Developing Parallel Krylov Subspace Methods on Heterogeneous Computing Systems

Extreme-scale Research Team
Division of Computational Science in Mathematics

National Institute for Mathematical Sciences (NIMS)

Hyoungseok Chu
(hschu@nims.re.kr)

2014. 06. 27

Contents

1 Introduction

2 Heterogeneous Computing

3 Data-parallel Approach

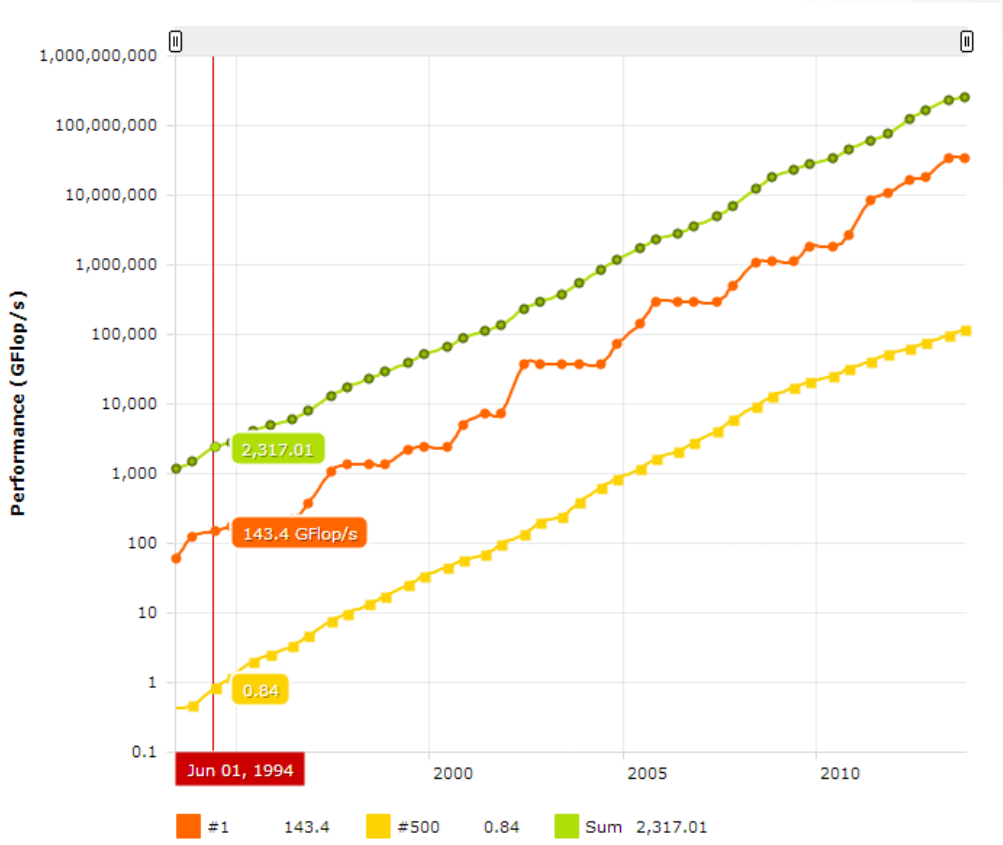
4 Task-parallel Approach

5 Conclusion

Introduction

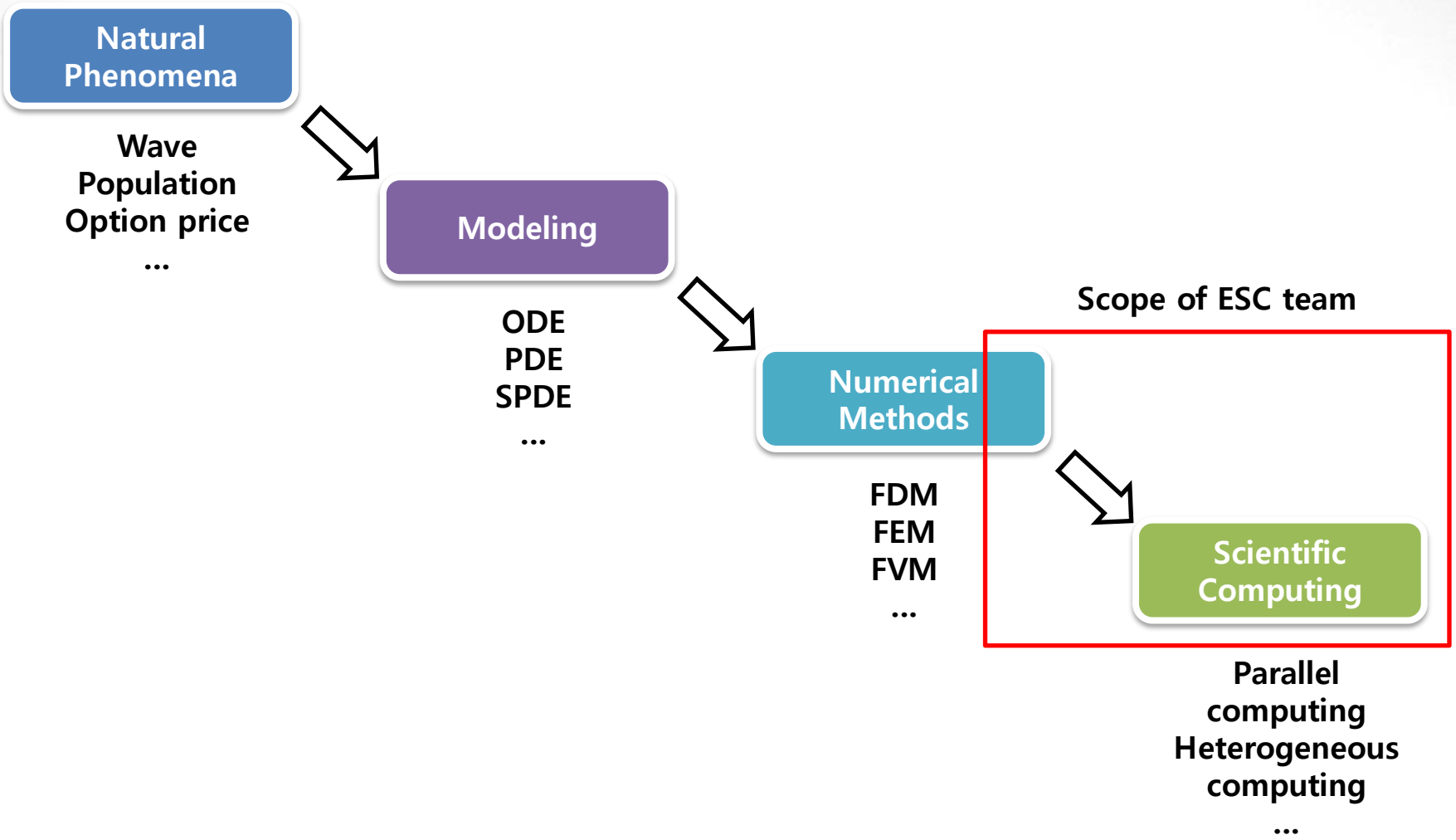
AP Performance

Galaxy S5 (Exynos 5422)
CPU: Cortex A15 + Coretex A7
GPU: ARM Mail-T628
Theoretical FLOPs(GPU): 142 GFLOPs



Top500.org
About $50M, 1994, 143.4GFLOPs

Scientific Simulation



Linear System

Consider following linear system

$$Ax = b,$$

where $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$. Simply the system which has n numbers of linear equations and unknowns.

If we know the inverse matrix of A (i.e. A^{-1}), then the solution $x = A^{-1}b$. Computing inverse is usually extremely heavy $O(n^4)$, and traditional Gaussian type solver, for instance LU factorization, takes $O(n^3)$.

Notice that most of scientific problems derived from PDE and numerical approximation tends to have sparse matrices.

The memory size of full matrix with $n = 1M$ is 8 TBytes (double)

Krylov Subspace Methods (KSM)

KSM is one of top ten algorithm in 20th century. KSM solves the linear system $Ax = b$ iteratively. In other words,

$$x_{j+1} = x_j + \alpha_j d_j.$$

KSM searches the solution with new direction d_j and scalar factor α_j. Choosing direction vector d_j is dependent on the type of linear system. The α_j plays a role that minimize the errors.

Mathematically, KSM guarantees that the solution converges at most n numbers of iterations. The finite precision in modern computers may not satisfy this convergence, which, as a result, makes implementation of KSM more difficult.

Conjugate Gradient Method (CG)

CG is the most popular method of KSMs, and NAS parallel benchmark include CG for evaluating power methods. CG works for the symmetric and positive definite matrix (SPD).

1. $\alpha_j = \frac{r_j^T r_j}{d_j^T A d_j}$
2. $\beta_j = \frac{r_j^T r_j}{r_{j-1}^T r_{j-1}}$
3. $r_j = r_{j-1} + \alpha_{j-1} A d_{j-1}$
4. $x_j = x_{j-1} + \alpha_{j-1} d_{j-1}$
5. $d_j = r_j + \beta_{j-1} d_{j-1}$

1 x Matrix-vector multiplication (GEMV)

2 x dot product (DOT)

3 x scalar multiple of vector addition (AXPY)

BLAS are easy to parallelize, but above routines have bottlenecks on memory transfers.

Preconditioned CG

CG method is unstable if a given linear system has poor condition. This is measurable by means of condition number $\kappa(A) = \|A\| / \|A^{-1}\|$. Preconditioning approaches may resolve unstabilities.

$$MAx = Mb$$

1. $\alpha_j = \frac{r_j^T M^{-1} r_j}{d_j^T A d_j}$
2. $\beta_j = \frac{r_j^T M^{-1} r_j}{r_{j-1}^T M^{-1} r_{j-1}}$
3. $r_j = r_{j-1} + \alpha_{j-1} A d_{j-1}$
4. $x_j = x_{j-1} + \alpha_{j-1} d_{j-1}$
5. $d_j = M^{-1} r_j + \beta_{j-1} d_{j-1}$

1 x Matrix-vector multiplication (GEMV)

2 x dot product (DOT)

3 x scalar multiple of vector addition (AXPY)

+ solve $M^{-1} r_j$!!!

Two major categories of preconditioner
- Incomplete Factorization
- Approximate Inverse

Research Goals

- Strategies for developing KSM on Heterogeneous System
  - CPU+GPU first (CPU+GPUs, CPU+XeonPhi, or CPU+FPGA)
  - Data-parallel approach
  - Task-parallel approach
- Benchmark existing software
- Performance Improvements

Heterogeneous Computing

Introduction

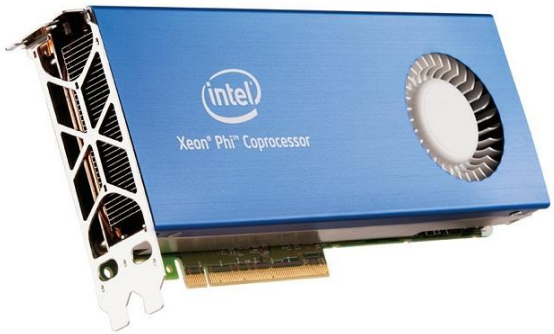
Converting a serial algorithm to parallel one is not easy.
Moreover, parallelization with two different resources will be even harder.
We don't want resources to be in idle. Rather than that we should struggle with using resources as much as possible.

Heterogeneous Computing means utilizing computational resources whose architectural instruction sets are different. CPU+GPU or CPU+GPUs are typical example, and there are increasing demands for other resources such as XeonPhi, FPGA, Cell B/E, and AP.

Accelerators

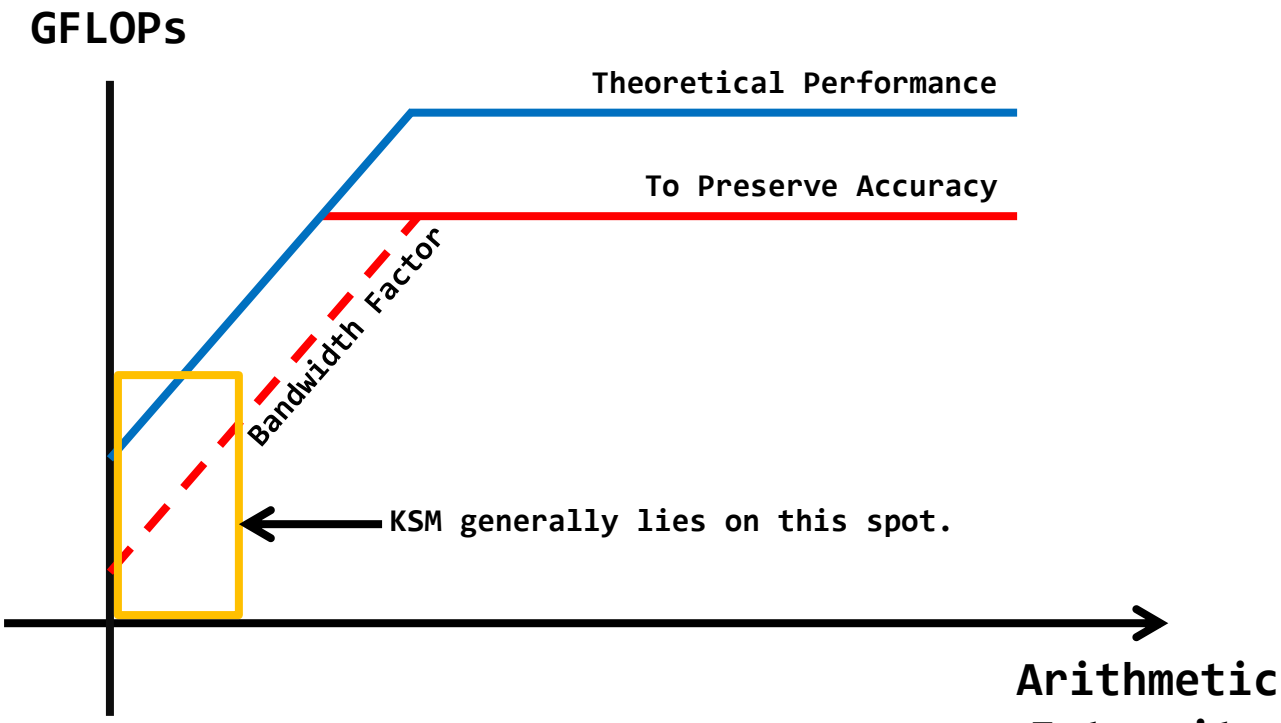
NVIDIA GTX 690
3072 cores
Clock Speed: 1.0 GHz
Memory: 6GB
Theoretical FLOPs:
5620 GFLOPs

ATI Radeon 7990
4096 cores
Clock Speed: 925 MHz
Memory: 6GB
Theoretical FLOPs:
8200 GFLOPs



Intel Xeon Phi 7120P
61 cores
Clock Speed: 1.3 GHz
Memory: 16GB
Theoretical FLOPs:
2596 GFLOPs

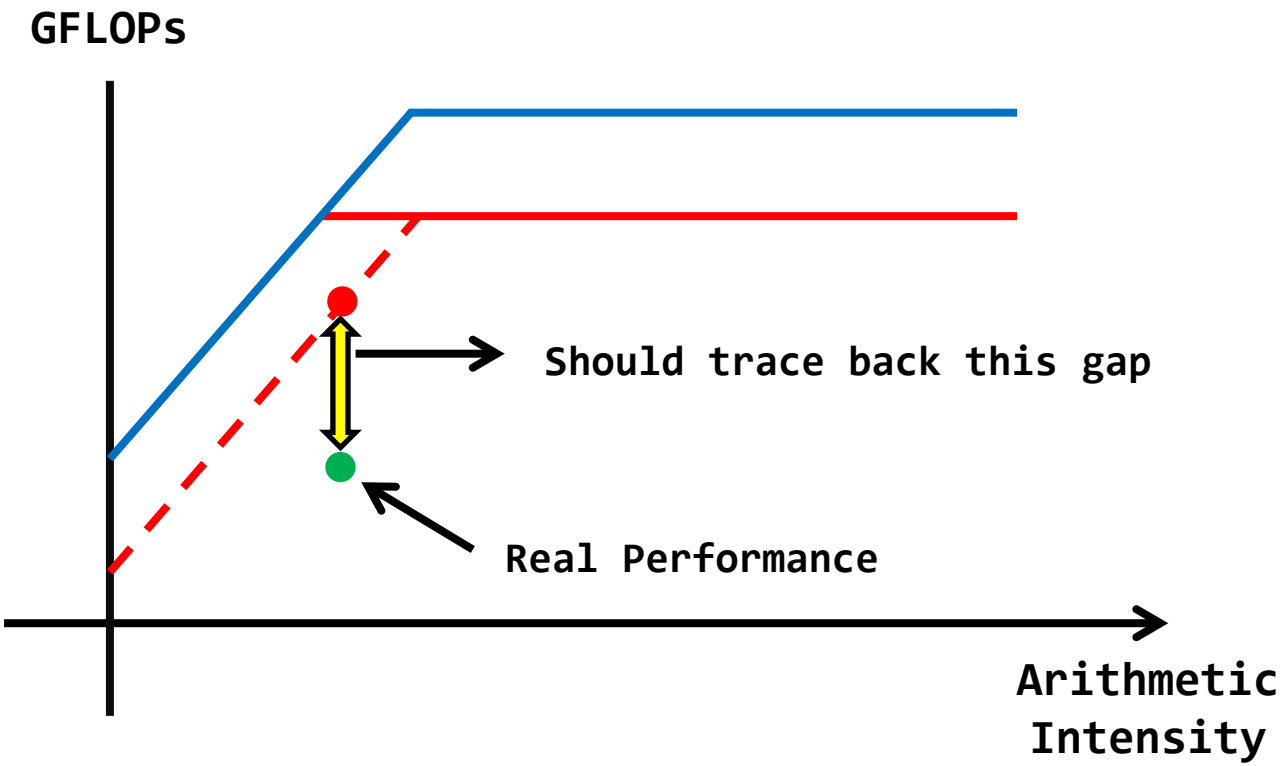
Roofline Model



$$r(AI) = \min(F, B \times AI)$$

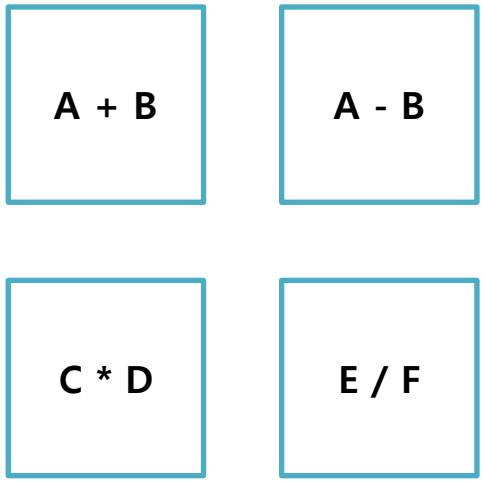
F: Theoretical FLOPs, B: Theoretical Bandwidth

Verifications



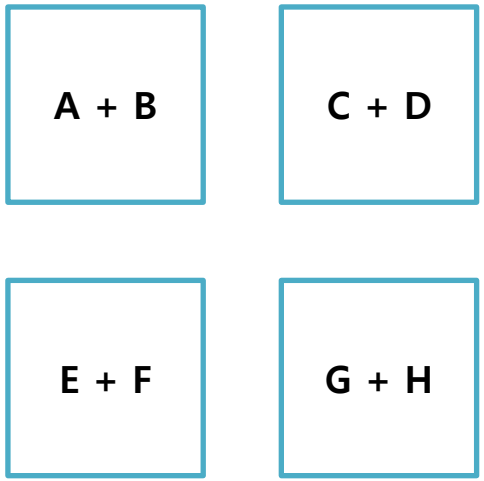
Parallel Methodologies

- Task parallel



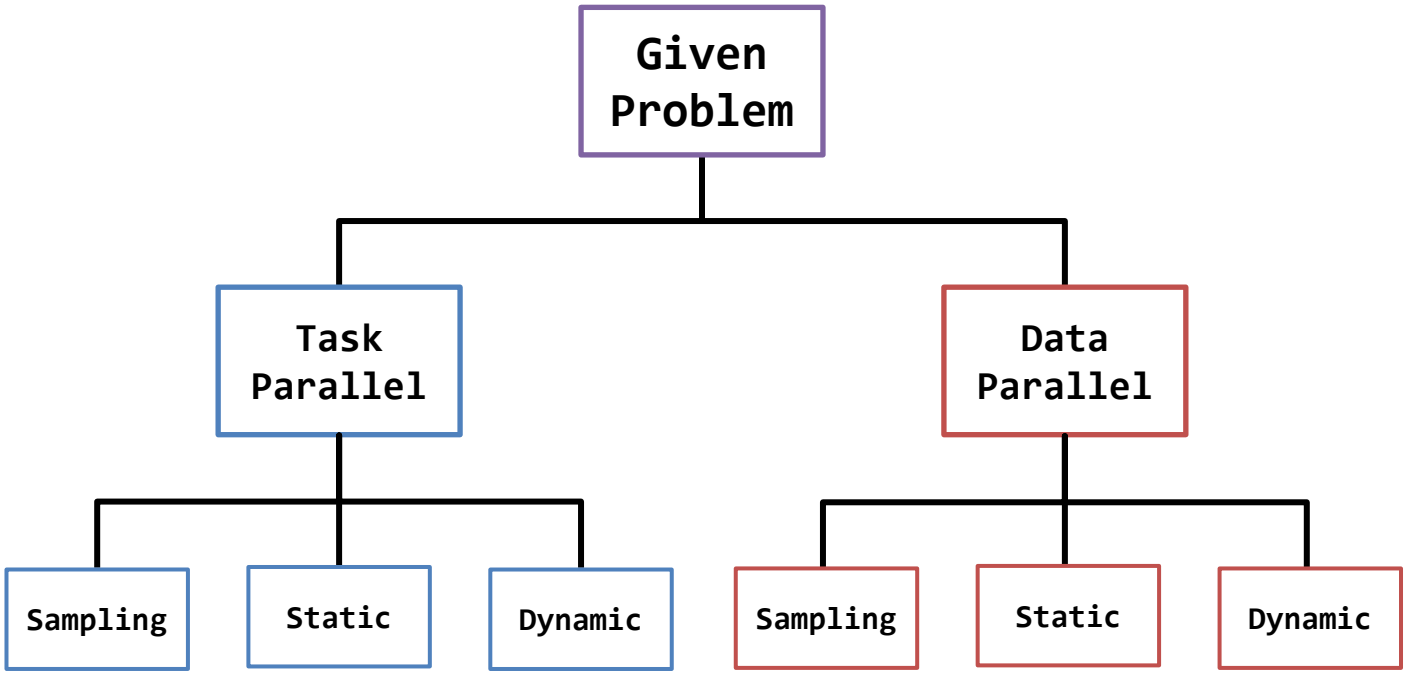
Multi-core CPU,
Clusters

- Data parallel



Many-core GPU,
Vector machine

Strategies


Given Problem
Task Parallel
Data Parallel
Sampling
Static
Dynamic
Sampling
Static
Dynamic

Subdivision

1) Sampling

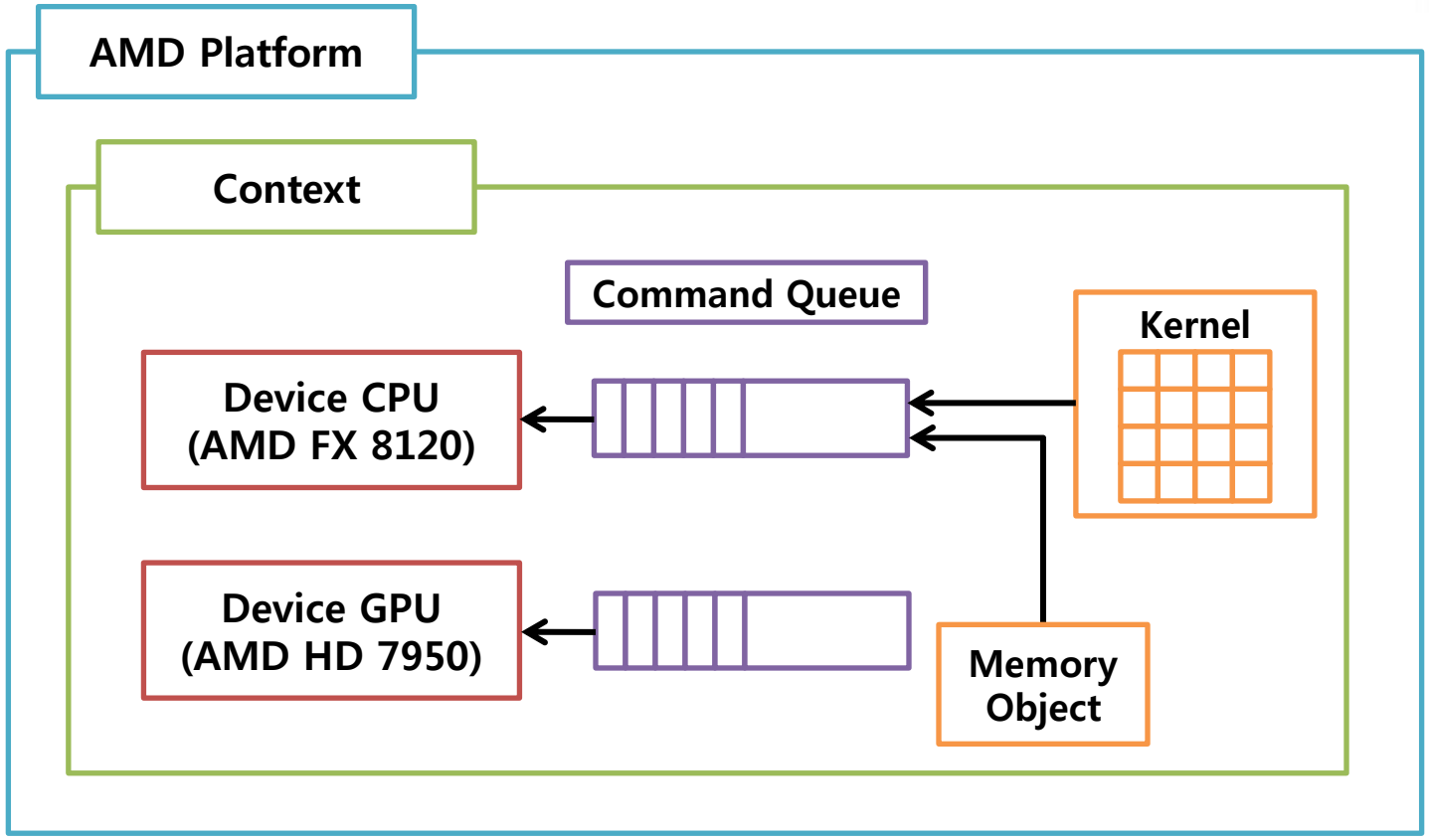
- Gathers pre-computed results to conclude a balancing point.
- Regression or interpolation methods are applicable to see what values are presented in intervals among pre-computed time.

2) Static

- Concludes a load balancing point without any pre-computation.
- Total execution time is mainly composed of amount of computation, data transfer, latencies and overheads.

3) Dynamic

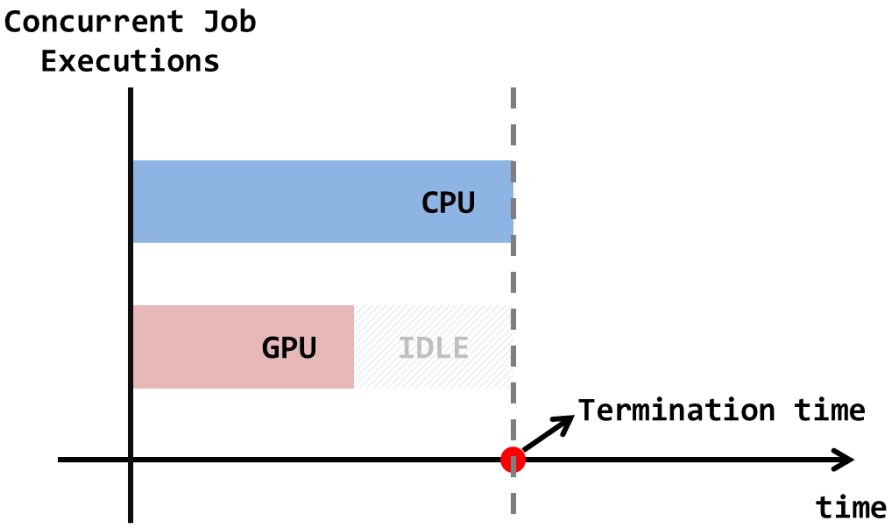
- Balances given workloads by applying a runtime scheduler.
- Tuning algorithm runs several times in order to specify the amount of workloads to be balanced. This appoarch aims at more general cases

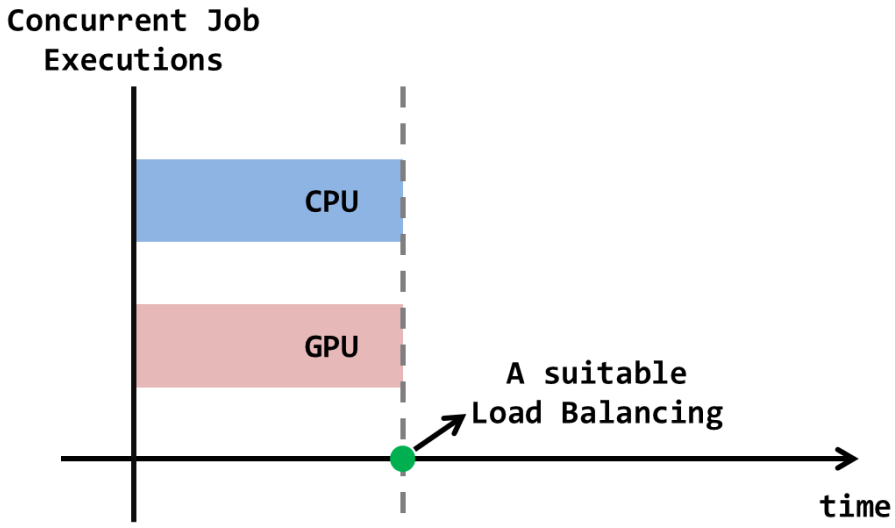


OpenCL supports concurrent executions of command queues.

Data-parallel Approach

Motivation


Concurrent Job Executions
CPU
GPU
IDLE
Termination time
time


Concurrent Job Executions
CPU
GPU
A suitable Load Balancing
time

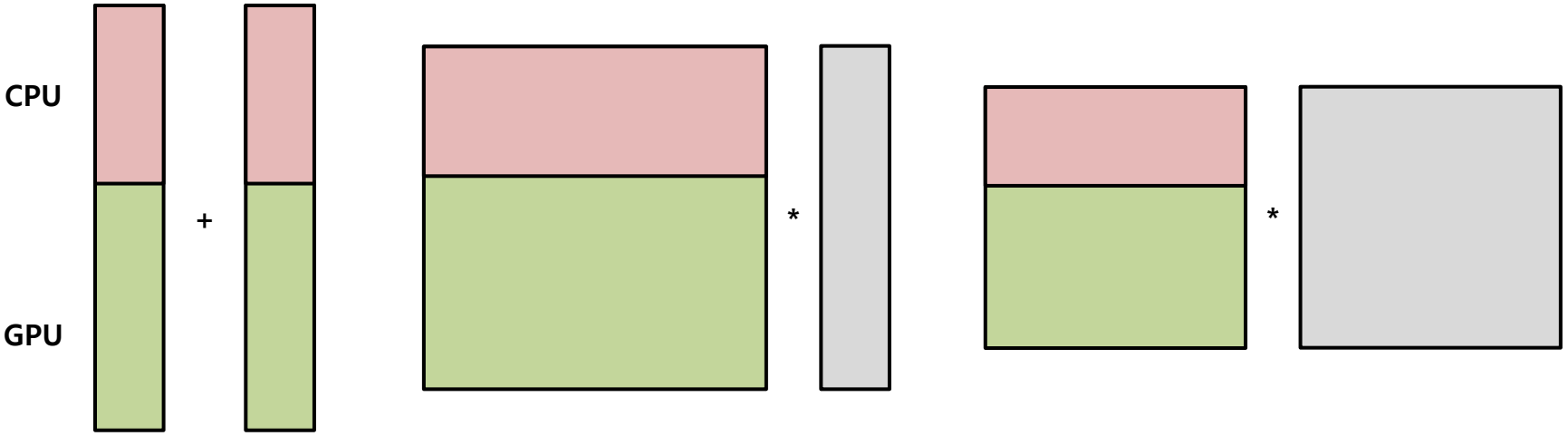
Taget applications

Target applications

BLAS level 1 : saxpy

BLAS level 2 : sgemv

BLAS level 3 : sgemm



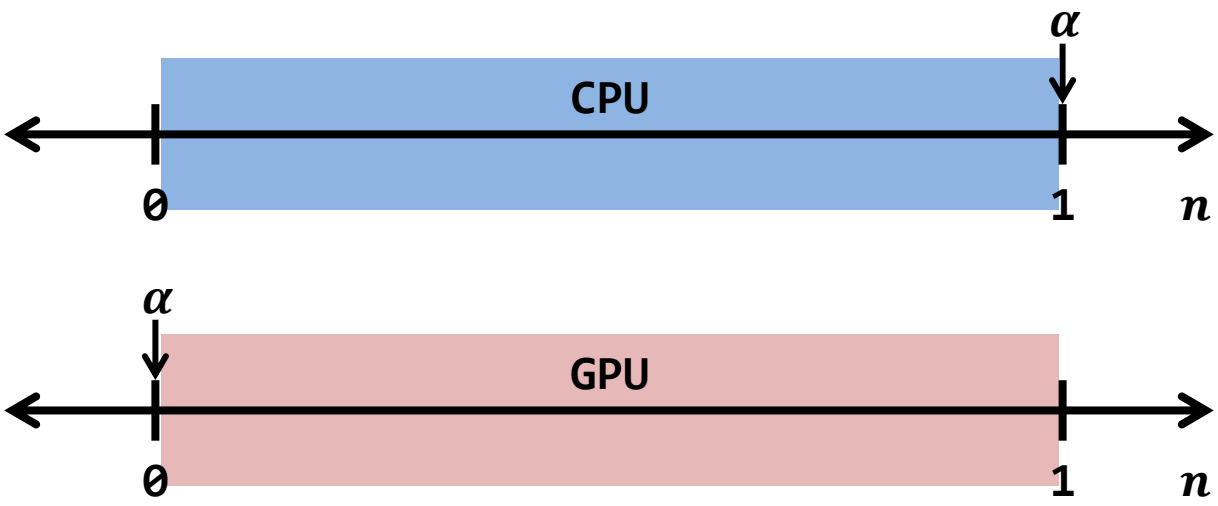
BLAS level 1 BLAS level 2 BLAS level 3

Finding α (1/2)

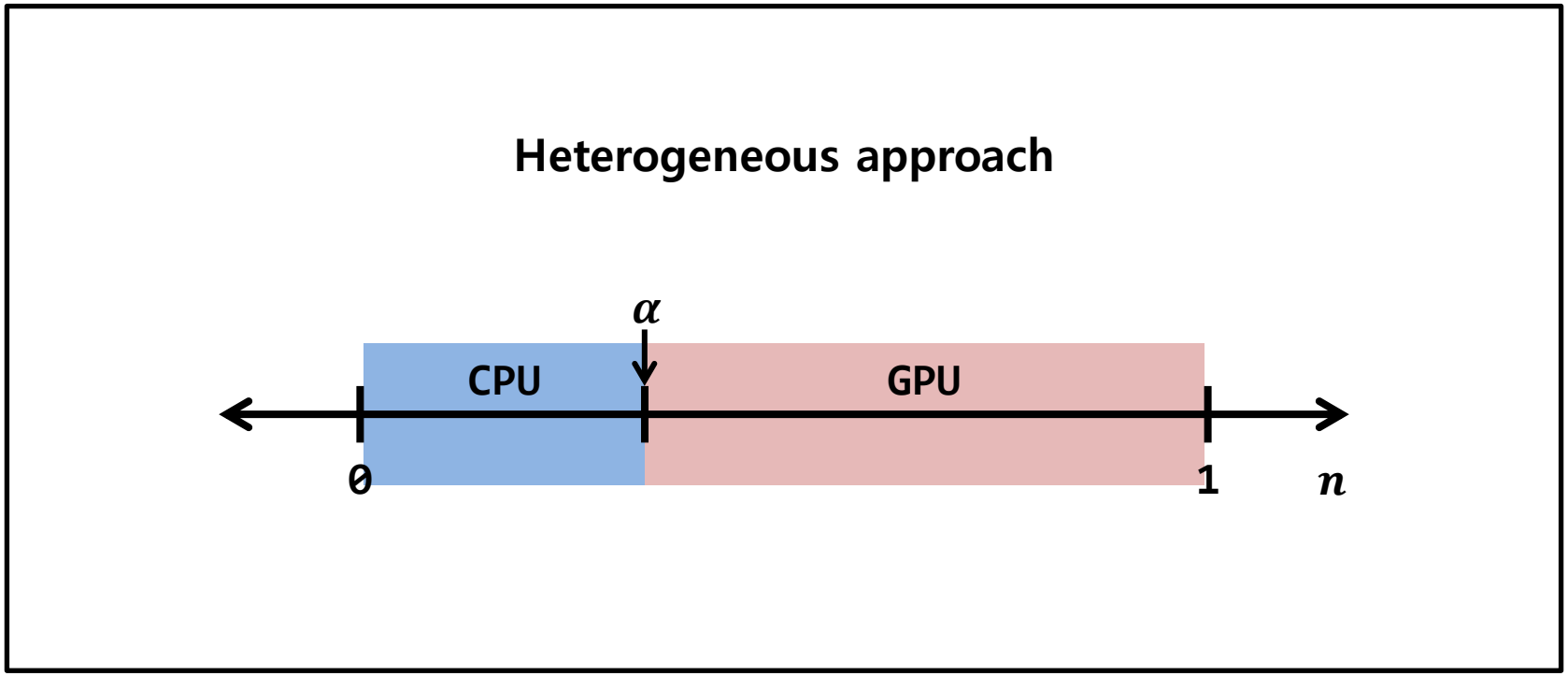
Using one specific computational resource.

CPU holds αn number of operations whereas

$(1-\alpha)n$ operations are assigned into GPU.

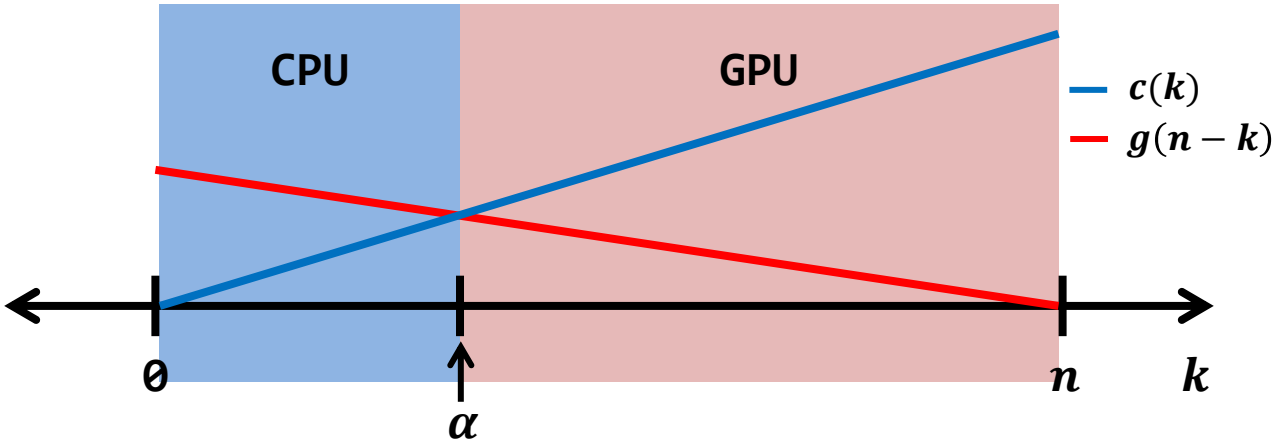


Finding α (2/2)



Load Balancing

1. Develop a min-max model between a CPU and GPU computational time.
2. Model functions $c(n)$ and $g(n)$.
3. Put those functions as objective functions of min-max model.
4. Generate sample data and regress those data.
5. Compare with sampling data and real heterogeneous computing results.



Modeling

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.

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

Assign αn amount of data to the CPU and $(1-\alpha)n$ to the GPU.

Reference Parameters

$$c(k) = \frac{3 \times 4bytes \times k}{B_{CPU}} + \frac{k}{F_{CPU}} + O_{CPU}$$

$$g(k) = \frac{2 \times 4bytes \times k}{B_{PCI,W}} + \frac{4bytes \times k}{B_{PCI,R}} + \frac{3 \times 4bytes \times k}{B_{GPU}} + \frac{k}{F_{GPU}} + O_{GPU}$$

We call parameters of above equations as reference parameters. Those parameters are estimated as follows. For example B_{GPU},

$$B_{GPU} = \frac{B_{GPU}^{64M} + B_{GPU}^{56M} + B_{GPU}^{48M} + \cdots}{number\ of\ samples}$$

Target Platform

Parameters	CPU AMD FX 8120	GPU AMD HD 7950
Number of compute units	8 (4)	28
Core Clock speed	3.1 GHz	0.8 GHz
Global Memory	16 GB	3 GB
Operating System	Windows 7	Windows 7
Architecture	64-bit	64-bit
OpenCL SDK	2.8	2.8
Peak Performance (single)	74.4 GFLOPs	2867 GFLOPs
Global Memory Bandwidth	21.3 GB/s	240 GB/s

Parameter Table

Parameters	PCI writing		PCI reading	
	Theoretical	Benchmarked	Theoretical	Benchmarked
Bandwidth	8.0 GB/s	2.686 GB/s	8.0 GB/s	3.353 GB/s
Minor Bandwidth		1.214 GB/s		0.150 GB/s

Parameters	CPU		GPU	
	Theoretical	Benchmarked	Theoretical	Benchmarked
Global memory bandwidth	21.3 GB/s	13.527 GB/s	240 GB/s	154.995 GB/s
Constant Overheads		3.528E-04 sec		2.527E-03 sec

BLAS Level 2

matrix vector multiplication with fixed n (sgemv)

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

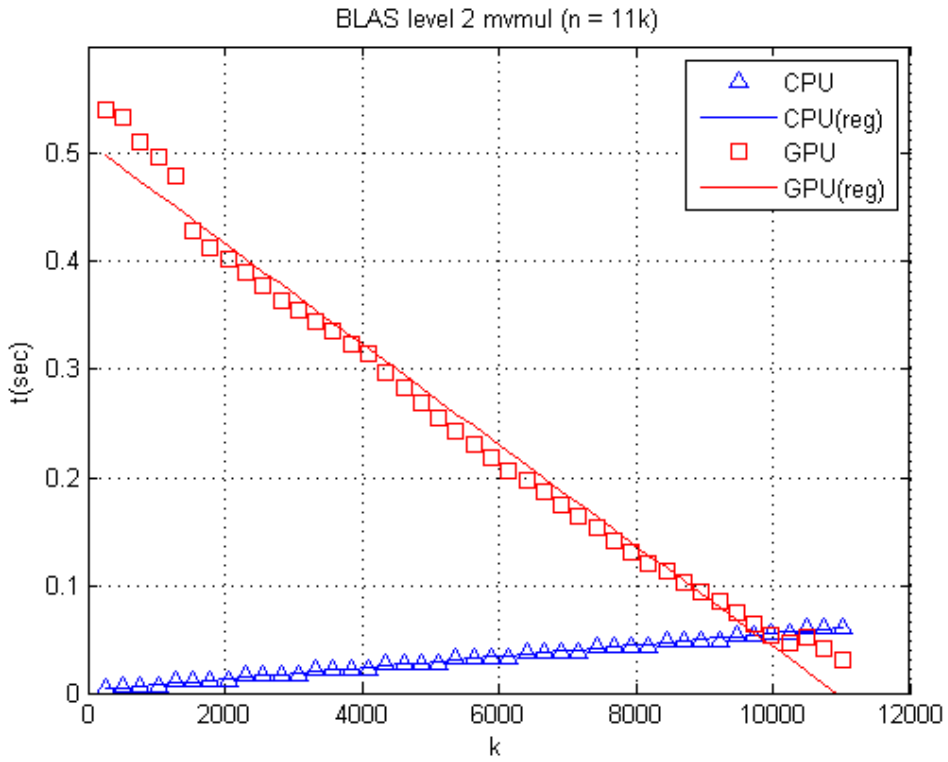
$$g(k) = \frac{4bytes \times (n+1)}{B_{PCI,W}} k + \frac{4bytes}{B_{PCI,R}} k + \frac{4bytes \times (n+2)}{B_{GPU}} k + \frac{n}{F_{GPU}} k + O_{GPU}$$

Parameters Table

Parameters	CPU	GPU
	Reference	Reference
Flops (F)	3.69 Gflops	2867 Gflops
Global memory bandwidth (B)	13 GB/s	4.757 GB/s
Constant Overheads (O)	3.528E-04 sec	2.527E-03 sec

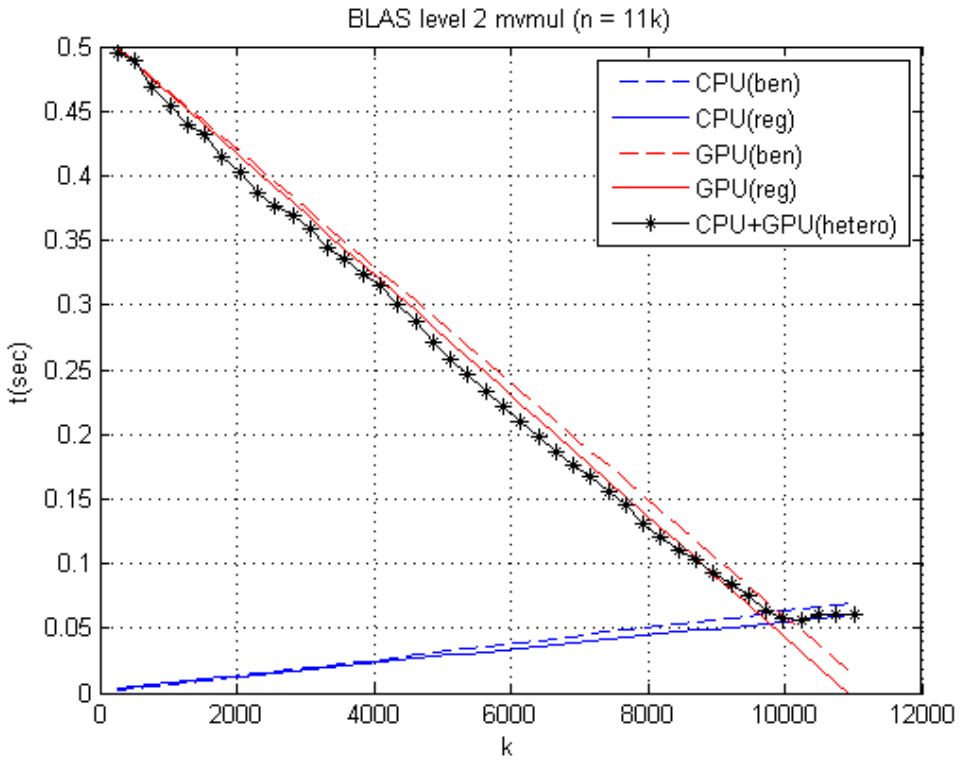
BLAS Level 2

1. Generating sample and regression



BLAS Level 2

2. Comparing with heterogeneous computing results

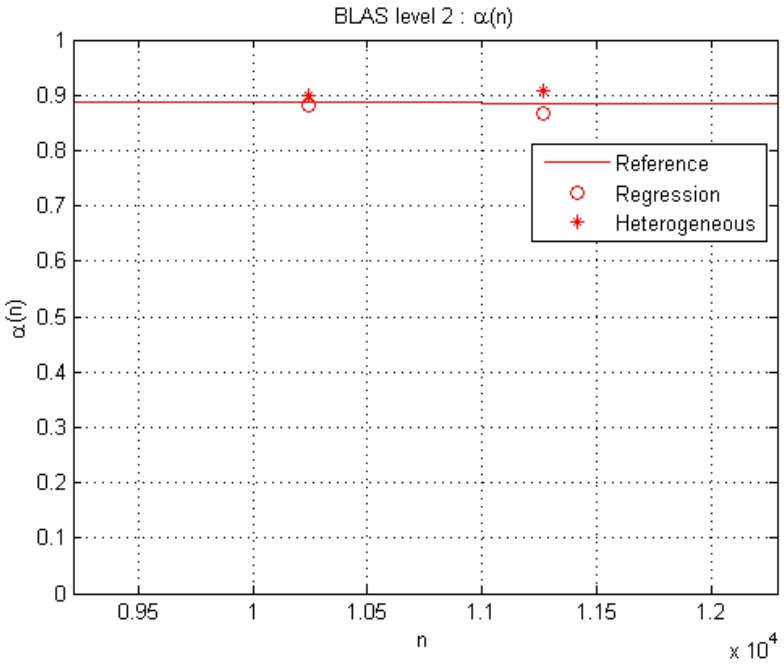


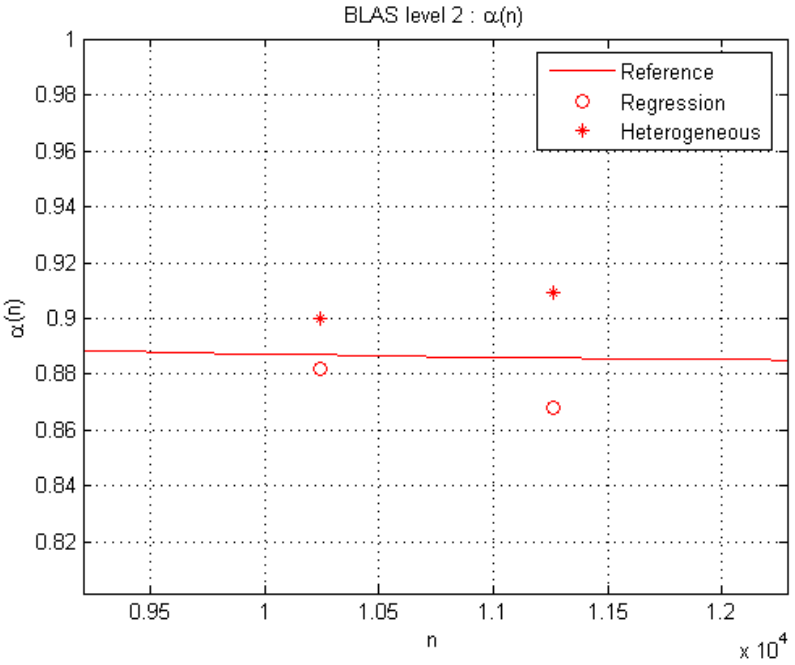
3. Splitting ratio and execution time

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

BLAS Level 2

4. Evaluating $\alpha(n)$



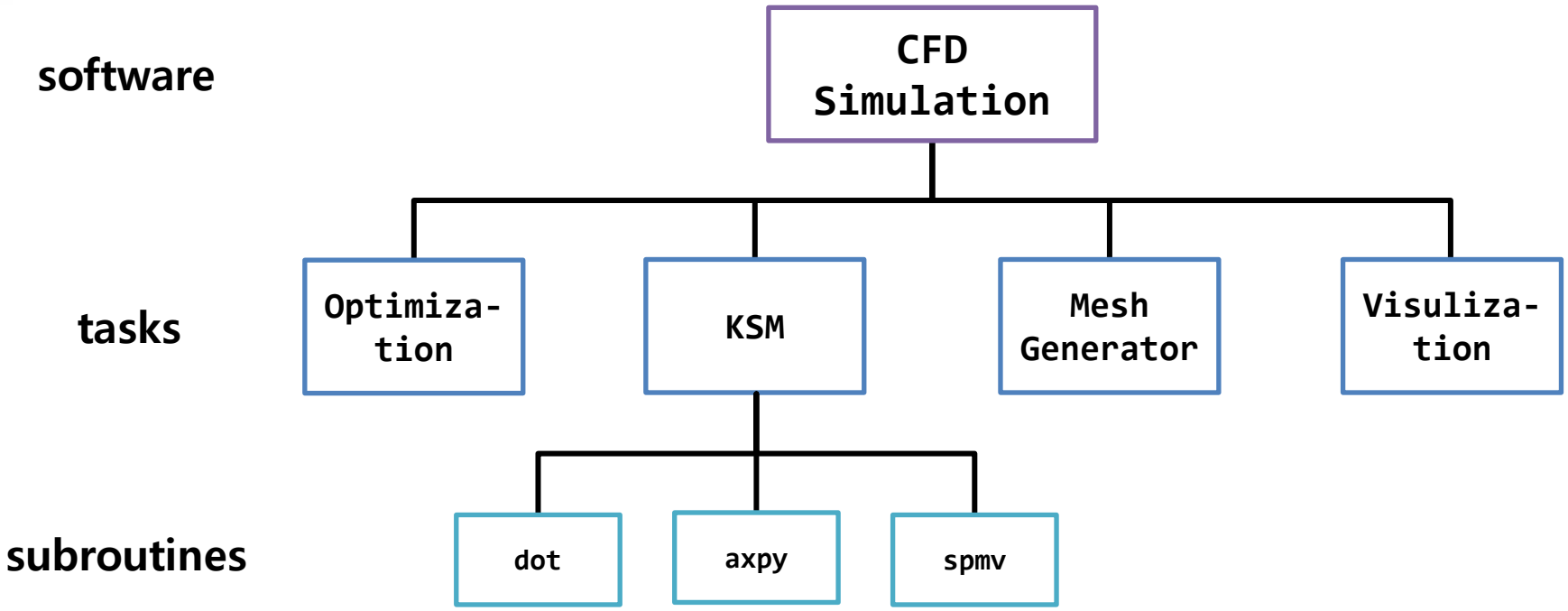


Task-parallel Approach

Motivation

- Performances of heterogeneous BLAS are poor.
  - The balancing point is 9:1. (CPU:GPU)
- KSM loads all data from host just once. We don't need to communicate data for each BLAS of KSM that may cause another issues.
- Preconditioned KSM is more applicable to task-parallel approach since splitting KSM into several independent task will be meaningless and even harder.

Examples

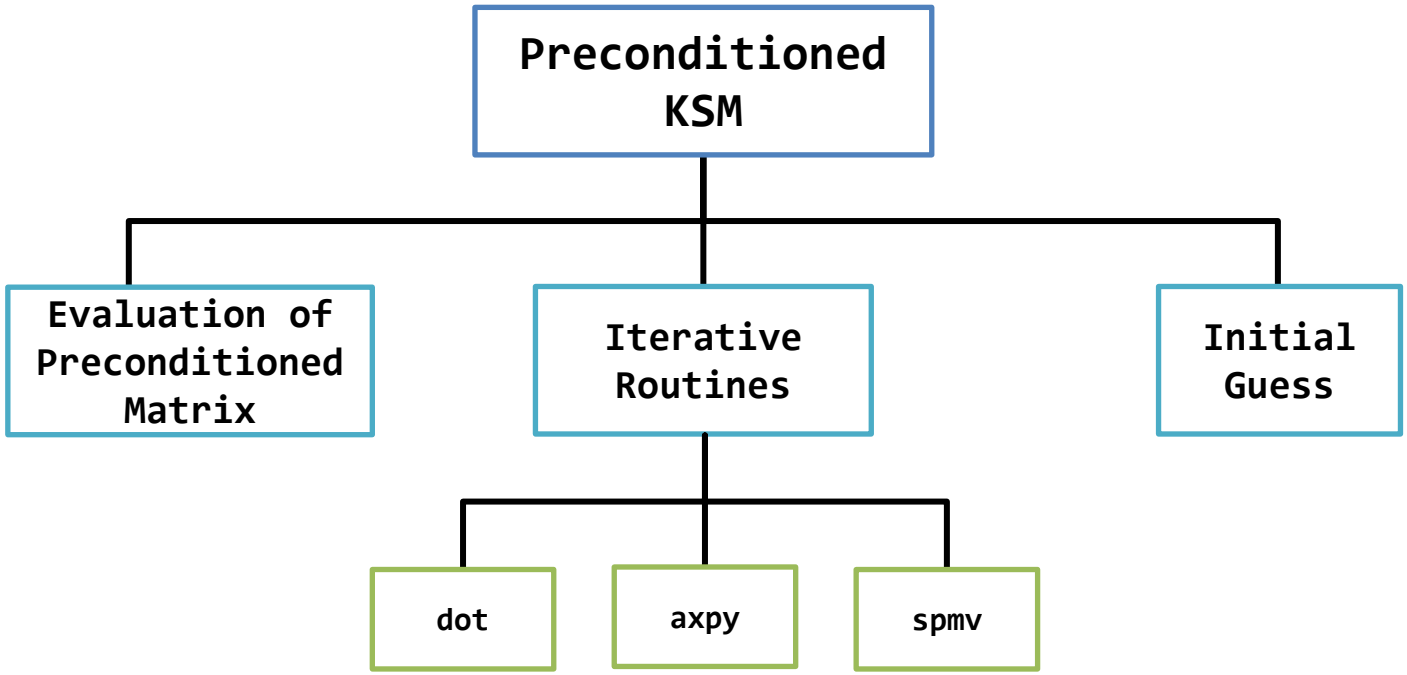


Those subroutines have dependencies!

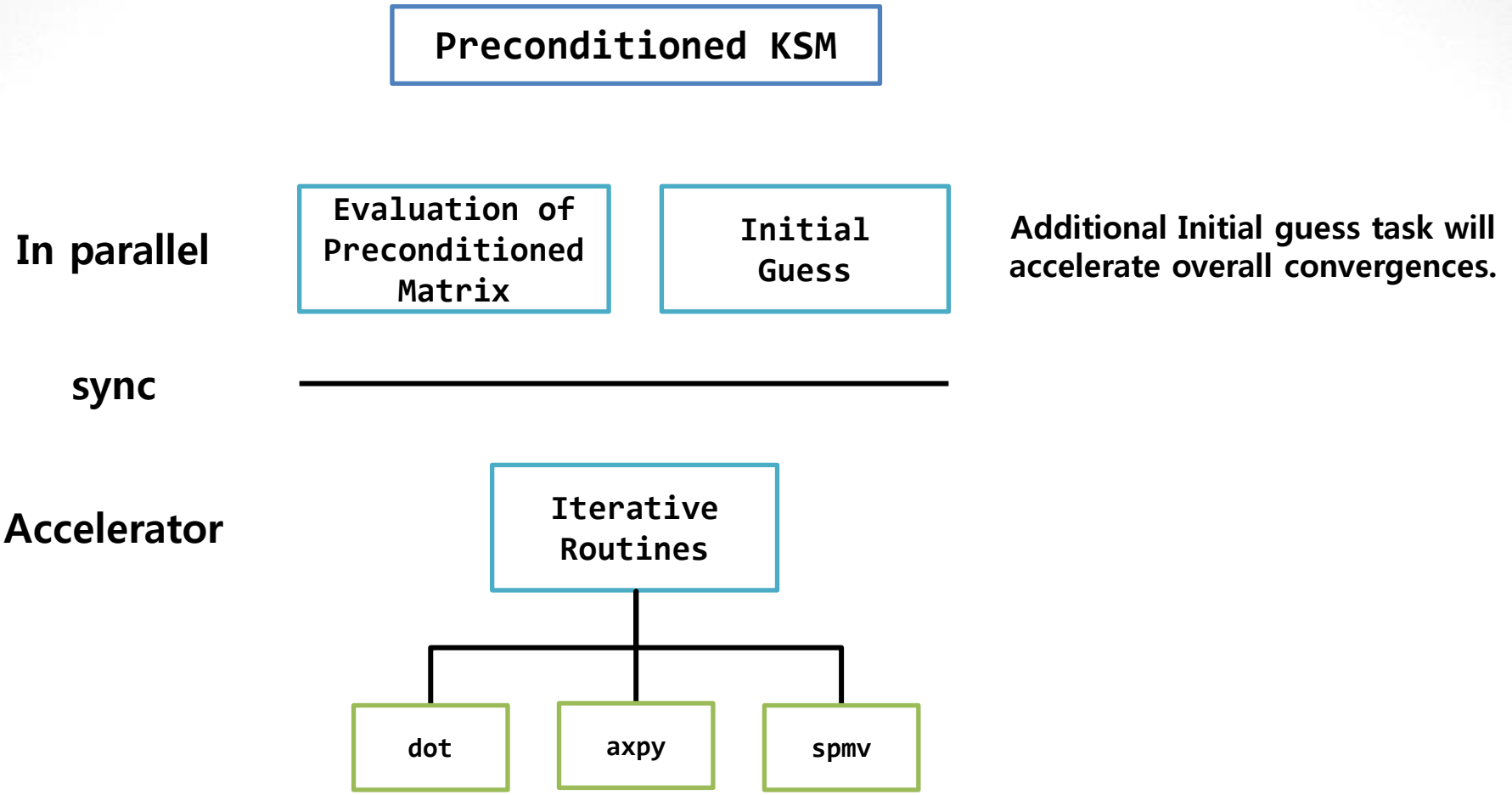
Preconditioned KSM

tasks

sub-tasks



Supplementary Acceleration



Additional Initial guess task will accelerate overall convergences.

Parallelizability

ILU

SPAI

Initial Smoothing (SOR, G-S,...)

Approximate Inverse

KSM

axpy

dot

spmv

$M^{-1}r = d$

Incomplete Factorization

KSM

axpy

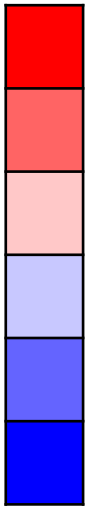
dot

spmv

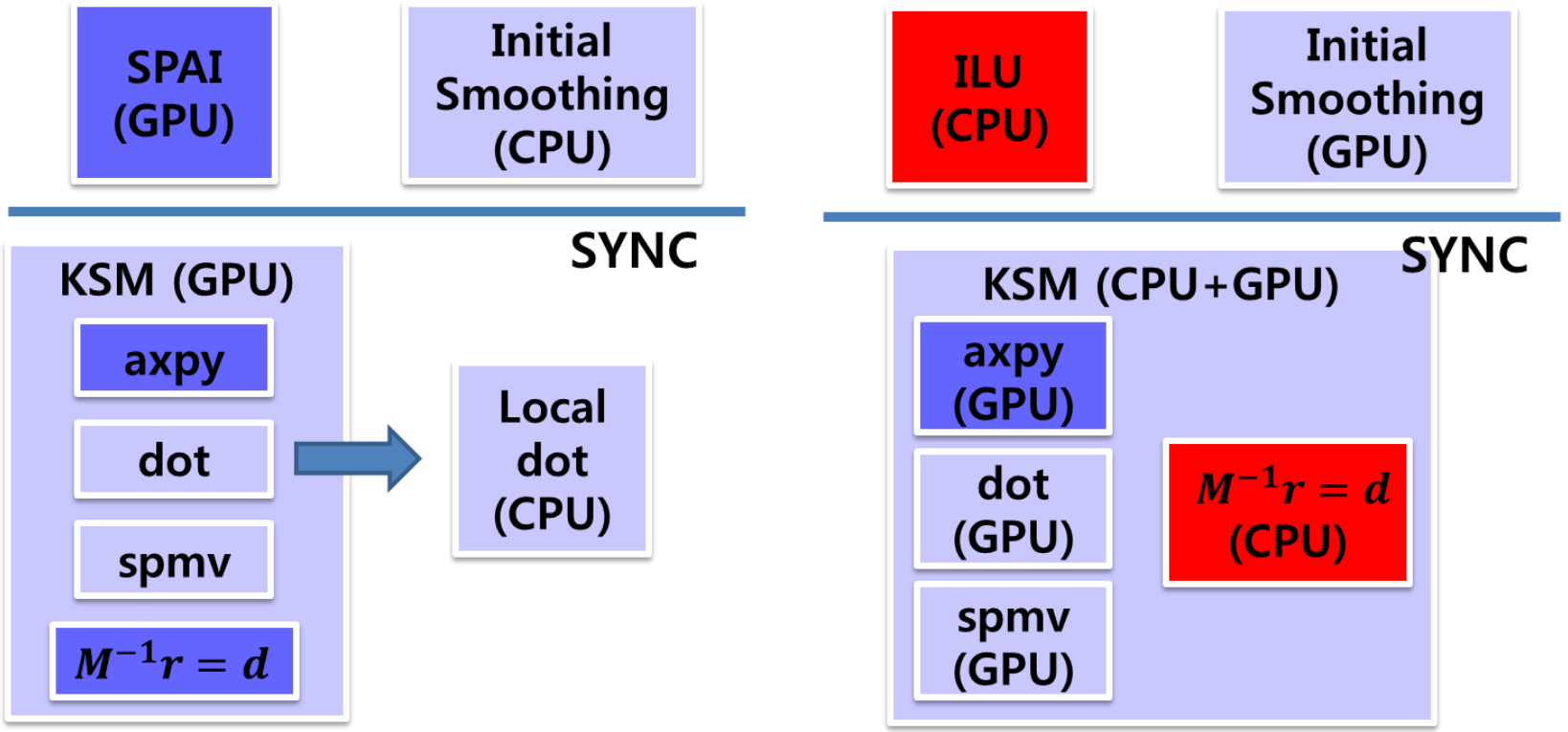
$M^{-1}r = d$

Serial

Parallel



Task-parallel Strategy



Progresses

- Parallel CG (OpenCL)
  - CPU or GPU only, CPU+GPU hybrid
  - dot(hybrid), spmv, axpy
- Review preconditioning methodologies
  - ILU, SPAI, MR
- FPGA implementation
  - S-step method (pipelining)
  - Pipelined BLAS

Conclusion

Conclusion

- Heterogeneous Computing Strategies using OpenCL
  - Data-parallel: Load balancing
  - Task-parallel: Supplementary acceleration
- Data transfer dependencies for KSM degrade overall performances (relatively small AI).
- Heterogeneous computing will be an inevitable choice since processor venders will adopt more and more cores in parallel.

Thank you