Heterogeneous Computing with OpenCL: Strategies of Utilizing both a CPU and GPU

Hyoungseok Chu (hschu@nims.re.kr) and Hwansun Kim (iou78@nims.re.kr)

Division of Computational Sciences in Mathematics, National Institute for Mathematical Sciences, South Korea

Introduction

▶ Goal

- Investigate efficient parallel strategies under heterogeneous computing systems.
- Our research interests are of Linear Solvers especially for Krylov Subspace Methods and their BLAS. (axpy, dot, and spmv)

▶ Strategies

- Data parallel: Load Balancing
- Task parallel: Supplementary Acceleration

Target Platforms

▶ HAECHI (4 nodes)

- High-performance Applications for Extreme-scale Computing with Heterogeneous Infrastructure.
  - 2 CPUs (Xeon E5-2650) and 3 GPUs (HD 7950)
  - Optional: Xeon Phi (7120P), FPGA (Nallatech 395)

▶ Target Devices

- One CPU and GPU
- Node level parallelism

Target Applications

▶ Basic Linear Algebraic Subroutines (BLAS)

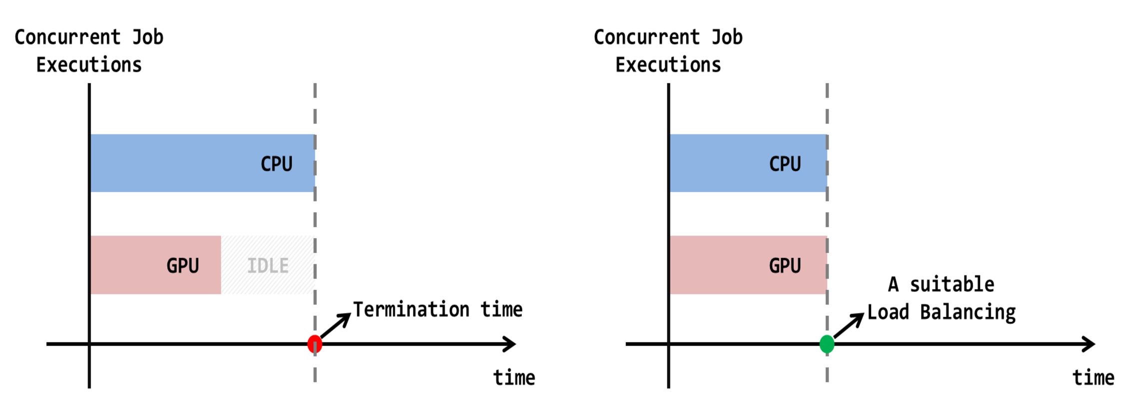
- Core algorithms for linear solvers
- axpy, dot, norm2, spmv

▶ Krylov Subspace Methods ($Ax = b$)

- Preconditioned approaches are considered
  - To guarantee robustness for general system
  - Include preconditioned CG and BiCG
  - Approximated inverse preconditioning

Data Parallel Strategy

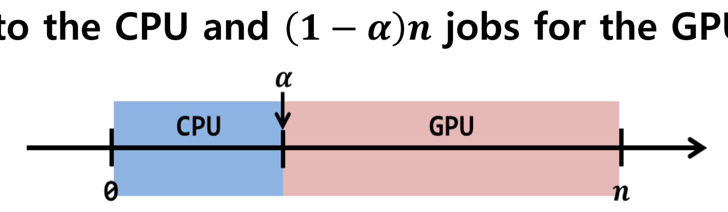
Motivation



Load Balancing

▶ Assume we have n numbers of data parallelizable jobs.

▶ We propose a job splitting ratio called α which divides workloads into αn and $(1-\alpha)n$ jobs so that assign αn jobs into the CPU and $(1-\alpha)n$ jobs for the GPU.



min-max Model for α

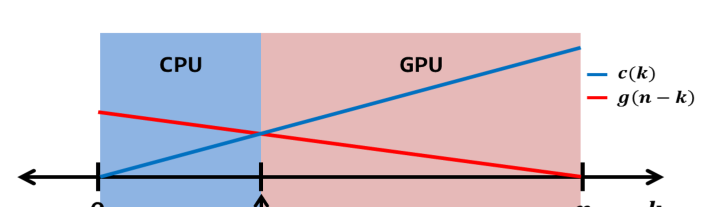
For given n, find a splitting point k^* which is the solution of the following min-max problem.

$$\mathcal{J}(k^*; n) = \min_{k\in[0,n]} \mathcal{J}(k; n)$$

where $\mathcal{J}(k;n) = \max(c(k), g(n-k))$

$c(k)$ denotes elapsed time for input k using CPU and $g(n-k)$ denotes elapsed time for input $n-k$ using GPU. The splitting ratio α with given n is defined as follows.

$$\alpha(n) = \frac{k^*}{n}.$$



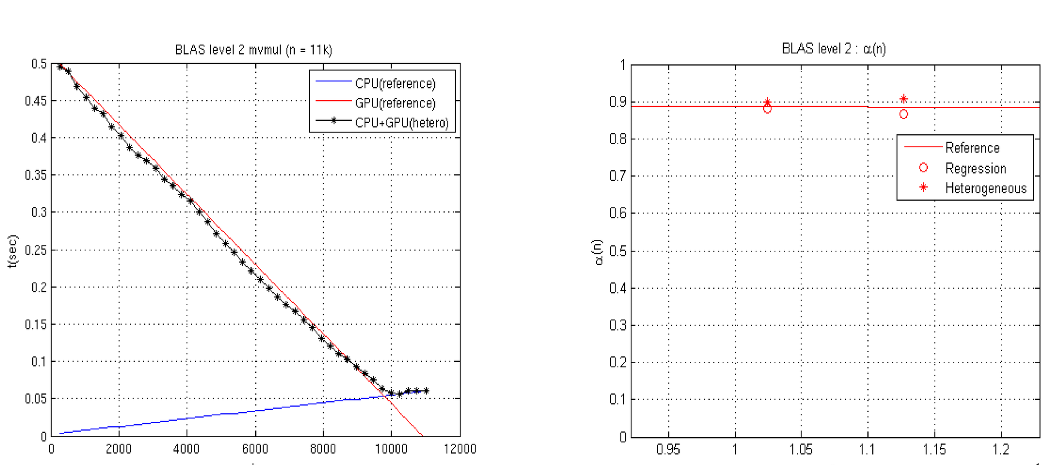
$$k^* = \min_{k\in[0,n]} \max(c(k), g(n-k))$$

Result (BLAS sgemv)

ENV: AMD OpenCL SDK 2.8, CPU: AMD FX 8120, GPU: HD7950

$$c(n) = \frac{4bytes\times(n+2)}{B_{CPU}}n + \frac{n^2}{F_{CPU}} + O_{CPU}$$

$$g(n) = \frac{4bytes\times(n+1)}{B_{PCI,W}}n + \frac{4bytes}{B_{PCI,R}}n + \frac{4bytes\times(n+2)}{B_{GPU}}n + \frac{n^2}{F_{GPU}} + O_{GPU}$$



values	Heterogeneous	Regression	Reference
Splitting ratio (α)	0.909	0.868	0.881
Execution time	0.05624 sec	0.06078 sec	0.06078 sec
Differences		0.00454 sec	0.00454 sec

Task Parallel Strategy

Motivation

▶ Iterative Krylov subspace methods (KSM) are composed of several routines of BLAS.

▶ Data-parallel strategy is seemed to be an easy approach, but has poor load balancing about 9:1 (CPU:GPU) which was caused by performance gaps of memory bandwidth.

Preconditioned KSM

▶ KSMs cover specific scopes of their usages with respect to characteristics of linear system.

▶ Preconditioning methods are better choices when it comes to consider about robustness of linear solvers.

▶ Of course, one should pay additional costs for precondi--tioning which may not fit well parallel implementation.

Well-known Two Preconditioners

$$Ax = b \qquad MAx = Mb$$

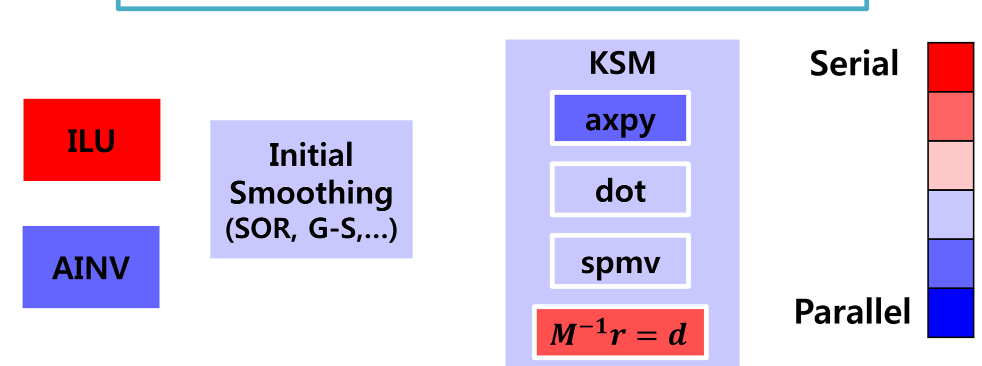
▶ A good preconditioner is to choose M to be closed to A^{-1}.

▶ The heaviest cost is solving additional system $M^{-1}r = d$.

▶ Incomplete Factorization (IC, ILU, ILUT, ...)

▶ Approximate Inverse (AINV, SAINV, ...)

Tasks for Preconditioned KSM



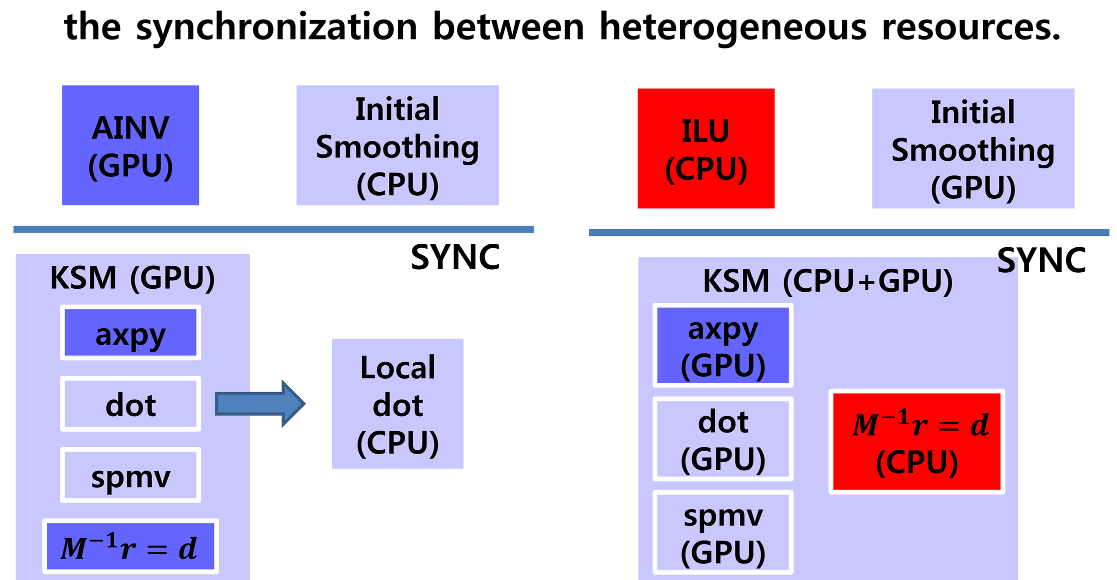
Supplementary Accelerations (SA)

▶ Assign each task based on its characteristics.

- CPU: good for serial algorithms, complex branches, frequent memory transfers.
- GPU: good for massively data-parallel algorithms.

▶ Additional jobs run concurrently with main algorithms so that accelerate the total throughput as well as convergence speed.

▶ OpenCL fits well to the implementation of SA by estimating performances of each task and by scheduling the synchronization between heterogeneous resources.



Implementation Issues

OpenCL Scheduling

▶ In the CPU case, using full processing elements (PEs) will affect the performance because of scheduling overheads. Concurrency may not be guaranteed.

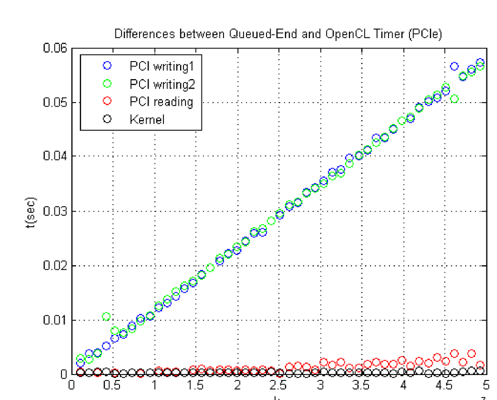
▶ `clCreateSubDevice()` creates sub-device as even or specific numbers of processing elements. Sub-division is considered to make one PE be managed for scheduler.

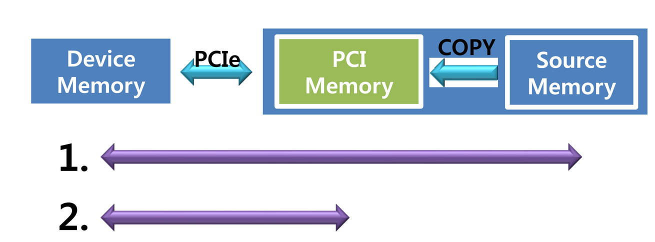
PCI express Bandwidth

▶ Benchmarking PCI express performance shows big diff--rences compared to SDK benchmark programs.

▶ Time measurements

1. `Overlapping by Linux timer between clEnqueue*.`
2. `Profiling queue status from Queued to End.`



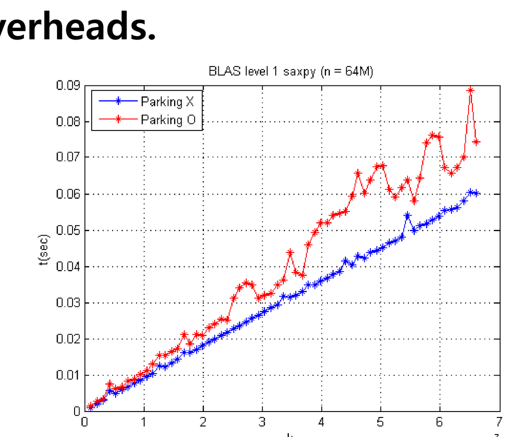


Theoretical	Peak	Q-E	Timer
8.0 GB/s	6.1 GB/s	3.0 GB/s	2.3 GB/s

Windows Problems

▶ CPU parking protocol runs on Windows by default.

▶ This affects the performance in terms of context switching overheads.



Integrated Processors Trials

▶ Integrated processors (Sandy- and Ivy-bridge, APU) share memory bandwidth for both the CPU and GPU.

▶ Simultaneous memory transfer will be serialized.

Conclusion

▶ Heterogeneous computing with OpenCL tends to be very sensitive on its implementation.

▶ Relatively huge time has taken for configuring problems on concurrent executions.

▶ Task-parallel approach covers wider scopes of practical implementation, but it may be hard to split the target algorithm into sub-tasks.