamic}$$

$$E_{static} \rightarrow P_{DRAM}^{background} \times t_{total}$$

$$E_{dynamic} \rightarrow (E_{access} + E_{TSV}) \times N_{access}$$

Source:kim'15

Energy Model



$$E_{total} = E_{Host} + E_{PNM-Memory} + E_{transfer}$$

$$E_{core} + E_{uncore} + E_{cache}$$

$$P_{host}^{active} \sum t_{active} + P_{host}^{idle} \sum t_{idle}$$

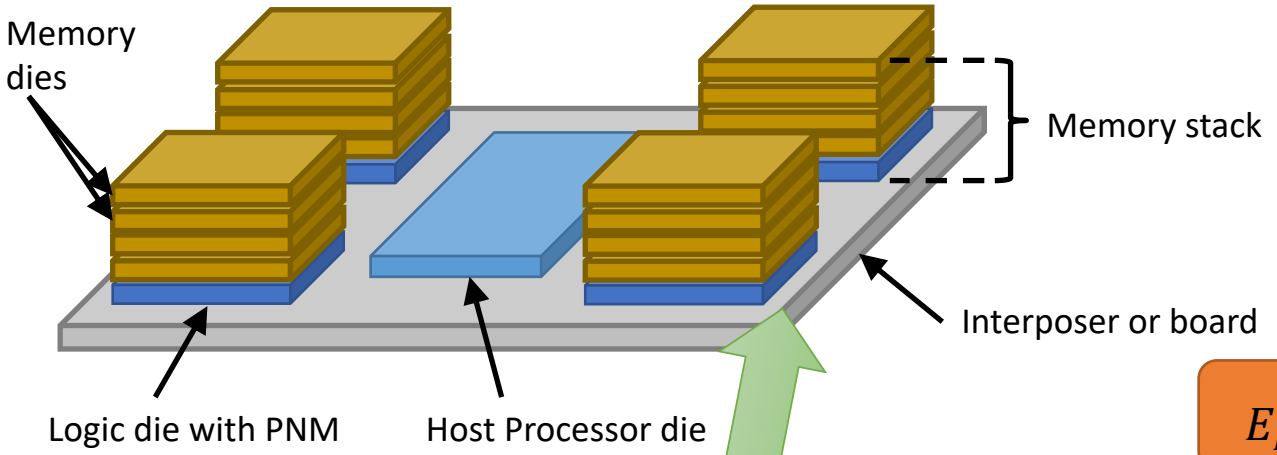
$$P_{host}^{uncore} \times N_{channel} \times t_{total}$$

$$E_{cache}^{static} + E_{cache}^{dynamic}$$

$$\sum_L P_{SRAM}^{leakage} \times t_{total} \times N_L^{SRAM} \quad \sum_L E_L^{access} \times N_L^{access}$$

Source:kim'15

Energy Model



$$E_{total} = E_{Host} + E_{PNM-Memory} + E_{transfer}$$

$$E_{transfer} = E_{PCB} \times N_{PCB}^{transfer}$$

$$E_{PNM-Memory} = E_{PNM-core} + E_{PNM-uncore} + E_{PNM-cache} + E_{memory}$$

$$E_{PNM-uncore} = (N_{link} \times P_{link} + P_{misc}) \times t_{total}$$

$$E_{memory} = E_{static} + E_{dynamic}$$

$$E_{static} = P_{DRAM}^{background} \times t_{total}$$

$$E_{dynamic} = (E_{access} + E_{TSV}) \times N_{access}$$

Source:kim'15

Energy Values in Model

Parameter	Values	Source
SRAM Leakage Power ($P_{SRAM}^{leakage}$)	4.050nW per cell	CACTI
L1 cache access energy (E_1^{access})	0.494nJ per access	CACTI
L2 cache access energy (E_2^{access})	3.307nJ per access	CACTI
L3 cache access energy (E_3^{access})	6.995 nJ per access	CACTI
DRAM background power ($P_{DRAM}^{background}$)	0.470W per package	Pugsley et al. [ISPASS'14]
DRAM access energy (E_{access})	23.034nJ per access	Udipi et al. [ISCA'10]
TSV transfer energy	0.078 pJ per bit	3D CACTI, Woo [HPCA'10]
Global transfer energy in PCB (E_{PCB})	4.700 pJ per bit	Pugsley et al. [ISPASS'14]
Host core Active/Idle power $P_{host}^{active}/P_{host}^{idle}$	10W / 1W	Intel Xeon E3-1275
PNM core active/idle power $P_{PNM}^{active}/P_{PNM}^{idle}$	80mW / 8mW	Pugsley et al. [ISPASS'14]
$Serdes\ Power\ (P_{link})$	1.445W per link	Sandhu [WETI' 12]
$PNM\ uncore\ misc\ power\ (P_{misc})$	2.890W per package	Sandhu [WETI' 12]

Source:kim'15

Georgia Tech comparch

Timing Simulation

- Cycle-level timing simulator
  - Macsim+SST[Sandia]
- Measured values
  - $\sum t_{active}$ (Host, PNM)
  - N_L^{access} (each level of cache, DRAM)
  - $N_{PCB}^{transfer}$
  - $\sum t_{idle}$ is assumed to be 0

Source:kim'15

Evaluated Applications

11 applications from Mantevo benchmark suite

- Known to have very large memory footprint and bandwidth

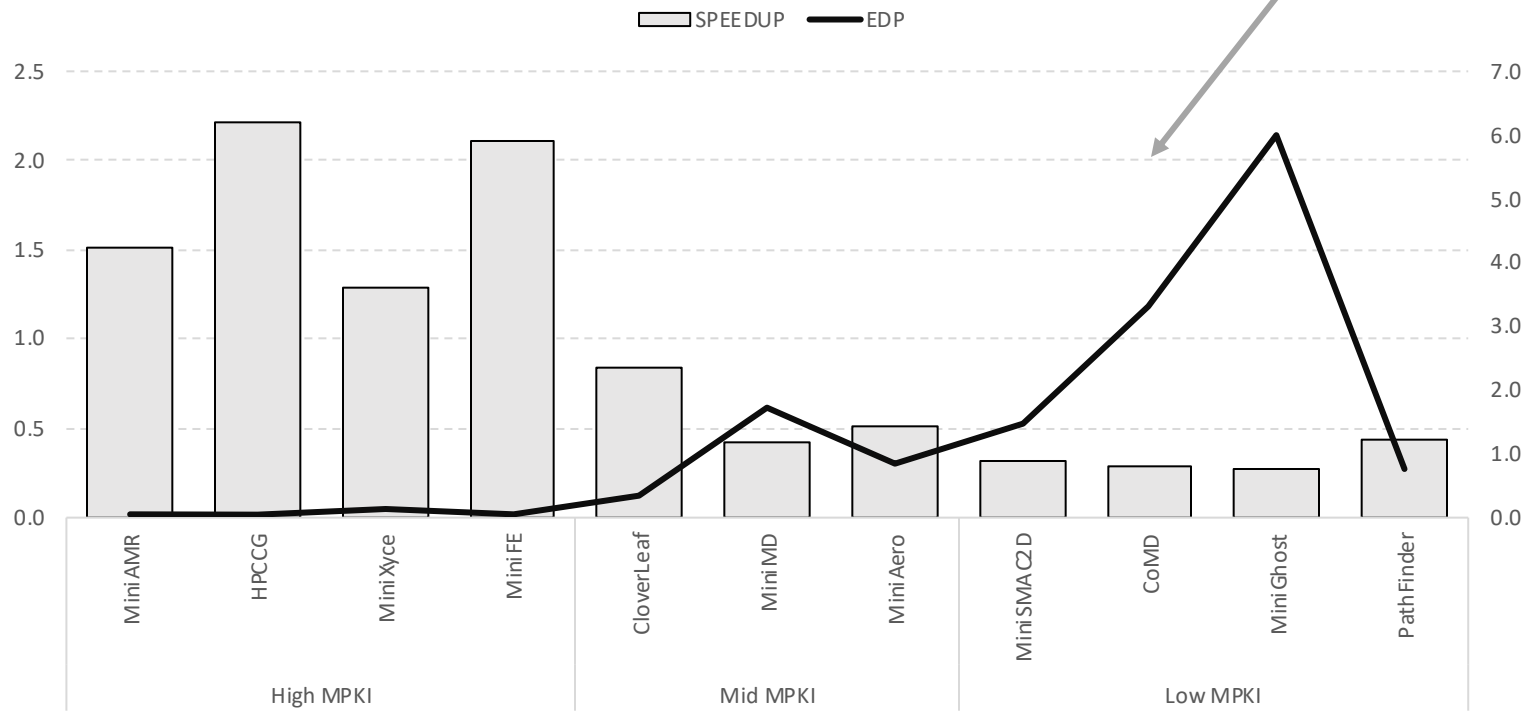
- For this study, we use only single threaded version.
  - Multi-threaded version will be future work

Class	Benchmark	MPKI	ILP
High MPKI	MiniAMR	60.915	1.369
	HPCCG	35.914	1.111
	MiniXyce	34.032	1.467
	MiniFE	28.383	1.104
Mid MPKI	CloverLeaf	6.924	1.431
	MiniMD	4.858	3.082
	MiniAero	3.203	1.867
Low MPKI	MiniSMAC2D	0.642	1.399
	CoMD	0.352	3.556
	MiniGhost	0.105	3.864
	PathFinder	0.093	1.745

Source:kim'15

MPKI matters

Low MPKI benchmarks increases EDP significantly





Source of ENERG Savings

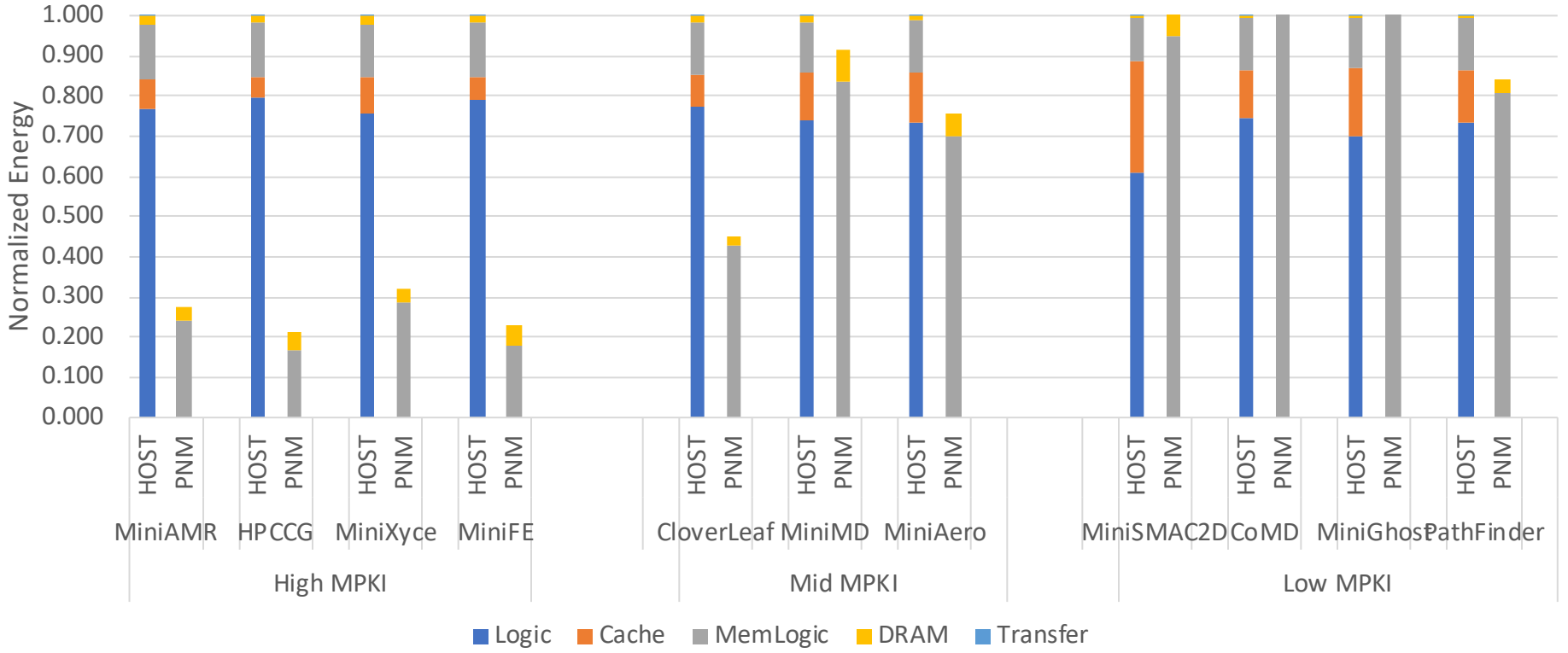
(1) Energy efficient PNM Cores + uncore parts

- 10W Host core vs. 80mW PNM Core

(2) Short memory accesses for cache unfriendly applications

- Memory latency for host core
  - L1 access + L2 access + L3 access + Interconnect+ PCB/TSV + DRAM
  - (3 cycles + 8 cycles + 30 cycles + communication latency + DRAM)
- Memory latency for PNM core
  - L1 access + TSV + DRAM
  - 3 cycles + TSV + DRAM
- Small cache increases DRAM accesses though

10X more power in PNM cores



- High MPKI benchmarks still show energy efficiency



Finding From the Energy Models

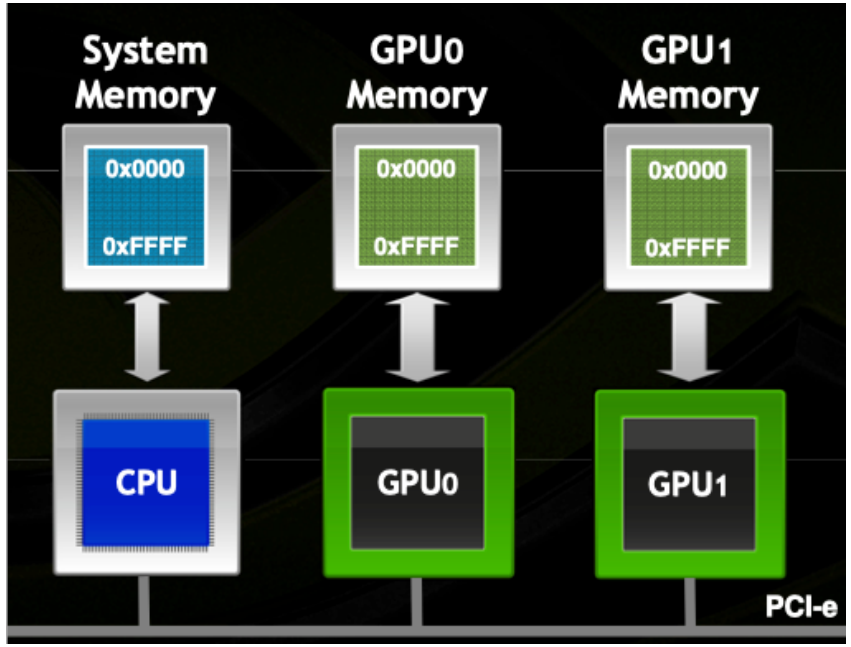
- Most energy savings come from exploiting power-efficient cores + uncore parts
- PNM allows to build energy efficient cores
  - SERDES energy consumption is significant
- Applications with high MPKI get the most significant savings in energy consumption without performance degradation
- Application ILP is also an important factor



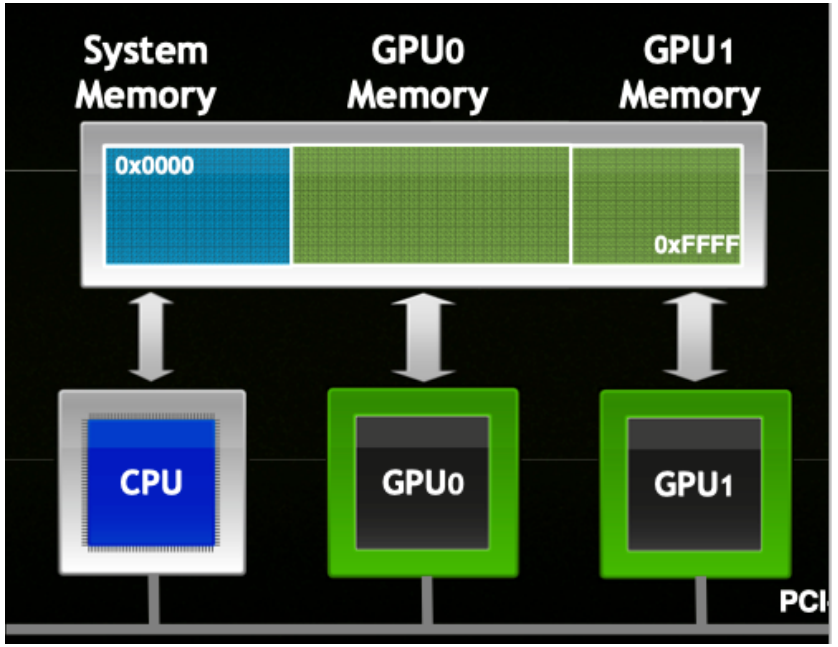
Case study 3: Unified Memory on CPU+GPU

Unified Virtual Memory (UVM) + Demand Paging

Multiple Address Spaces



Single Address Space



- Single virtual memory shared between processors
- Can run large GPU applications (> GPU Memory)

Georgia Tech comparch

Performance vs. Graph Size

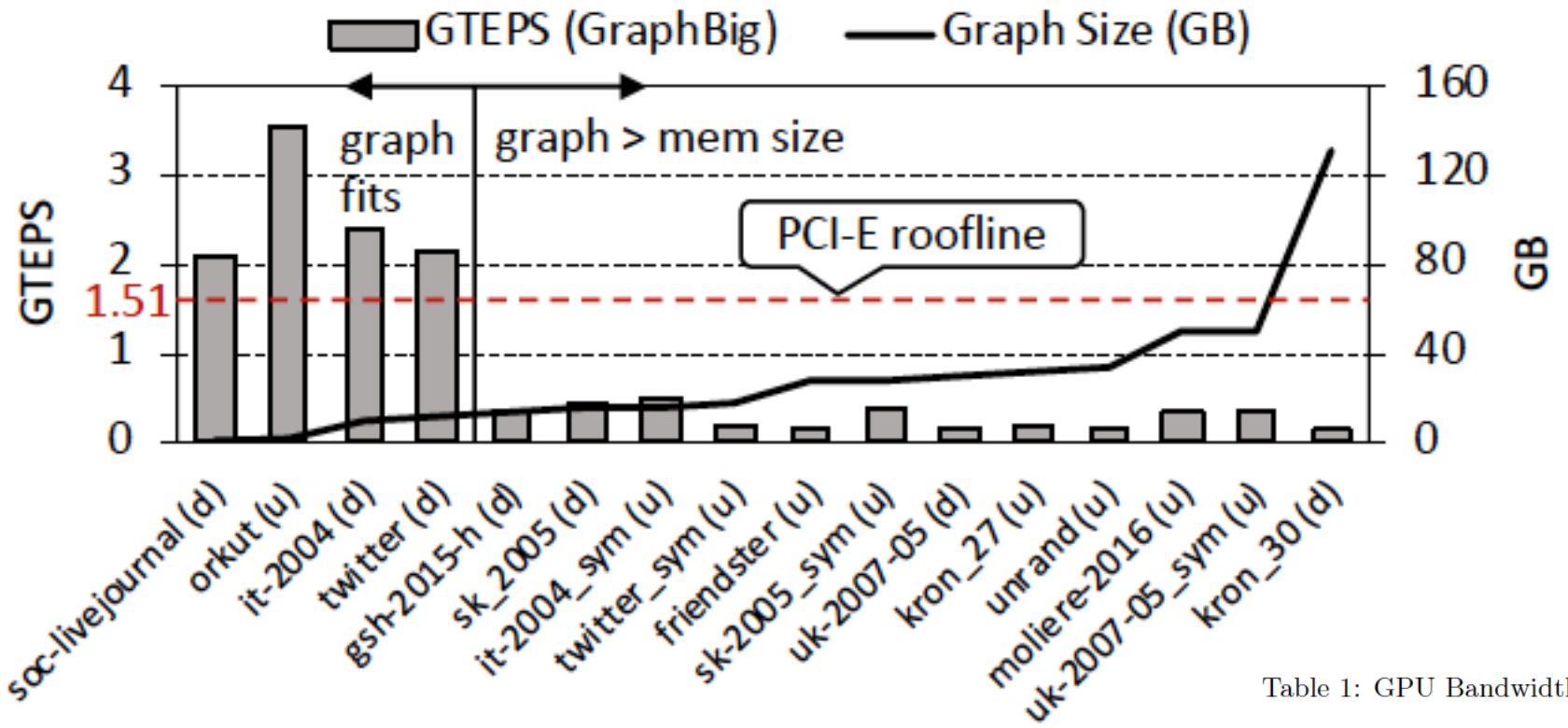


Table 1: GPU Bandwidth Characteristics

GPU	Mem.	HtoD Link	DtoD BW	HtoD BW
Titan Xp	12 GB	PCI-e 3.0	417.4 GB/s	12.1 GB/s

From:Gera'20


Conclusions

- Heterogeneous computing brings the question of when to use what.
- Modeling can provide good guidelines
  - Modeling memory behavior is always challenge
  - Run-time profiling or input analysis can be very useful