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

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 processor’s voltage and frequency dynamically, energy consumption can be reduced.
  • Theoretically: $Power \propto f^{2.4}$ and $ExecTime \propto f$.
  • E.g.: ↓ freq by 50% $\rightarrow$ ↓ power by 81% $\rightarrow$ 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.

---

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

\[ f_{CPU-Ideal} = \frac{f_{CPU-High} \times T_1}{T_2} \]

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 \(\rightarrow\) inaccurate target frequency
  • Target frequency > ideal frequency \(\rightarrow\) Still waste energy

\[
\begin{align*}
T'_{2} & \quad \text{Sync.} \\
\text{CPU: } & \quad \text{Task1} \quad \text{Slack} \\
\text{GPU: } & \quad \text{Task2}
\end{align*}
\]

• Target frequency < ideal frequency \(\rightarrow\) Impact performance

\[
\begin{align*}
T'_{2} & \quad \text{Sync.} \\
\text{CPU: } & \quad \text{Task1} \quad \text{Slack} \\
\text{GPU: } & \quad \text{Task2}
\end{align*}
\]
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 $\rightarrow$ inaccurate target frequency $\rightarrow$ less energy saving or even more energy cost and worse performance.
  • It requires to use first 10%-20% iteration as training set $\rightarrow$ 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.: \( R_A(i) = \frac{T_A^{(i)}}{T_A^{(i-1)}}, R_B(i) = \frac{T_B^{(i)}}{T_B^{(i-1)}} \))

• Online prediction
  1. \( T_A^{(0)} , T_B^{(0)} \leftarrow \text{profiling } \rightarrow \text{capture architecture characteristic} \\
  2. Iterate from \( 1 \rightarrow n \)
  3. \( T_A^{(i)} \leftarrow T_A^{(i-1)} \ast R_A(i) \rightarrow \text{utilizing algorithmic knowledge} \\
  4. \( T_B^{(i)} \leftarrow T_B^{(i-1)} \ast R_B(i) \rightarrow \text{utilizing algorithmic knowledge} \\
  5. Ideal freq. \leftarrow \text{Calculate Ideal Frequency}( T_A^{(i)} , T_B^{(i)} ) \\
  6. Slack Reclamation(Ideal freq.) \\
  7. ... process tasks ...
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 assigns 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

Potential Slacks
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

- $T_{Slack}^{(i)} \approx T_{PF}^{(i)} + T_{DT}^{(i)} - T_{TMU}^{(i)}$
- $\text{Sign} \left( T_{Slack}^{(i)} \right) \rightarrow \text{Which side is the slack}$
- $|T_{Slack}^{(i)}| \rightarrow \text{Length of the slack}$
Application on LU factorization

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

\[
\frac{T_{PF}^{(i)}}{T_{PF}^{(i-1)}} = \frac{\text{Flop}_{PF}^{(i)}}{\text{Flop}_{PF}^{(i-1)}} = 1 - \frac{nb}{N-i*nb} = R_{PF}(i)
\]

\[
\frac{T_{TMU}^{(i)}}{T_{TMU}^{(i-1)}} = \frac{\text{Flop}_{TMU}^{(i)}}{\text{Flop}_{TMU}^{(i-1)}} = \left(1 - \frac{nb}{N-i*nb}\right)^2 = R_{TMU}(i)
\]

\[
\frac{T_{DT}^{(i)}}{T_{DT}^{(i-1)}} = \frac{\text{Size}_{DT}^{(i)}}{\text{Size}_{DT}^{(i-1)}} = 1 - \frac{1}{N-i} = R_{DT}(i)
\]

- \(T_{PF}^{(0)}, T_{DT}^{(0)}, T_{TMU}^{(0)}\) can be determined by offline profiling
Application on LU factorization

• Online prediction
  1. $T_{PF}^{(0)}, T_{DT}^{(0)}, T_{TMU}^{(0)} \leftarrow$ profiling $\rightarrow$ architecture characteristic
  2. Iterate from 1 $\rightarrow$ n
  3. $T_{PF}^{(i)} \leftarrow T_{PF}^{(i-1)} \ast R_{PF}(i)$
  4. $T_{DT}^{(i)} \leftarrow T_{DT}^{(i-1)} \ast R_{DT}(i)$
  5. $T_{TMU}^{(i)} \leftarrow T_{TMU}^{(i-1)} \ast R_{TMU}(i)$
  6. Ideal freq. $\leftarrow$ **Calculate Ideal Frequency**($T_{PF}^{(i)}, T_{DT}^{(i)}, T_{TMU}^{(i)}$)
  7. if Slack on CPU $\rightarrow$ SetGPUFreq(Ideal freq.)
  8. if Slack on GPU $\rightarrow$ 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

\[
T_{2} = rT_{Low} + (r - 1)T_{High}
\]
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. \( r \leftarrow \) determine freq. split ratio
    3. if \( r < RlxFctr \)
    4. Avoid freq. split, use original freq.
    5. else
    6. Perform freq. split
Outline

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

- Heterogeneous System: IVY

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

\[ \text{Error Rate} = \text{Average}(\frac{|\text{Pred. Slack Time} - \text{True Slack Time}|}{\text{True Slack Time}}) \]

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Statistic Learning Prediction</th>
<th>Algorithmic Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base Iter.(First 10%)</td>
<td>Base Iter. (First 20%)</td>
</tr>
<tr>
<td>Cholesky</td>
<td>10.51%</td>
<td>6.62%</td>
</tr>
<tr>
<td>LU</td>
<td>9.95%</td>
<td>5.45%</td>
</tr>
<tr>
<td>QR</td>
<td>11.29%</td>
<td>5.77%</td>
</tr>
</tbody>
</table>

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 Comparison

- Cholesky
- LU
- QR

GPU Energy Saving Comparison

- OS_r2h
- OS_cpsr_str
- OS_cpsr_rlx
- lib_r2h
- lib_cpsr_str
- lib_cpsr_rlx

CPU energy saving on input matrix size: 20480*20480

GPU energy saving on input matrix size: 20480*20480
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!