PLB-HAC: Dynamic Load-Balancing for Heterogeneous Accelerator Clusters

Göttingen, Germany

Luis Sant'Ana[1] Daniel Cordeiro[2] **Raphael Y. de Camargo**[1]

raphael.camargo@ufabc.edu.br

[1]Universidade Federal do ABC, Brazil

[2]Universidade de São Paulo, Brazil

EURO-PAR CONFERENCE 2019

FAPESP

CAPES

UFABC

Outline

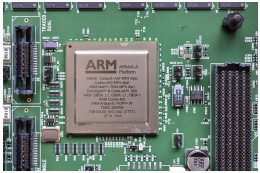
Introduction

Proposed Algorithm

Experimental Results

Introduction

Heterogeneous Clusters



Many types of processing units exist in current clusters

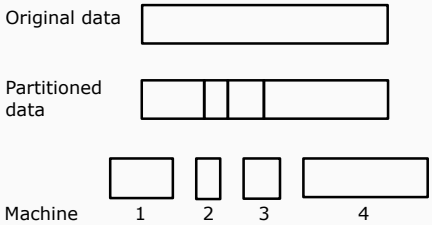
- x86 CPUs, RISC CPUs, FPGAs, Many-cores and GPUs

Heterogeneity in supercomputers and custom HPC platforms is becoming common

For clusters using commodity machines, the heterogeneity level is even higher

- A new generation of hardware is launched every couple of years
- Researchers want to use all existing machines to increase the availability of resources

Load Balancing



The problem with heterogeneous machines is on how to distribute computation among processing units (PUs)

Load-balancing mechanism for all kinds of applications is not feasible

We focus on data-parallel applications

- Data-parallel applications can have its data divided in smaller blocks
- The blocks are processed independently in parallel
- Results are merged at the end

UFABC

Approaches for Load-Balancing in Heterogeneous Clusters

- Some simple heuristics to distribute data
  - Master-worker architectures and work stealing
  - May result in suboptimal distributions at the end of simulations
  - Prevents optimizations such as data prefetching
- Distribute data using simple criteria, such as peak performance (in GFLOPS)
  - Is ineffective, as performance on each architecture is application dependent
  - The relationship between input size and execution time is frequently non-linear
- Use more elaborate load-balancing algorithms
  - Are normally specific for certain classes of applications and cause a higher overhead
  - Compensate with better task distribution and potentially smaller execution time

Existing approaches for Data-Parallel Applications

Divide data into small chunks and profile their execution time in each PU

Use simple relative processing power values

- Belviranli *et al.* (2013) → Heterogeneous Dynamic Self-Scheduler (HDSS) for GPUs
  Initial phase, where blocks of different sizes are sent to GPUs
  A RP (relative performance) value is assessed for each GPU
- Kaleen *et al.* (2014) → different processing rates Gr and Cr for GPUs and CPUs

Generate more detailed performance profiles of tasks on each PU type

- de Camargo *et al.* (2015) → Use of a execution time vs block size curve for each GPU

Zhong et al. (2012) *!* workload is split using an Functional Performance Model, but which requires prior information about the problem

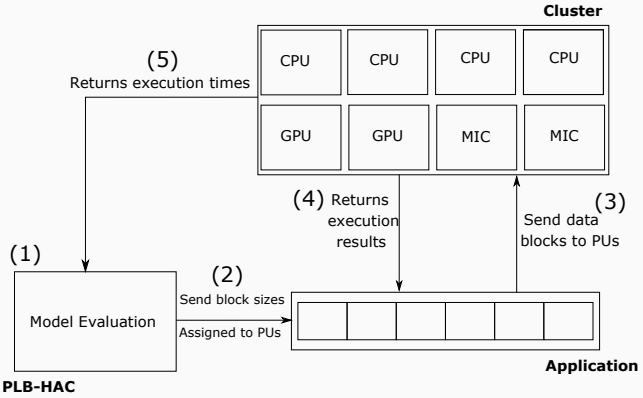
Existing work focus mainly on GPUs

Generate static distributions, preventing adaptations to changes

They also contain multiple synchronization steps that reduces performance

Proposed Algorithm

Overview

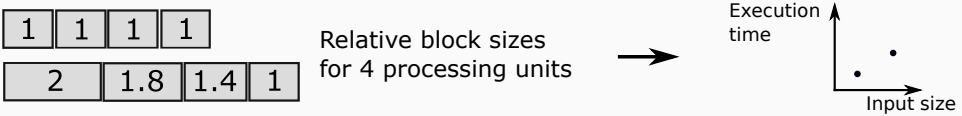


Create performance model for each PU ! determine block size distribution

Dispatching of blocks for execution and results collection by the application

Analysis of execution times by PLB-HAC to update the performance models

Initial Performance Modeling



Application for which we have no information of execution time behavior

Devise a performance model for each PU based on execution time measurements

First step: sends a block of size x_{init} to each unit

Next step: double block size to fastest unit

- Other units receive blocks size proportional to their speed
- Units should complete execution approximately at the same time

With two points, an initial model is devised by fitting a line to the points

Continuous Performance Modeling

More measurements obtained ! better performance models for the execution time on the different PUs

Use least-squares to find the best curve fit using:

$$F_p[x] = a_1 f_1(x) + a_2 f_2(x) + ... + a_n f_n(x)$$

where $f_i(x)$ is one function of the set $\ln x$, x, x^2, x^3, e^x, $x \quad e^x$, and $x \quad \ln x$

Should work for any kind of Processing Unit

For modeling the time spent transmitting the data, we use

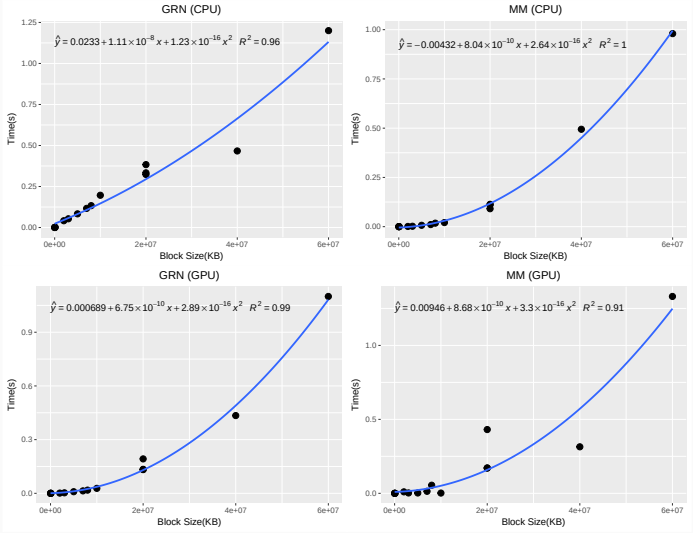
$$G_p[x] = a_1 x + a_2$$

Captures time required to transfer data to different kinds of PUs

The total execution time is given by:

$$E_p[x] = F_p[x] + G_p[x]$$

Examples of Curve Fitting



Execution times and performance models using a CPU (Top) and GPU (Bottom) for the gene regulatory network (GRN) and matrix multiplication (MM) applications.

UFABC

Assignment of Block Sizes

Objective: divide data chunks of k bytes, normalized to 1, among the PUs

Determine the set X of block sizes x_g for each processor

$$X = f x_g \ 2 \ \mathbb{R} : [0, 1] \, j \sum_{g=1}^{n} x_g = 1 g$$

Same execution time $E_k(x_k)$ on each processor k

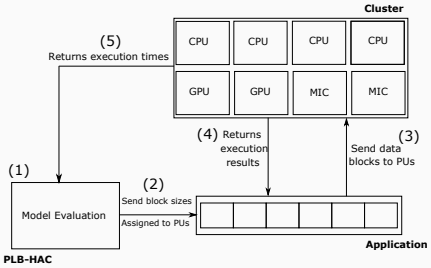
$$E_1(x_1) = E_2(x_2) = \quad = E_n(x_n)$$

To find X, we need to solve the following set of equations:

$$\begin{cases} E_1(x_1) = F_1(x_1) + G_1(x_1) \\ E_2(x_2) = F_2(x_2) + G_2(x_2) \\ \dots \\ E_n(x_n) = F_n(x_n) + G_n(x_n) \end{cases} \tag{1}$$

Apply an interior point line search filter method

Application Execution



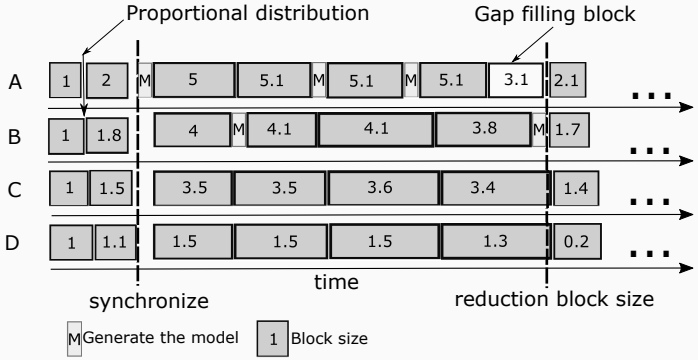
The scheduler sends a list of blocks sizes x_g for the application

x_g is a floating-point number, which is rounded to the closest valid block size

The application dispatches blocks of these sizes to the processing units g.

When a PU finishes executing a task, it returns the results to the application and the execution times to PLB-HAC.

Execution Control and Rebalancing (1/2)



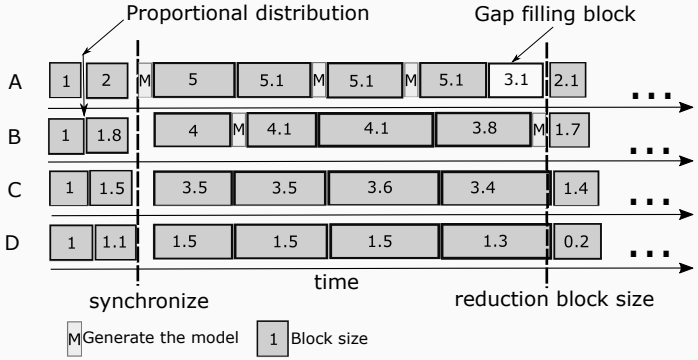
Initial phase ! PLB-HAC creates an initial model which is used to determine task sizes

The remaining of execution is divided in *virtual steps*

- Each PU receives a block size according to its performance model
- Performance model is re-evaluated after the end of the virtual step

This monitoring allows the algorithm to respond to changes in the execution environment

Execution Control and Rebalancing (2/2)



If the execution time is far from the expected value, a gap filling block is executed

- The objective is to maintain a virtual synchronization between tasks
- Threshold is sufficiently large to permit the efficient use of PU resources

A final phase where blocks are smaller to reduce the impact of sub-optimal distributions

- Execution is similar, but block sizes are smaller to reduce the effects of unbalances

UFABC

Experimental Results

Implementation Using StarPU

StarPU ! a task-based programming library for hybrid architectures

- Runtime layer that manages the execution of tasks and data transferring between processing units
- Support for CPU, GPU, and Xeon Phi ! implementations using *codelets*
- Offers an API that allows the implementation of new scheduling policies
- Default scheduling strategy is the greedy one

Implemented PLB-HAC over the StarPU framework

Also implemented the HDSS algorithm for comparison

- Determines a RP (Relative Power) value for each GPU, which is use to select block sizes

Applications

We ported three applications to the StarPU Framework

- GPU, CPU and Xeon Phi implementations

Matrix Multiplication

- We used the optimized version from the CUBLAS 4.0 library
- Computational complexity: $O(n^3)$ for an $n \quad n$ matrix.

Gene Regulatory Network (GRN) inference

- Exhaustive search of gene subset with cardinality k that best predict a target gene
- Computational complexity: $O(n^k)$, where n is the number of genes.

K-means clustering algorithm

- Partitions n d-dimensional observations into k clusters
- Computational complexity: $O(n^{d \cdot k+1})$

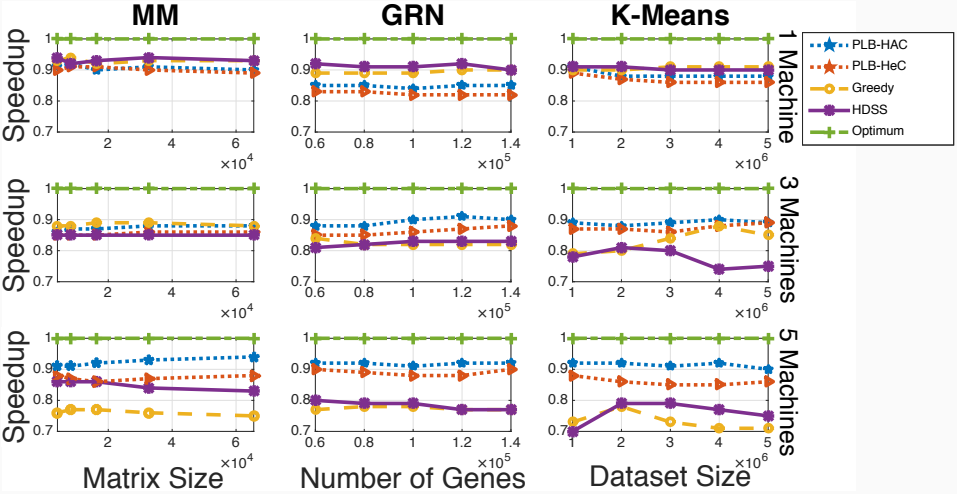
Machine Configurations

Table 1: Machine configurations

Machines	Description				
	PU type	**Model**	**Core count**	**Cache/Throughput**	**Memory**
A	**CPU**	Intel i7 - 5930K	6 cores @ 3.5 GHz	15 MB cache	32 GB
	GPU	Quadro K5200	2304 cores	192 GB/s	8 GB
B	**CPU**	Intel i7 - 5930K	6 cores @ 3.5 GHz	15 MB cache	32 GB
	GPU	GTX 970	1667 cores	224 GB/s	4 GB
	Xeon Phi	3120 series	57 cores	240 GB/s	6 GB
C	**CPU**	Intel Xeon E-2620	6 cores @ 2.10 GHz	15 MB cache	32 GB
	GPU	Quadro K620	384 cores	29 GB/s	2 GB
D	**CPU**	Intel i7-4930k	6 cores @ 3.40 GHz	12 MB cache	32 GB
	GPU	GPU Titan	2688 cores	288.4 GB/s	6 GB
E	**CPU**	Intel Xeon E-2620	6 cores @ 2.10 GHz	15 MB cache	32 GB
	GPU	Quadro K620	384 cores	29 GB/s	2 GB
	Xeon Phi	3120 series	57 cores	240 GB/s	6 GB

Setups: 1 machine [A], 3 machines [A, B, C], and 5 machines[A, B, C, D, E]

Speedup vs Optimal distribution

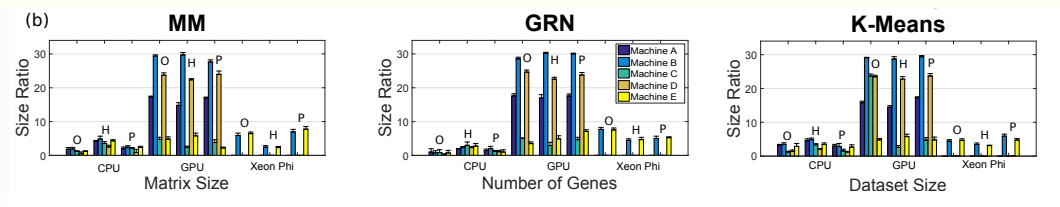


Matrix Multiplication (MM), Gene Regulatory Network (GRN) and K-Means applications

PLB-HAC was mostly within 10% of the optimal and better than other algorithms

UFABC

Block Size

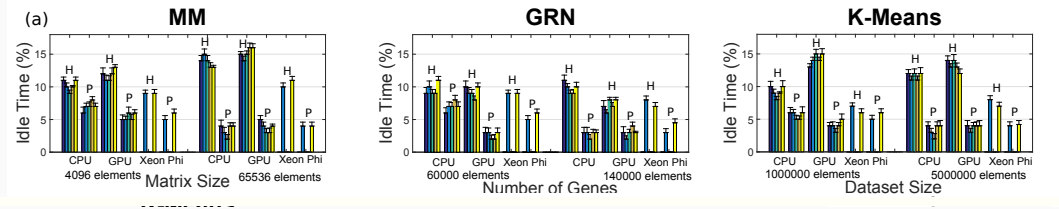


The block size ratio distributed to each PU in the five machines (A to E) for:
Optimal (O), HDSS (H) and PLB-HAC (P) distributions
CPU, GPU and Xeon Phi accelerators.

The distribution of blocks sizes generated by PLB-HAC was very similar to the optimal.

HDSS had a larger allocation to CPUs and smaller to Xeon Phi

Idleness



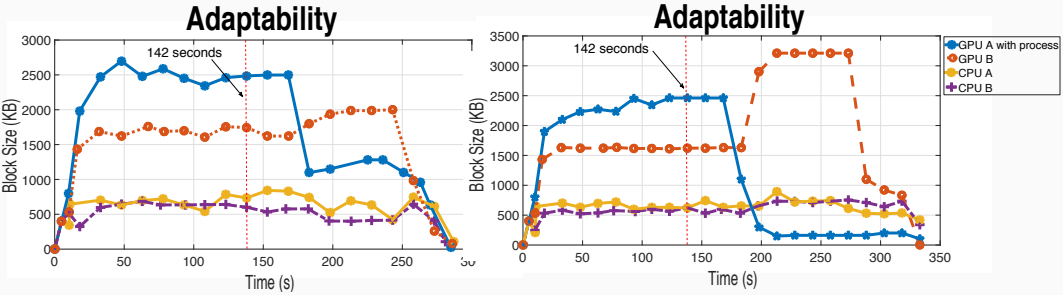
Percentage of idle time for each PU class (CPU, GPU, and Xeon Phi) in five machines for: HDSS (H) and PLB-HAC (P) distributions

Two different input sizes for the applications

HDSS had much higher idle times, which occurred mostly in the initial phase, during model construction

UFABC

Adaptability



Evolution of the block size distribution for two machines (A and B) executing the Matrix Multiplication application in the presence of a competing process

(Left) GRN application is started at GPU A at instant 142s, denoted by the vertical line.

(Right) A resource intensive rendering application is started at GPU A at 142s

Conclusions

PLB-HAC: Profile-based Load-Balancing for Heterogeneous Accelerator Clusters

Online performance modeling and precise block size selection by solving a non-linear system of equations

Improved execution time, specially with more heterogeneous clusters

Future work:

- Fault-tolerance: execution could continue, with a new block distribution using the performance model
- Overlapping communication with computations, as application knows in advance the size of the next blocks to each PU
- Scalability: evaluate the use of applications executing on hundreds of PUs. But most of them would be similar.
- Multiple kernels: extend to applications that have multiple kernels and execution phases

PLB-HAC: Dynamic Load-Balancing for Heterogeneous Accelerator Clusters

Göttingen, Germany

Luis Sant'Ana[1] Daniel Cordeiro[2] **Raphael Y. de Camargo**[1]

raphael.camargo@ufabc.edu.br

[1]Universidade Federal do ABC, Brazil

[2]Universidade de São Paulo, Brazil

EURO-PAR CONFERENCE 2019

FAPESP

CAPES

UFABC