GreenLA: Green Linear Algebra Software for GPU-Accelerated Heterogeneous Computing

Jieyang Chen[*§], Li Tan[*†§], Panruo Wu[*], Dingwen Tao[*], Hongbo Li[*], Xin Liang[*], Sihuan Li[*], Rong Ge[‡], Laxmi Bhuyan[*], and Zizhong Chen[*]

[*]University of California, Riverside
[‡]Clemson University
[†]Ultra Scale Systems Research Center, Los Alamos National Laboratory
[§] Authors contributed equally

Outline

- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

Outline

- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

Introduction

- Large-scale HPC system
  - High power consumption

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660

Energy Consumption of Top5 Supercomputers(June, 2016 - TOP500 List)

Summary of Our Contributions

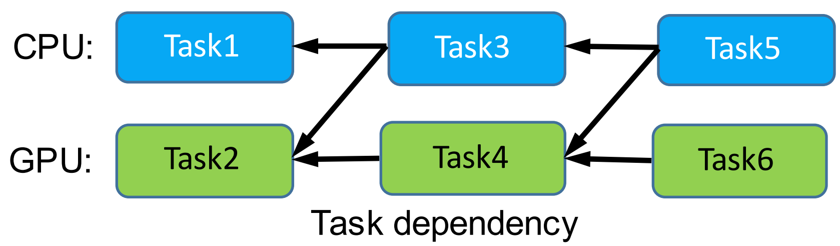
- Most of the existing energy saving approaches focus on the system/OS level or architecture level, without considering application characteristic.
- In this work, we proposed an energy saving technique that combine both ***architecture*** and ***specific application*** knowledge to achieve better energy efficiency.
- We designed **GreenLA**: Green Linear Algebra Software for GPU-Accelerated Heterogeneous Computing.

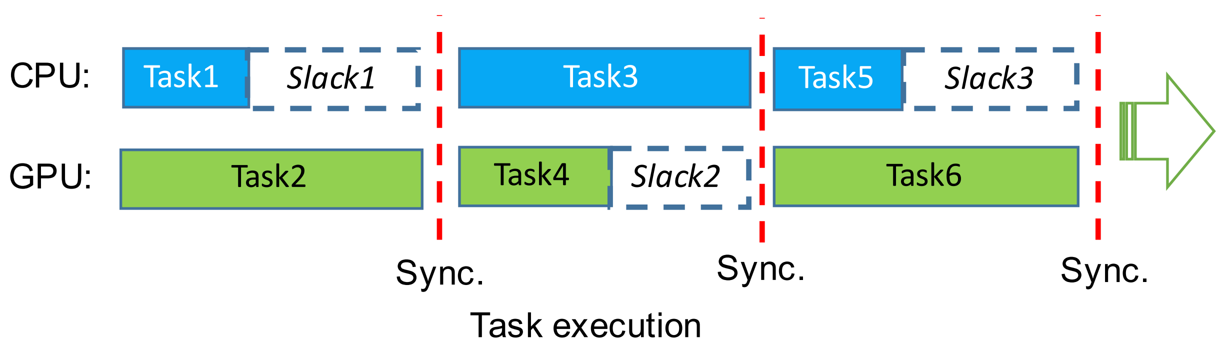
DVFS for energy saving

- Dynamic Voltage and Frequency Scaling (DVFS)
  - By adjusting the processer's voltage and frequency dynamically, energy consumption can be reduced.
  - Theoretically: $Power \propto f^{2.4}$ and $ExecTime \propto f$.
  - E.g.: $\downarrow freq\ by\ 50\%$→ $\downarrow power\ by\ 81\%$→$Save\ 62\%\ Energy.$
  - It is proved in many previous works that applying DVFS to slacks in application can save considerable energy without significantly impact performance.

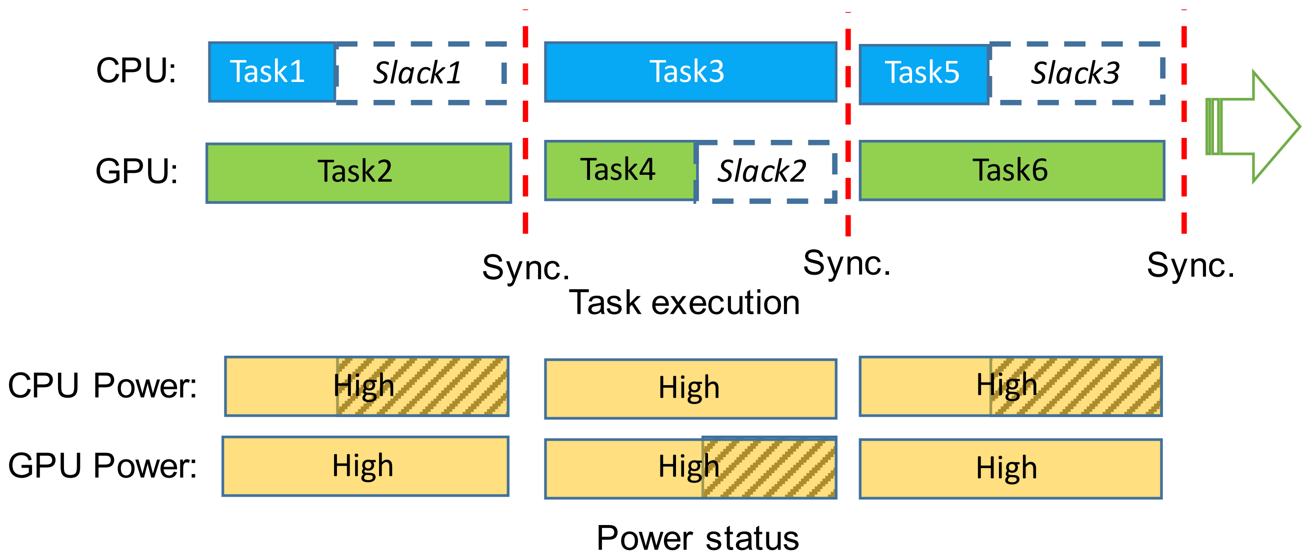
Slacks

- Slacks exist in many applications in parallel computing systems.
- Slacks can be caused by: dependency, communications, I/O, etc.



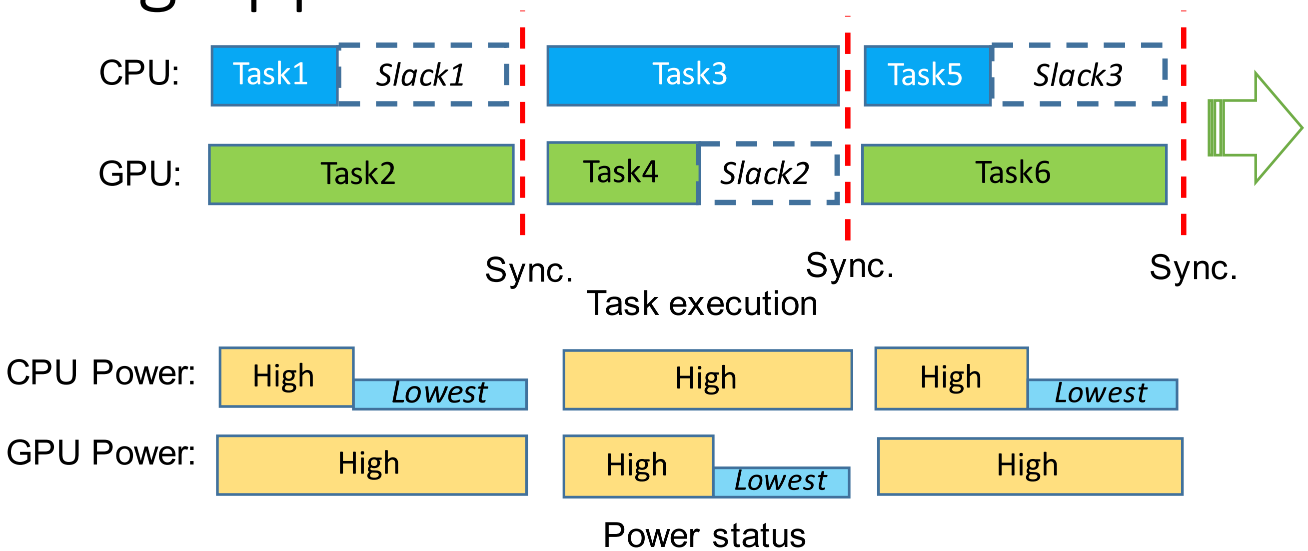


Motivation of Slack-based Energy Saving



- If we keep the same frequency and voltage, the energy in shadow areas would be wasted.
- Many energy saving techniques have been developed utilizing the slacks
  - Race-to-Halt (R2H);
  - Slack Reclamation (SR).

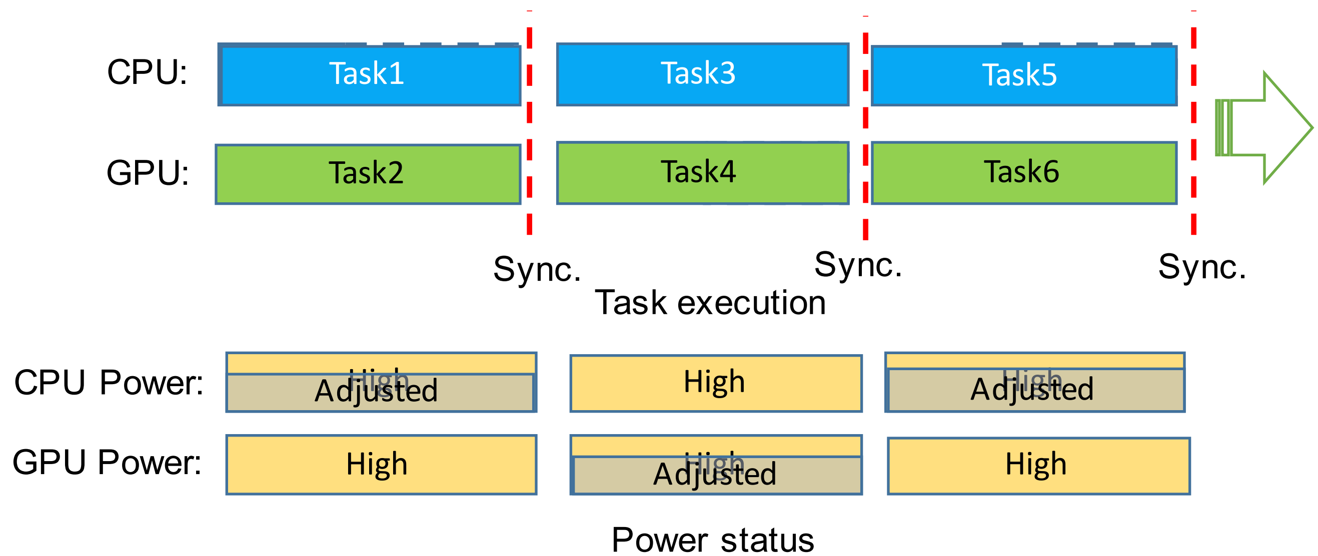
Existing Approach: Race-to-Halt



- Race: process tasks at the processors highest power state.
- Halt: idling at the processors to lowest power state.
- Simple, no application knowledge required.
- But it is proved*, running tasks at highest power state is ***not*** very energy efficient.

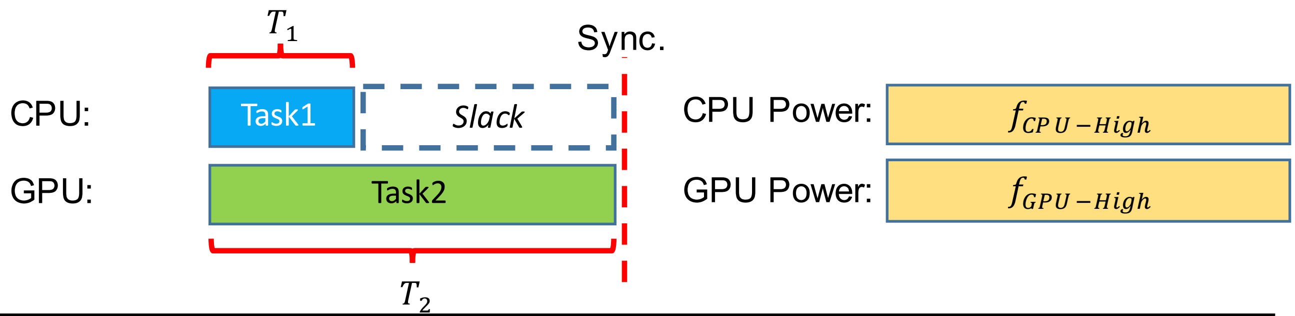
*Rountree, Barry, et al. "Adagio: making DVS practical for complex HPC applications." *Proceedings of the 23rd international conference on Supercomputing.* ACM, 2009.

Existing Approach: Slack Reclamation



- Identify critical path of application
- Adjust frequency when running tasks on non-critical path → reclaim slacks
- More energy efficient than R2H, requires prediction on slack to adjust freq. in advance.

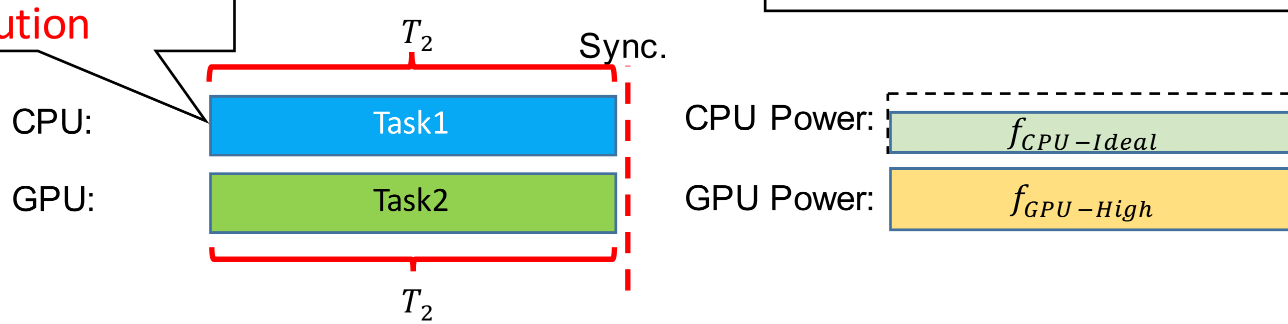
Existing Approach: Slack Reclamation



- Assume freq. is linearly proportional to execution time
- $f_{CPU-Ideal} = \dfrac{f_{CPU-High} \times T_1}{T_2}$

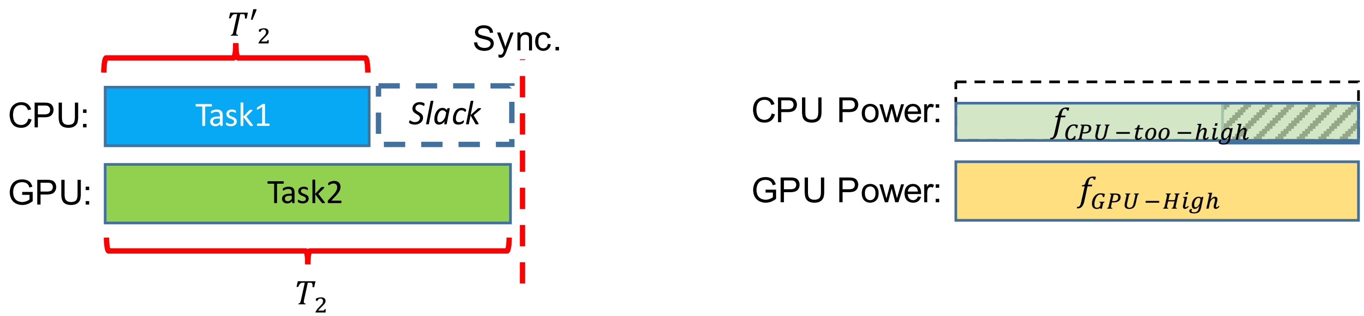
Requies prediction on execution time(i.e.: T_1, T_2)

Adjust freq. before execution

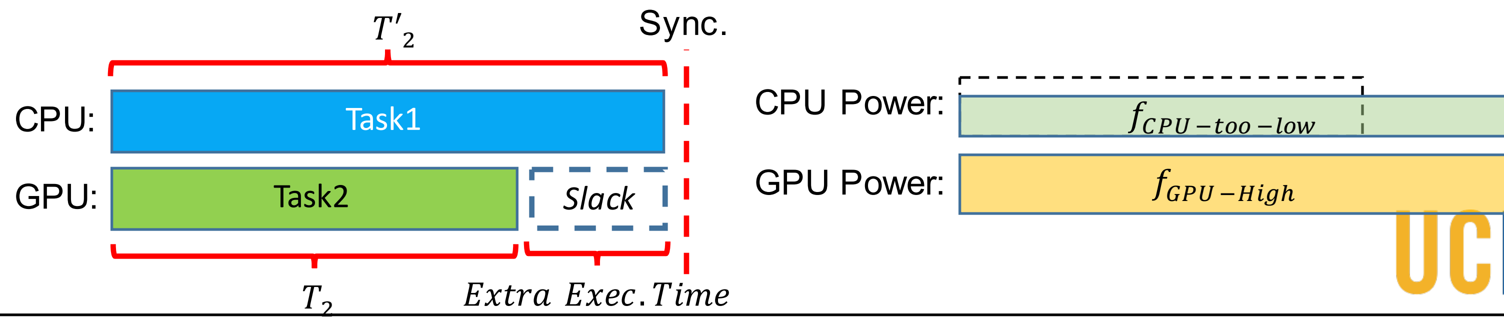


Challenges of Slack Reclamation

- It requires ***accurate*** prediction on execution time.
- Inaccurate predicted execution time → inaccurate target frequency
  - Target frequency > ideal frequency → Still waste energy



  - Target frequency < ideal frequency → Impact performance



UCR

Existing Slack Prediction

- It requires accurate prediction on execution time.
- Existing approach: statistic learning based slack prediction
  - It predicts future execution time based on history.
  - It assumes constant slack length or based on history data.
- However, its prediction ability is ***limited***.
  - It can only capture **limited** or **inaccurate** ***application characteristic*** from history data.
  - When the target application is complicated or irregular patterned, the history data can be less useful, the prediction results can be **inaccurate**.
  - **Inaccurate execution time prediction → inaccurate target frequency → less energy saving or even more energy cost and worse performance.**
  - **It requires to use first 10%-20% iteration as training set → extra energy cost and wastes valuable energy saving opportunities.**
- So, in order to achieve better prediction result, we need to know ***more*** about the target application.

Outline

- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

Linear Algebra Operations

- To achieve high accurate slack prediction, we need to know:
  - **Algorithmic knowledge:** the application characteristic in algorithm level.
  - **Architecture characteristic**: the application characteristic running on given hardware platforms.
- In most linear algebra operations, the algorithms inside them usually follows iterative fashion, which can help us capturing those information.
  - Most linear algebra libraries are open-sourced, we are able to capture the **algorithmic knowledge** inside each linear algebra operation.
  - The iterative execution fashion makes the computation work of each task on different iterations similar. With minimum profiling, we can easily approximate the **architecture characteristic.**
- So, we proposed **Algorithmic Slack Prediction** model for linear algebra operations.

Our Propose: Algorithmic Slack Prediction

- Leverage both ***algorithmic*** knowledge and ***architecture*** characteristic to achieve more accurate slack prediction.
- ***Algorithmic*** knowledge
  - We exam the target application to identify: (1) ***potential slacks*** (2)***overall tasks execution characteristic*** (We assume source code is assessable.)
- ***Architecture*** characteristic
  - We do profiling on the first slack-related tasks on target hardware platform to capture the ***computing efficiency***. (We assume this efficiency for each tasks in each iteration stays constant.)
- Benefits over statistic learning based prediction
  - Our model is built directly from application → more accurate result.
  - One-time profiling is good enough to capture the architecture characteristic → much lower prediction model training cost.

Algorithmic Slack Prediction Model

- Offline application inspection → capture ***algorithmic*** knowledge
  - **Find potential slacks:** Tasks scheduled concurrently with same synchronization point (e.g.: Slacks may cause by $Task_A$ and $Task_B$)
  - **Derive task execution characteristic:** Identify the relationship of execution time between two neighbor iterations. (e.g.: $\boldsymbol{R_A}(i) = \frac{T_A^{(i)}}{T_A^{(i-1)}}$, $\boldsymbol{R_B}(i) = \frac{T_B^{(i)}}{T_B^{(i-1)}}$)
- Online prediction
  1. $T_A^{(0)}, T_B^{(0)} \leftarrow$ profiling →capture ***architecture*** characteristic
  2. Iterate from 1 → n
  3. $T_A^{(i)} \leftarrow T_A^{(i-1)} * \mathbf{R_A}(i)$ → utilizing ***algorithmic*** knowledge
  4. $T_B^{(i)} \leftarrow T_B^{(i-1)} * \mathbf{R_B}(i)$ → utilizing ***algorithmic*** knowledge
  5. Ideal freq. ←**Calculate Ideal Frequency**($T_A^{(i)}, T_B^{(i)}$)
  6. **Slack Reclamation**(Ideal freq.)
  7. … process tasks …

UCR

Outline

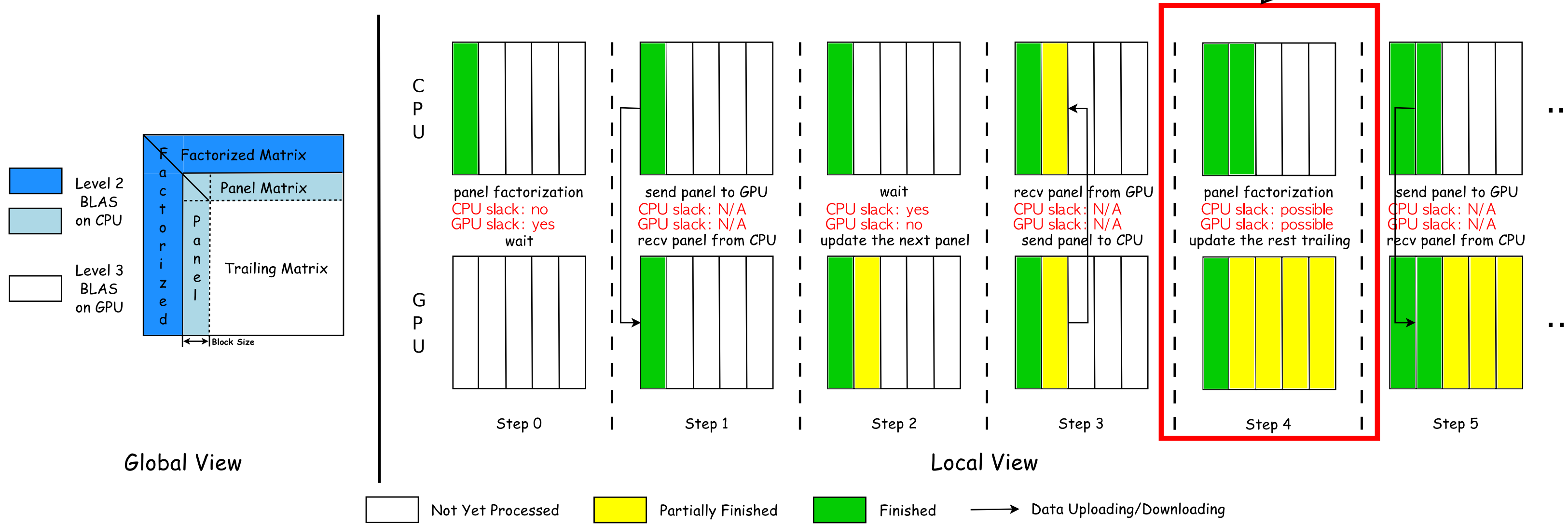
- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

GreenLA Library

- Build based on MAGMA
  - State-of-the-art highly optimized linear algebra library on heterogeneous system with GPUs.
  - It assign tasks statically to CPUs and GPU to achieve high performance than traditional linear algebra library.
  - We identified that potential slacks exist in MAGMA's implementation. So, we implemented our algorithmic slacks prediction energy saving to MAGMA library.
- As the initial stage of this project, we start with three core linear algebra functions: **LU**, **Cholesky** and **QR** factorization.

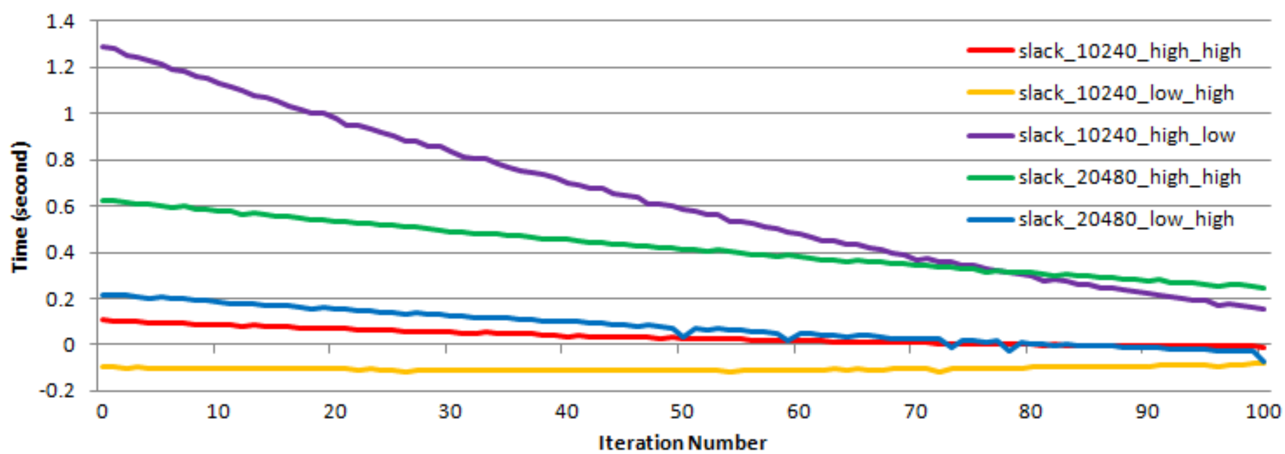
Application on LU factorization

- Capture ***algorithmic knowledge***
  - Identify potential slacks



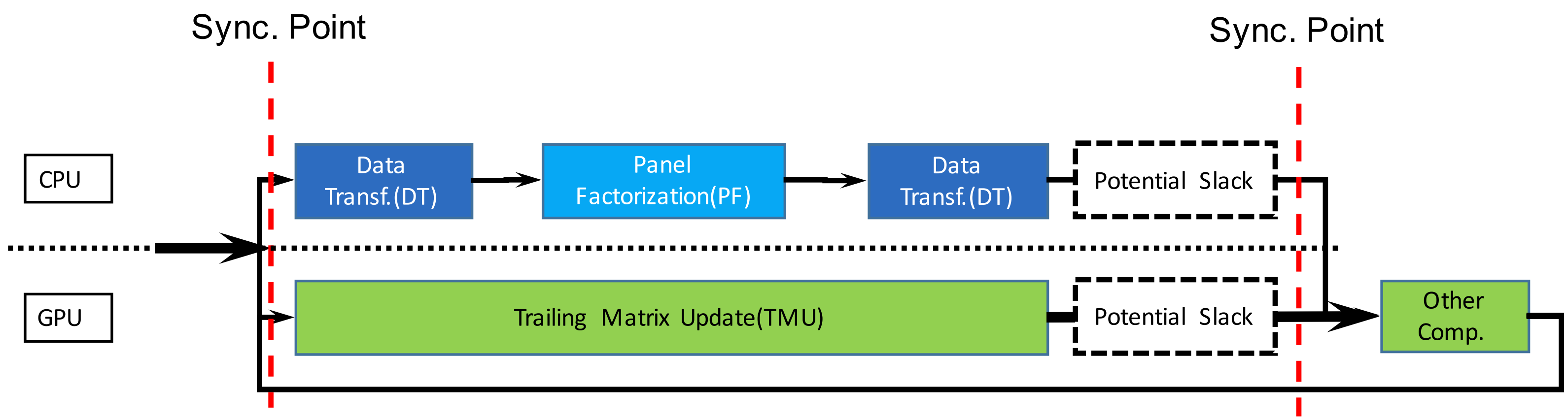
One iteration of LU factorization

Example: Slacks in LU factorization



Slack time of the first 100 iteration of LU factorization in MAGMA
Slack time = Trailing Matrix Update(on GPU) - Panel Factorization(on CPU)

Application on LU factorization



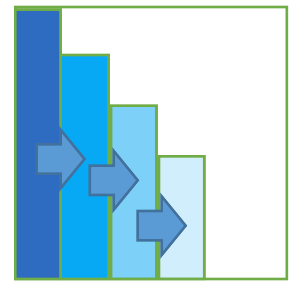
- $T_{Slack}^{(i)} \approx T_{PF}^{(i)} + T_{DT.}^{(i)} - T_{TMU}^{(i)}$
- $Sign\left(T_{Slack}^{(i)}\right)$→Which side is the slack
- $\left|T_{Slack}^{(i)}\right|$→Length of the slack

Application on LU factorization

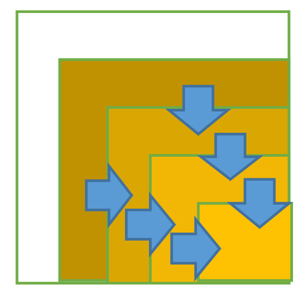
- Capture ***algorithmic knowledge***
  - **Derive task execution characteristic**
  - We determine the execution relation between neighbor iterations as follow:

    - $\frac{T_{PF}^{(i)}}{T_{PF}^{(i-1)}} = \frac{Flop_{PF}^{(i)}}{Flop_{PF}^{(i-1)}} = 1 - \frac{nb}{N-i*nb} = R_{PF}(i)$
    - $\frac{T_{TMU}^{(i)}}{T_{TMU}^{(i-1)}} = \frac{Flop_{TMU}^{(i)}}{Flop_{TMU}^{(i-1)}} = \left(1 - \frac{nb}{N-i*nb}\right)^2 = R_{TMU}(i)$
    - $\frac{T_{DT}^{(i)}}{T_{DT}^{(i-1)}} = \frac{Size_{DT}^{(i)}}{Size_{DT}^{(i-1)}} = 1 - \frac{1}{N-i} = R_{DT}(i)$

- $T_{PF}^{(0)}, T_{DT.}^{(0)}, T_{TMU}^{(0)}$ can be determined by offline profiling



Panel Factorization



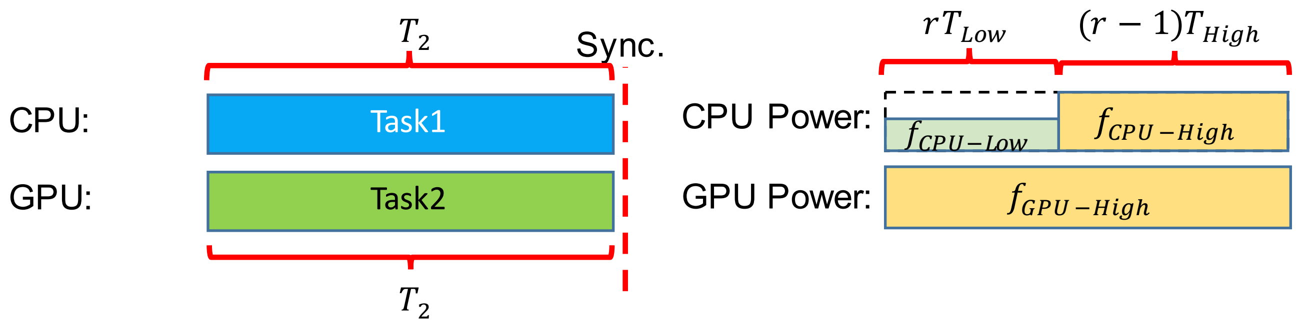
Trailing Matrix Update

Application on LU factorization

- Online prediction
  1. $T_{PF}^{(0)}, T_{DT.}^{(0)}, T_{TMU}^{(0)} \leftarrow \text{profiling}$ → ***architecture*** characteristic
  2. Iterate from 1 → n
  3. $\mathrm{T_{PF}^{(i)}} \leftarrow \mathrm{T_{PF}^{(i-1)}} * \mathbf{R_{PF}}(\mathrm{i})$
  4. $\mathrm{T_{DT}^{(i)}} \leftarrow \mathrm{T_{DT}^{(i-1)}} * \mathbf{R_{DT}}(\mathrm{i})$
  5. $\mathrm{T_{TMU}^{(i)}} \leftarrow \mathrm{T_{TMU}^{(i-1)}} * \mathbf{R_{TMU}}(\mathrm{i})$
  6. Ideal freq. ←**Calculate Ideal Frequency**($\mathrm{T_{PF}^{(i)}}, \mathrm{T_{DT}^{(i)}}, \mathrm{T_{TMU}^{(i)}}$)
  7. if Slack on CPU → SetGPUFreq.(Ideal freq.)
  8. if Slack on GPU → SetCPUFreq.(Ideal freq.)
  9. Process task: PF, DT, and TMU (Slack-related tasks)
  10. Restore power state after slack-related computations.

Optimization 1

- If target freq. no available:
  - We use Frequency Split
  - Combine two nearby frequencies to approximate ideal frequency.
  - Determine freq. split ratio *r:*
  - $T_2 = rT_{Low} + (r-1)T_{High}$ for *r*
  - T_{Low} and T_{High} can be determined by our slack prediction algorithm



Optimization 2

- Reduce DVFS Overhead
  - Frequent power state adjustment may brings much overhead
  - We use Relax Factor to adjust necessity of power state adjustment.
  - Algorithm with Relaxed Slack Reclamation:

1. If f_{ideal} not available
2. $\quad r \leftarrow$ determine freq. split ratio
3. $\quad$ if $r < RlxFctr$
4. $\quad\quad$ Avoid freq. split, use original freq.
5. $\quad$ else
6. $\quad\quad$ Perform freq. split

Outline

- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

Experimental Environment

- Heterogeneous System: IVY

Component	CPU	GPU
Processor	2*10-core Intel Xeon E5-2670	2496-core NVIDIA Kepler K20c
Peak Perf.	0.4 TFLOPS	1.17 TFLOPS
Frequency Scaling Capability	Core Freq. (GHz)	Core Freq. (MHz)
	1.2-2.5(↑ by 0.1)	324, 614, 640, 666, 705,758
Memory	64 GB RAM	5 GB RAM
Power Meter	Power Pack	Nvidia-smi tool
Freq. Adjustment Method	CPU Freq. Register Files	NVIDIA GPU Freq. Setting API

Experiment Settings

- We tested and compared:
  - Original MAGMA implementation
  - OS Level Race-to-Halt
  - Library Level Race-to-Halt
  - OS Level Statistic Learning Based DVFS
  - OS Level Statistic Learning Based DVFS(Relaxed)
  - Algorithmic Prediction Based DVFS
  - Algorithmic Prediction Based DVFS(Relaxed)
- All versions are build on MAGMA 1.6.1
- Cholesky, LU, QR factorization on 4 sizes:
  - $5120, 10240, 15360, and\ 20480$

Experiment

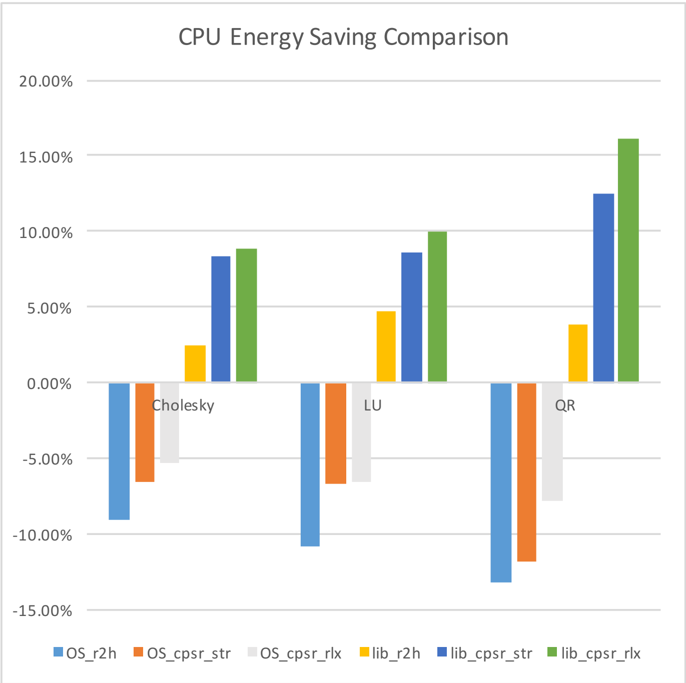
- Slack Prediction Error Rate:
- Error Rate = Average($\frac{|Pred. Slack\ Time\ - Ture\ Slack\ \ Time|}{True\ Slack\ Time}$)

Benchmarks	Statistic Learning Prediction		Algorithmic Prediction
	Base Iter.(First 10%)	Base Iter. (First 20%)	
Cholesky	10.51%	6.62%	0.96%
LU	9.95%	5.45%	0.16%
QR	11.29%	5.77%	0.52%

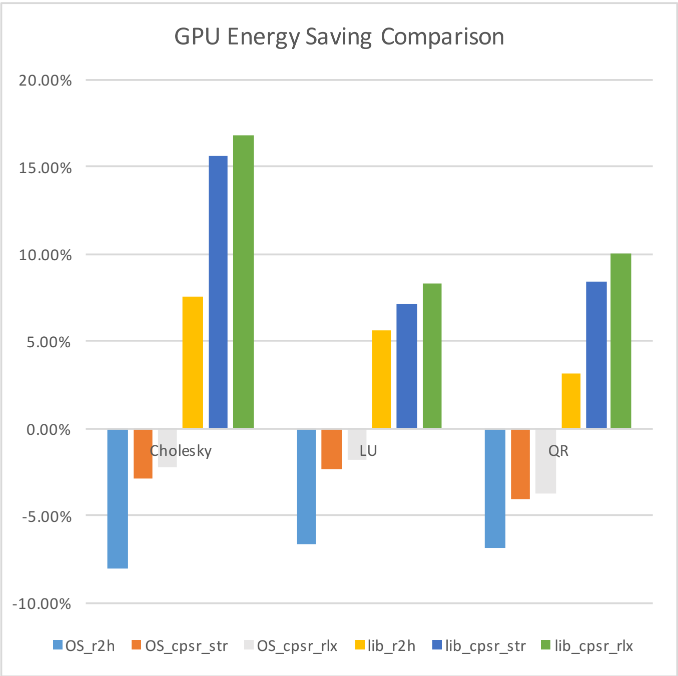
Slack prediction accuracy comparison on input matrix size 20480*20480

- Our Algorithmic prediction has much lower prediction error rate.

CPU & GPU Energy Saving

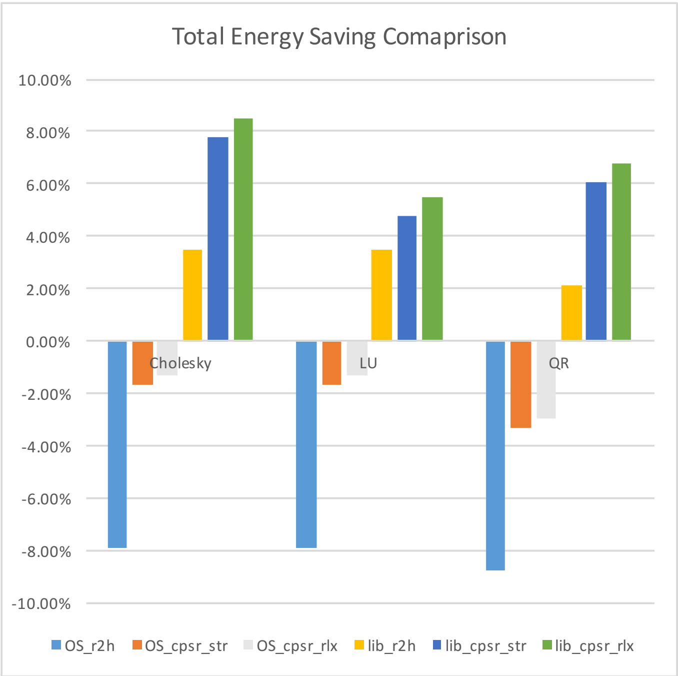


CPU energy saving on input matrix size: 20480*20480

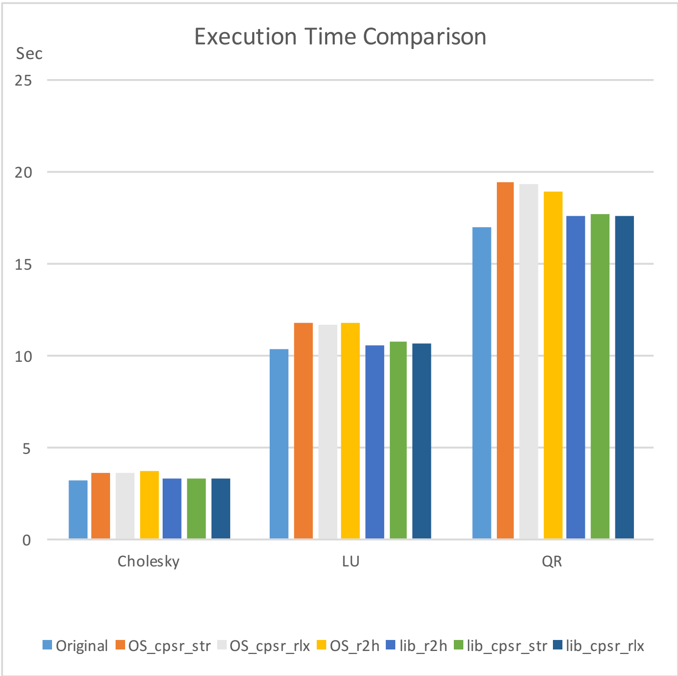


GPU energy saving on input matrix size: 20480*20480

Overall Energy Saving & Performance



Total energy saving on input matrix size: 20480*20480



Overall execution time with input matrix size: 20480*20480

Outline

- Introduction and our research motivation
- Our new algorithmic slack prediction
- GreenLA Library
- Experimental Evaluation
- Conclusion

Conclusion

- Slack-based energy saving approach requires accurate slack prediction to achieve high energy saving.
- Existing approaches cannot accurately predicts slacks.
- So, we proposed a new algorithmic slack prediction approach considering both algorithmic knowledge and architecture characteristic.
- Our experiments show that our approach can save 2x-3x more energy than existing approaches.

- Thanks everyone for attending!