UiO : Department of Informatics
University of Oslo

A High-Precision, Hybrid GPU, CPU and RAM Power Model for the Tegra K1 SoC

Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, Pål Halvorsen
{krisrst, haakonks, griff, paalh}@ifi.uio.no

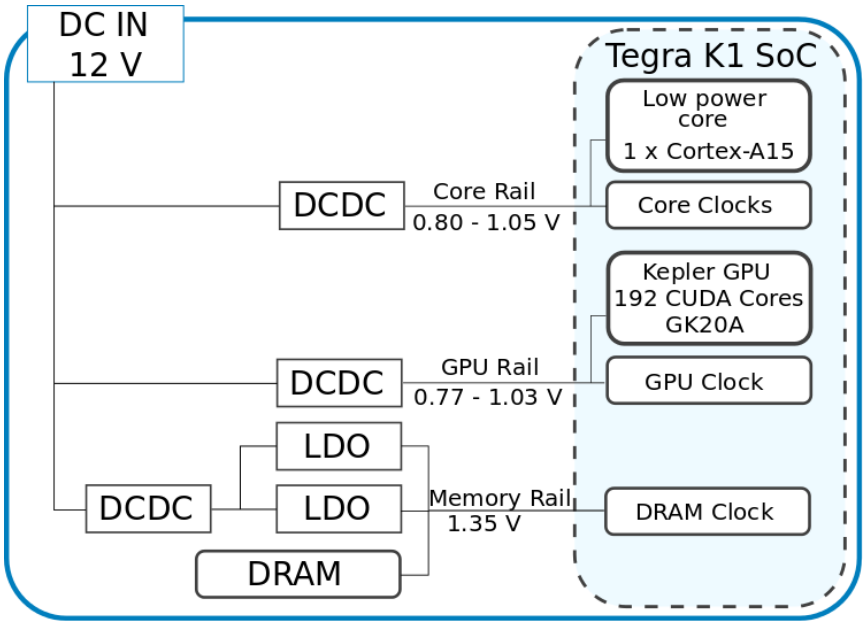
[simula . research laboratory]

Mobile Multimedia Systems

- Tegra K1 – *mobile* multicore SoC
  - 192-core CUDA-capable GPU (+ CPU cores)
  - **Enables:** smart phones, tablets, laptops, drones, satelites..
  - **Applications:** Video filtering operations, game and data streaming, machine learning..
- Energy optimisation
  - Battery limitation
  - Environmental aspect
  - Device failure
  - Thermal Management

- *How can we understand the relationship between software activity and power usage?*

Tegra K1 SoC Architecture: Rails and Clocks



- Power on a rail can be described using the *standard CMOS equations*

$$P_{rail} = P_{stat} + P_{dyn}$$

$$P_{stat} = V_{rail} I_{leak} \qquad P_{dyn} = \alpha C V_{rail}^2 f$$

Transistor leakage

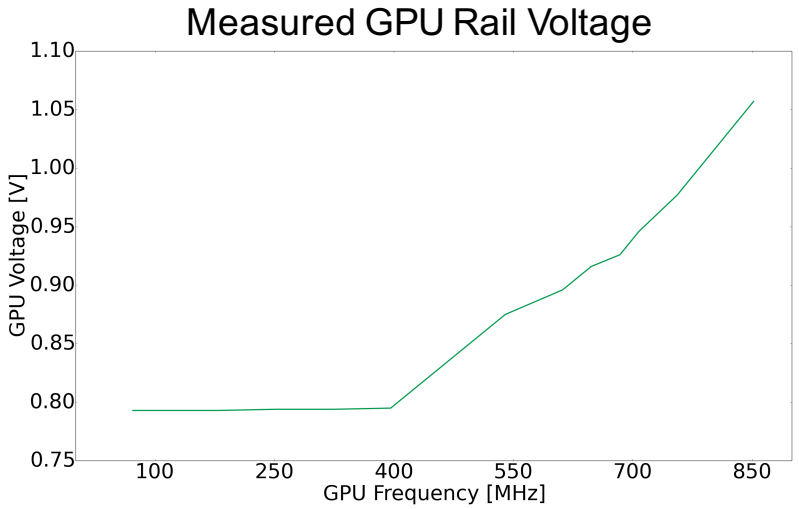
Capacitive load per cycle

Cycles per second

- Rail voltage V_{rail}
  - Increases with clock frequency
- **Total power**
  - ..is the sum of power of all rails

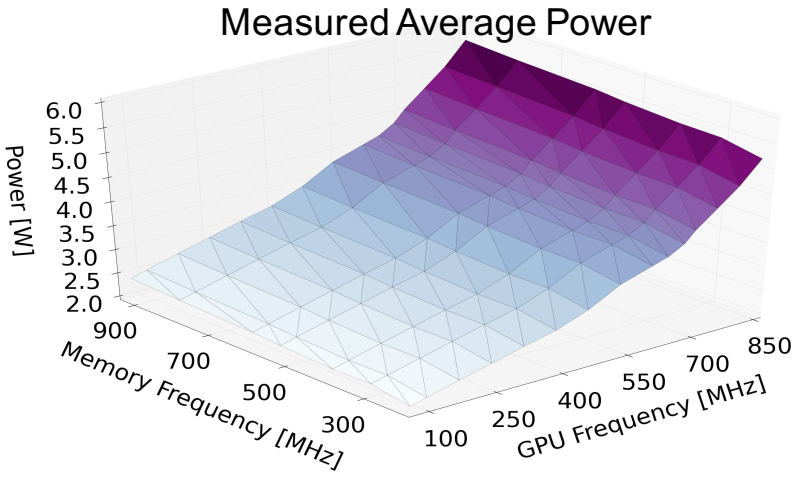
Tegra K1 SoC Architecture: Rails and Clocks

- Clock frequency, rail voltage and power usage are **deeply coupled**
  - Increasing clock frequency increases voltage, and vice versa



- From previous slide: power $\propto V^2$

$$P_{stat} = V_{rail} I_{leak} \qquad P_{dyn} = \alpha C V_{rail}^2 f$$



Rate-Based Power Models

- Widespread use since 1997 (Laura Marie Feeney)
  - On-line power models for smart phones, such as *PowerTutor*
- Concept is simple
  - Power is *correlated* with utilisation levels
    - E.g. *rate* at which instructions are executed, or *rate* of cache misses
  - Multivariable, linear regression
  - A typical model for total power

$$P_{tot} = \beta_0 + \sum_{i=1}^{N_p} \beta_i \rho_i$$

Events per second

Constant base power

Cost ($\frac{W}{Event\ per\ second}$)

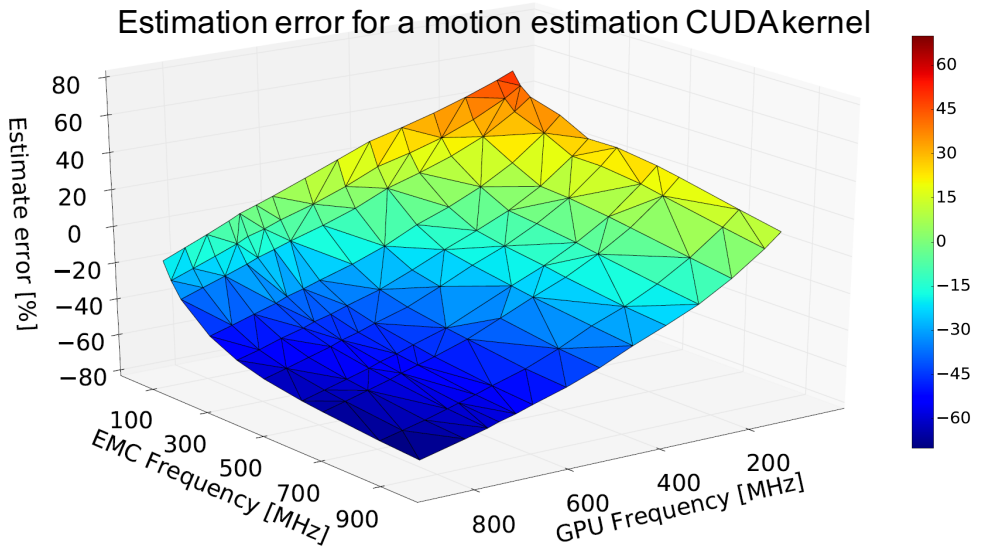
A Rate-Based Power Model for the Tegra K1

Device	Predictor (CUPTI and PERF)	Coefficient
GPU	L2 32B read transactions per second	-18.6 nW per eps
	L1 4B read transactions per second	0.0 nW per eps
	L1 4B write transactions per second	-3.7 nW per eps
	Integer instructions per second	6.2 pW per eps
	Float 32 instructions per second	6.6 pW per eps
	Float 64 instructions per second	279 pW per eps
	Misc. instructions per second	-300 pW per eps
	Conversion instructions per second	236 pW per eps
CPU	Active CPU cycles per second	887 pW per eps
	CPU instructions per second	1.47 nW per eps

- Model ignores
  - Voltage variations
  - Frequency scaling
- Negative coefficients (we «gain» power per event per second)

A Rate-Based Power Model for the Tegra K1

- Motion estimation CUDA-kernel
- Estimation error can be as high as 80 %, and for *some areas* (green) it is near perfect at 0 %



Rate-based models should be used with care over frequency ranges!

CMOS-Based Power Models

- Model switching capacitance αC directly for rails using the CMOS equations

$$P_{rail} = P_{stat} + P_{dyn}$$

$$P_{stat} \rightarrow V_{rail} I_{leak} \qquad P_{dyn} \rightarrow \alpha C V_{rail}^2 f$$

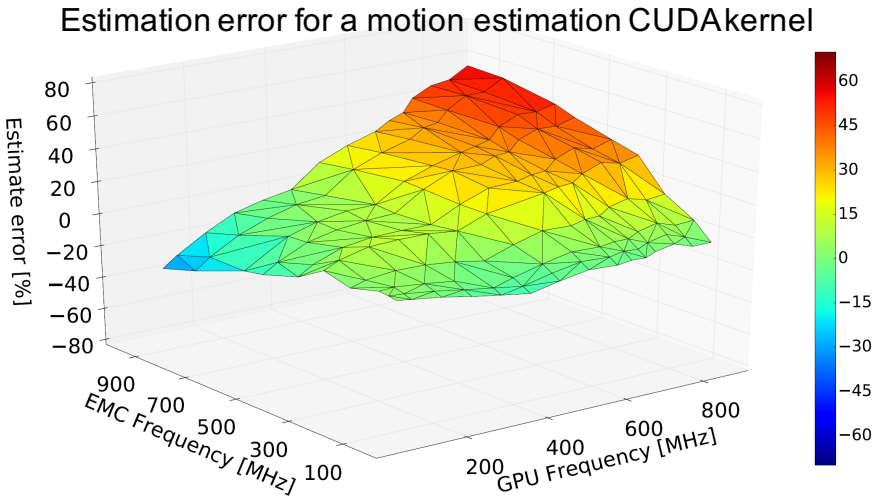
- Use several CPU-GPU-memory frequencies, log rail voltages and power
  - Estimate I_{leak} and αC using regression

- Advantages
  - Voltages and leakage currents considered

CMOS-Based Power Models

- Better than the rate-based one
- Accuracy generally > 85 %, but only about 50 % accurate on high frequencies

- Disadvantages / reasons
  - αC varies depending on workload
  - Switching activity in one domain (memory) varies depending on frequency in another (GPU)
  - ..but model assumes independent relationship between αC and frequency in other domains



Building High-Precision Power Models

- The problem is in the *dynamic part* of the CMOS equation:

$$P_{rail} = V_{rail} I_{leak} + \alpha C f V_R^2$$

  - ..which doesn't consider that αC on a rail is actually depending on frequencies in other domains (e.g. memory rail αC depends on CPU and GPU frequency)

- We now want to express switching activity in terms of **measurable hardware activity** (similarly to rate-based models):

$$P_{rail} = V_{rail} I_{leak} + \sum_{i=1}^{N_R} C_{R,i} \rho_{R,i} V_R^2$$

Number of utilisation predictors on rail R

Capacitive load per event per second

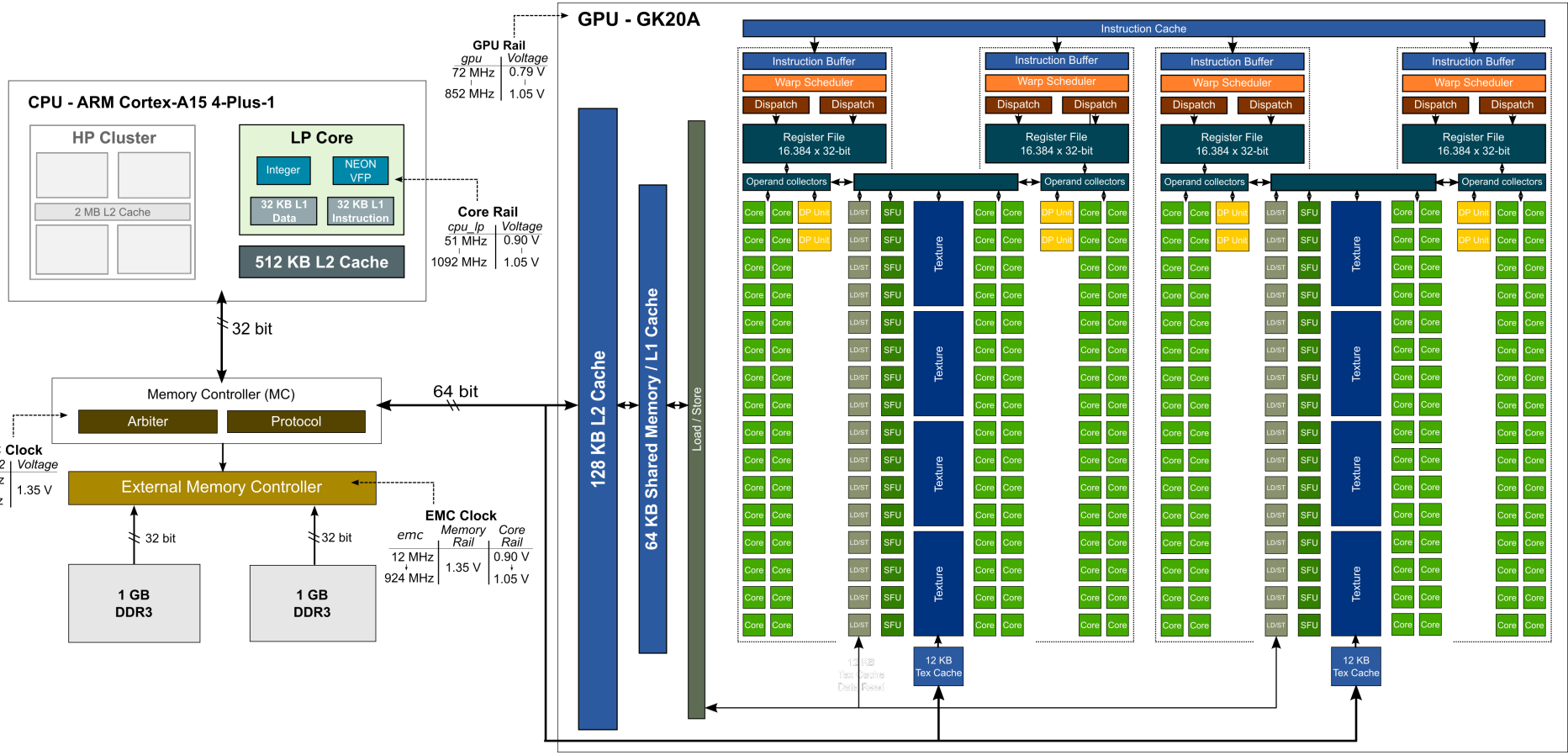
Hardware utilisation predictor (events per second)

Understanding Hardware Activity

$$P_{rail} = V_{rail} I_{leak} + \sum_{i=1}^{N_R} C_{R,i} \rho_{R,i} V_R^2$$

- We need to measure *hardware activity* in each of the three rails
  - Memory, Core and GPU rails
- **What constitutes good hardware activity predictors?**
  - $\rho_{R,i}$ can be cache misses, cache writebacks, instructions, cycles..
  - Should ideally cover all hardware activity in a rail
  - **Major task in understanding and/or guessing what is going in in hardware**

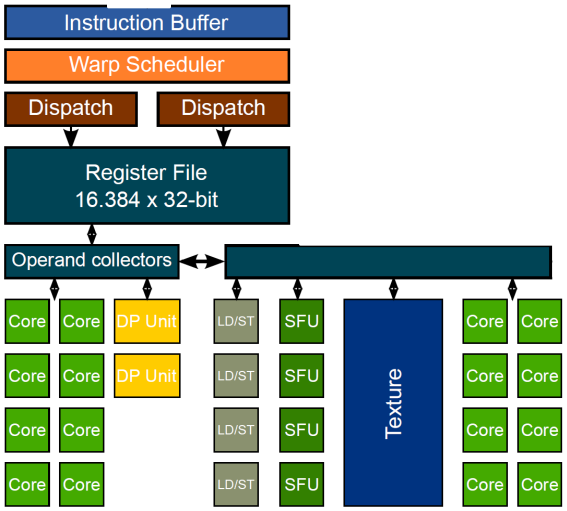
Understanding Hardware Activity: GPU



Understanding Hardware Activity: GPU Cores

- NVIDIA provides CUPTI
  - **Fine-grained instruction counting**

- We can therefore estimate switching capacitance per instruction type
  - Some out of scope, such as Special Function Unit (SFU) instructions (sin, cos, tan, ..)

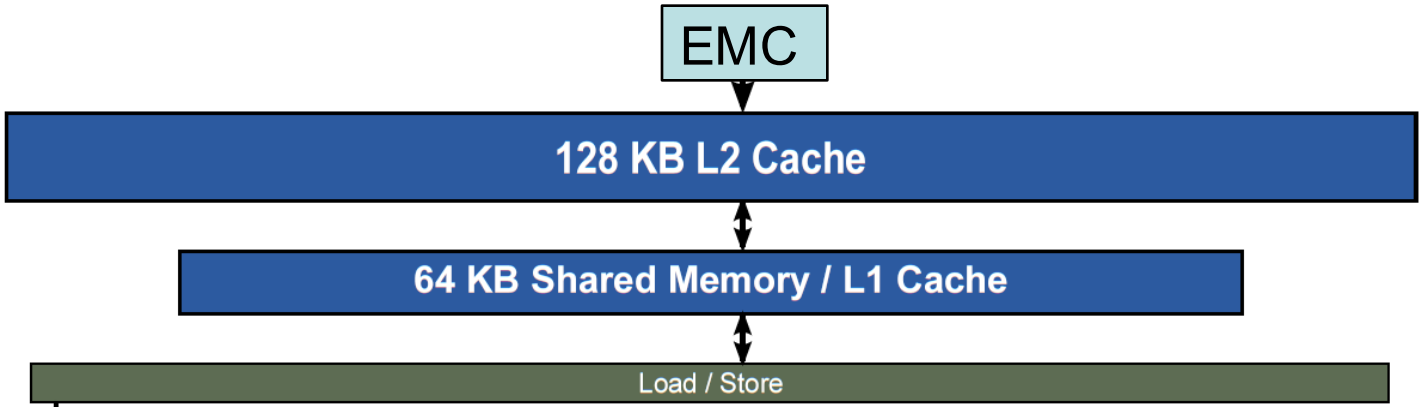


Core block dynamic power predictors

HPC Name	Description
inst_integer	Integer instructions
inst_bit_convert	Conversion instructions
inst_control	Control flow instructions
inst_misc	Miscallaneous instructions
inst_fp_32/64	Floating point instructions

Understanding Hardware Activity: GPU Memory

- Easily the **most complex part** of dynamic power because memory **is so flexible**
  - ..and because documentation is confusing (**nvprof --query-events --query-metrics)**
- L2 cache serves read requests
  - (CUPTI HPC) **l2_subp0_total_read_sector_queries**
  - HPC for writes (**l2_subp0_total_write_sector_queries**), but we cannot estimate a capacitance cost for it – this indicates that L2 cache is **write-back**
    - **Which is surprising!**



Understanding Hardware Activity: GPU Memory

- L1 GPU cache has many uses:
  - Caching global (RAM) reads not writes
  - Caching local data (function parameters) and register spills
  - Shared memory (read and written by thread blocks)

- No CUPTI HPC counts **raw L1 reads and writes**
  - Must combine the HPCs for all types of L1 accesses to make our own counter:

HPC Name	Description
l1_global_load_hit	L1 cache hit for global (RAM) data
l1_local_{store/load}_hit	L1 register spill / local cache
l1_shared_{store/load}_transactions	Shared memory
shared_efficiency	

GPU Summary

- Dynamic power (hardware activity predictors)
  - $\rho_{GPU,int}, \rho_{GPU,f32}, \rho_{GPU,f64}, \rho_{GPU,cnv}, \rho_{GPU,msc}$: Integer, float32, float64, conversion and misc. instructions per second
  - $\rho_{GPU,l2r}$, $\rho_{GPU,l1r}$, $\rho_{GPU,l1w}$: L2 reads, L1 reads and L1 writes per second
  - $\rho_{GPU,clk}$: Active cycles per second (not subject to clock gating)

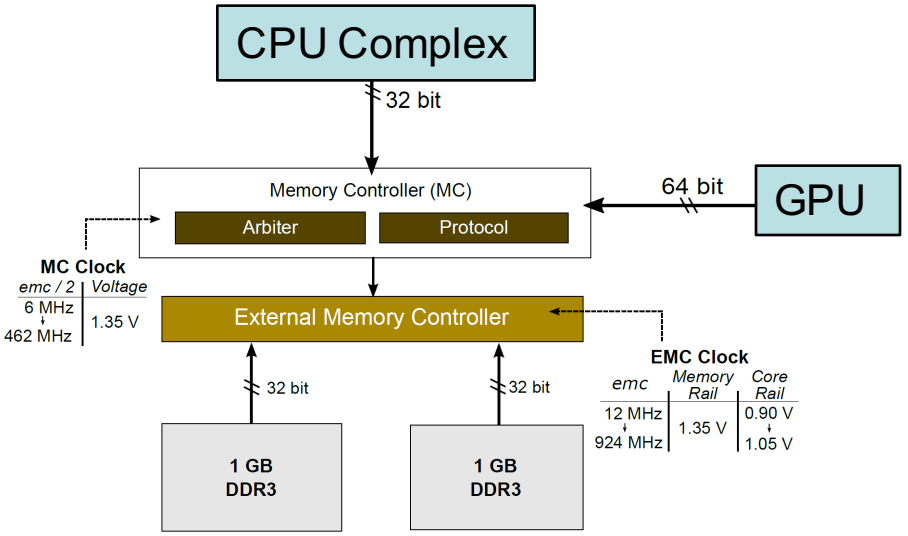
- Static power
  - $I_{GPU,leak}$: GPU leakage current when rail on

- **Total power for GPU rail:**

$$P_{GPU} = V_{GPU} I_{GPU,leak} + \sum_{i=1}^{N_{GPU}} C_{GPU,i} \rho_{GPU,i} V_{GPU}^2$$

Understanding Hardware Activity: Memory

- Monitoring RAM activity is **very challenging**
- The Tegra K1 however has an **activity monitor**
  - **emc_cpu**: total RAM cycles spent serving CPU requests
  - **emc_gpu**: total RAM cycles spent serving GPU requests
- In addition, the RAM continuously spends cycles (no matter if it is inactive) to maintain its own consistency



Memory Summary

- Dynamic power (hardware activity predictors)
  - $\rho_{MEM,cpu}$, $\rho_{MEM,gpu}$: Active memory cycles from CPU and GPU workloads
  - $\rho_{MEM,clk}$: Active cycles per second (not subject to clock gating)

- Static power
  - Memory is driven by LDO regulators and the rail voltage is always 1.35 V
  - Therefore it is not possible to isolate leakage current

- **Total power for memory rail:**

$$P_{MEM} = \sum_{i=1}^{N_{MEM}} C_{MEM,i} \rho_{MEM,i} V_{MEM}^2$$

$$V_{MEM} = 1.35\ V$$

LP Core Summary

- Dynamic power
  - $\rho_{HP,ipc}$: Instructions per cycle
  - $\rho_{HP,clk}$: Active cycles per second (subject to clock gating)
- Static power
  - $I_{core,leak}$: Core rail leakage current (always present)

- **Total power for core rail:**

$$P_{core} = V_{core} I_{core,leak} + \sum_{i=1}^{N_{core}} C_{core,i} \rho_{core,i} V_{core}^2$$

Finding the Right Answer

$$P_{jetson} = \sum_{R \in \mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base}$$

GPU, Core and memory rail

$$P_R = \sum_{i=1}^{N_R} C_{R,i} \rho_{R,i} V_R^2 \qquad P_{R,stat} = V_{rail} I_{R,leak}$$

- Unknown variables
  - The switching capacitances $C_{R,i}$
  - The leakage currents $I_{R,leak}$
  - And the base power P_{base}

- The resulting expression is **linear** where all voltages and predictors are known
  - Which means we can find the coefficients using **multivariable linear regression**
  - **..If** we are careful enough..

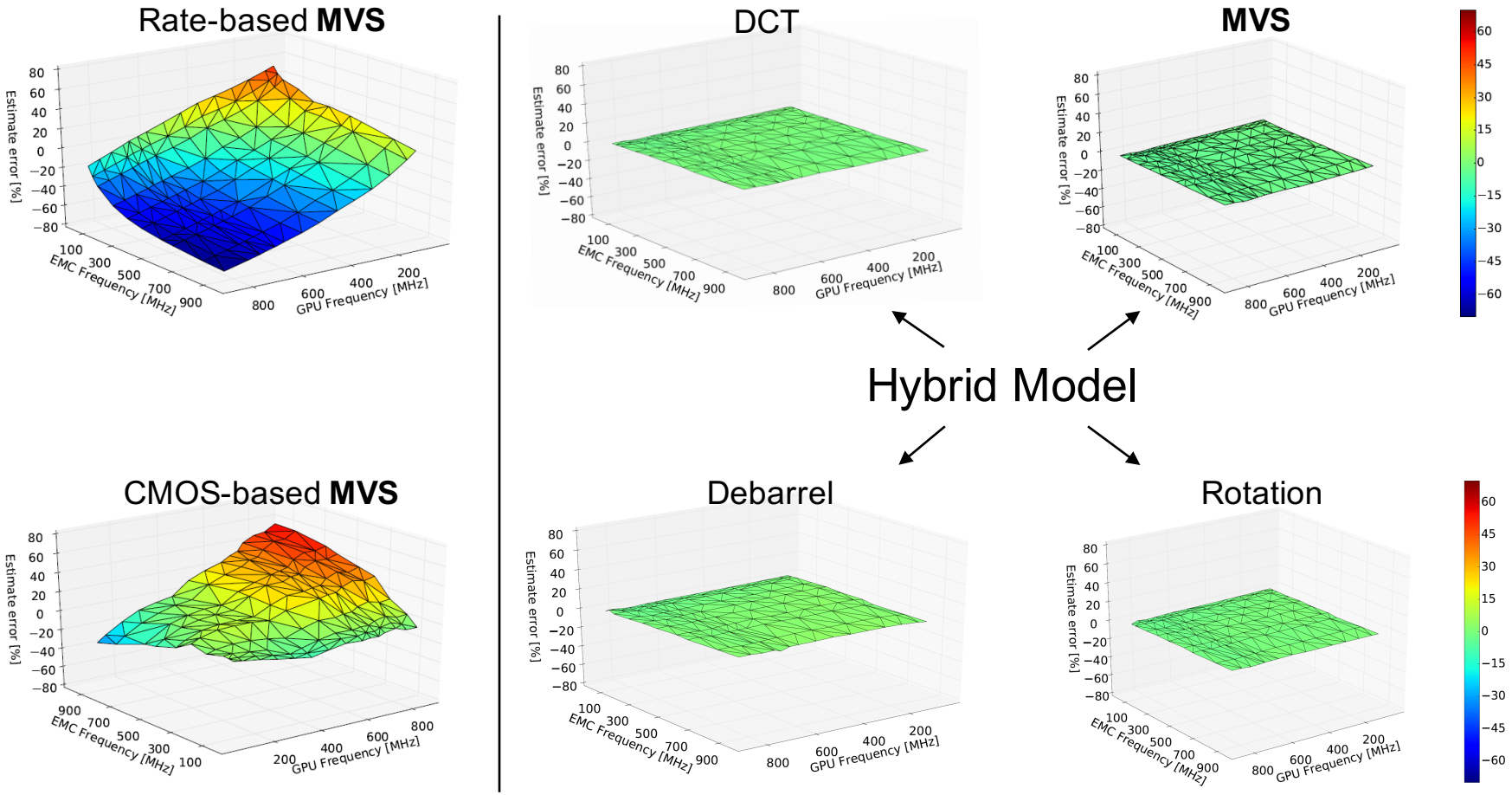
Finding the Right Answer

- For regression to work, a **training data set** must be generated
  - ..and the training software must be **carefully designed** to ensure that the predictors **vary** enough compared to one another

- The following is the benchmark suite for the GPU
  - Stress a few number of architectural units first
  - **All** benchmarks run over **all** possible GPU and memory frequencies

Benchmark	Description	Components / instructions under explicit stress										
		CPU	RAM (CPU)	GPU	RAM (GPU)	L2	L1	INT	F32	F64	Conv.	Misc.
Idle CPU	GPU off, CPU in idle state.	✓										
CPU-workload	GPU off, CPU processing.	✓	✓									
Idle GPU	GPU on and idle, CPU in idle state.	✓		✓								
L2 Read	Stresses L2 cache reads only.	✓		✓		✓						
L1 Read	Stresses L1 cache reads.	✓	✓	✓			✓					
L1 Write	Stresses L1 cache writes.	✓	✓	✓			✓					
RAM	Stresses RAM activity (GPU EMC).	✓		✓	✓							
Integer	Stresses integer arithmetic unit.	✓		✓		✓		✓				
Float32	Stresses floating point unit.	✓		✓		✓		✓	✓			
Float64	Stresses floating point unit.	✓		✓		✓		✓		✓		
Control	Stresses conversion instructions.	✓		✓		✓		✓			✓	
Misc	Stresses miscellaneous instructions.	✓		✓		✓		✓				✓

Model Precision

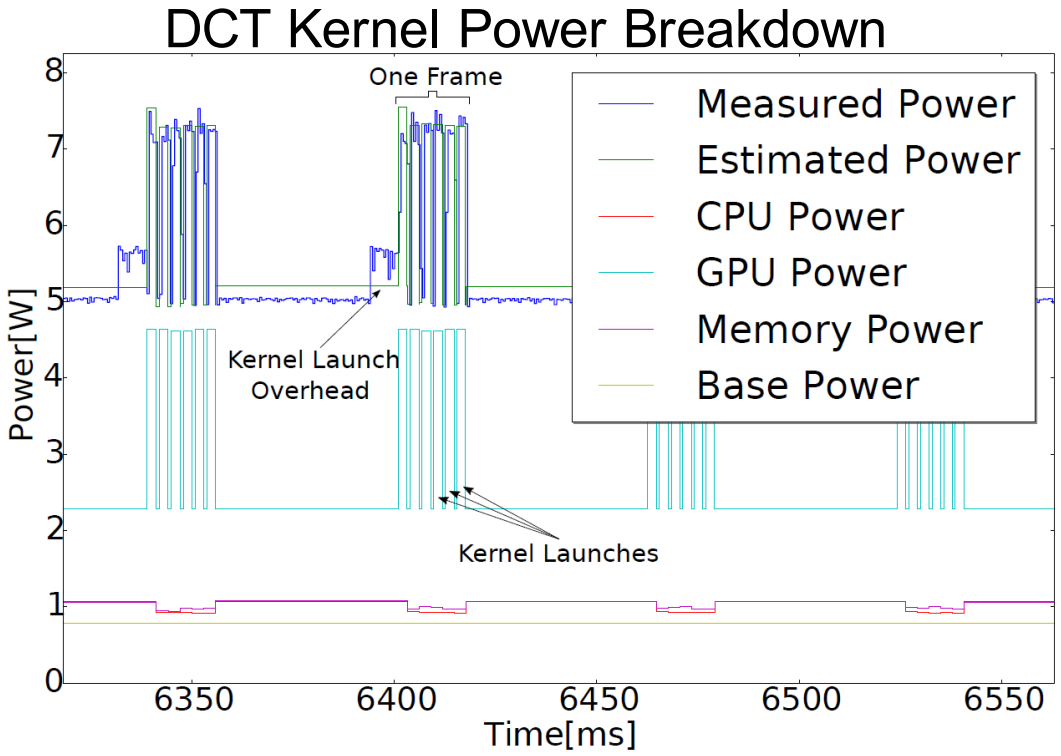


Conclusion

- We have introduced a power modelling methodology which captures power usage with very high precision
  - Considers voltages and detailed hardware utilisation on separate power rails
  - Can be used to analyse power usage of software
- Can be used to optimise power of different multimedia workloads (**10-40 % increased battery time**)
- **A word of caution**
  - Power and energy in modern computing systems are complex topics
  - At least use models that are extensively verified and shown to yield good accuracy across a wide range of workloads

Backup Slides

Power Prediction Over Time



- Our model is able to predict power usage of both CPU and GPU execution with very high accuracy

GPU Model Coefficients

Leakage

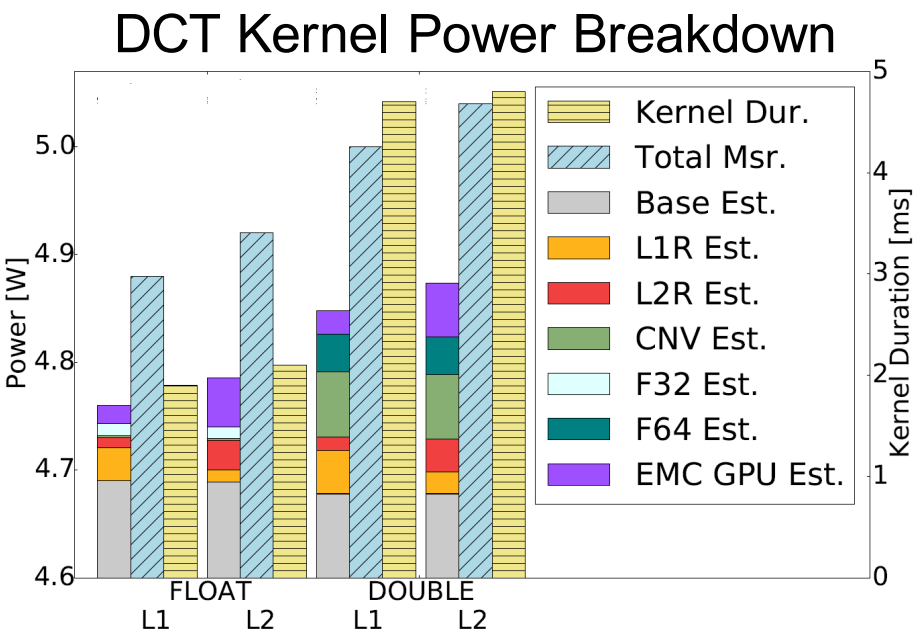
Rail	Number	Predictor	Description	Coefficient	Value
GPU	0	V_{gpu}	GPU voltage	$I_{gpu,leak}$	$0.27A$
	1	$\rho_{gpu,clock}$	Total clock cycles per second	$C_{gpu,clock}$	$2.10\frac{nC}{V}$
	2	$\rho_{gpu,L2R}$	L2 cache 32B reads per second	$C_{gpu,L2R}$	$10.79\frac{nC}{V}$
	3	$\rho_{gpu,L1R}$	L1 cache 4B reads per second	$C_{gpu,L1R}$	$8.90\frac{nC}{V}$
	4	$\rho_{gpu,L1W}$	L1 cache 4B writes per second	$C_{gpu,L1W}$	$8.43\frac{nC}{V}$
	5	$\rho_{gpu,INT}$	Integer instructions per second	$C_{gpu,INT}$	$41.11\frac{pC}{V}$
	6	$\rho_{gpu,F32}$	Float (32-bit) instructions per second	$C_{gpu,F32}$	$38.15\frac{pC}{V}$
	7	$\rho_{gpu,F64}$	Float (64-bit) instructions per second	$C_{gpu,F64}$	$115.33\frac{pC}{V}$
	8	$\rho_{gpu,CNV}$	Conversion instructions per second	$C_{gpu,CNV}$	$72.42\frac{pC}{V}$
	9	$\rho_{gpu,MSC}$	Miscellaneous instructions per second	$C_{gpu,MSC}$	$28.36\frac{pC}{V}$
Memory	0	$\rho_{mem,clock}$	Total clock cycles per second	$C_{mem,clock}$	$258.66\frac{pC}{V}$
	1	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	$-0.03W$
	2	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	$0.05W$
	3	$\rho_{mem,CPU}$	CPU busy memory cycles per second	$C_{mem,cpu}$	$2.25\frac{nC}{V}$
	4	$\rho_{mem,OTH}$	Other (GPU) busy memory cycles per second	$C_{mem,oth}$	$2.17\frac{nC}{V}$
Core	0	V_{cpu}	CPU voltage	$I_{cpu,leak}$	$0.79A$
	1	$\rho_{cpu,cpi}$	CPU instructions per cycle	$C_{cpu,cpi}$	$3.72\frac{mC}{V_s}$
	2	$\rho_{cpu,acl}$	CPU active cycles per second	$C_{cpu,acl}$	$166.62\frac{pC}{V}$
Other	P_{base}		Base power	-	$0.78W$

- The «memory offsets» compensate for variation in power across memory frequencies (ref slide 9)
  - Supposed to be negative!

Positive estimates ☺ ☺ ☺

Power Optimisation

- Caching in L1 over L2 saves power due to reduced external memory accesses (EMC GPU)
  - Because L1 is not cache coherent
- Using shorter datatypes (float32 over float64) also conserves energy
  - Less direct computation and less conversion instructions in our example
  - **Pascal and mixed precition (16-bit float)?**
- In our experience, optimising for power is equivalent to optimising for performance
  - Which is good news ☺



Understanding Hardware Activity: GPU Memory (3)

- Shared memory complicates the picture..
  - Memory is often broadcasted to all threads of a warp
  - In this case, the **l1_shared_load_transactions** HPC counts all of the accesses, but in **hardware** there was only **a single access**
    - Same for writes
  - Impossible to fix, but it is possible to approximate the **actual accesses**:
    - **l1_shr_{load/store} = l1_shared_{load/store}_transactions * shared_efficiency**
    - Although it is not a really good solution.

HPC Name	Description
l1_shared_{store/load}_transactions	Shared memory
shared_efficiency	