

Optimizing the Performance of Multi-threaded Linear Algebra Libraries, a Task Granularity based Approach

PhD Proposal

Shahrzad Shirzad

December 4, 2019

Division of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Louisiana State University

Outline

Objective

Introduction

Background

Method

Related Work

Proposed Study

Objective

Objective

A compile-time and runtime solution to optimize the performance of a linear algebra library based on

- Machine architecture
- Number of cores to run the program on
- Expression to be evaluated
 - Type of operations
 - Number of matrices involved
 - Matrix sizes

Introduction

Introduction

- Current programming models would not be able to keep up with the advances toward exascale computing
 - More complex machine architectures, deeper memory hierarchies, heterogeneous nodes, complicated networks
- AMT(Asynchronous Many-Task) model and runtime systems
 - Examples: HPX, Charm++, Legion

Introduction

- Performance of HPC applications heavily rely on the linear algebra library they are using.
- Linear algebra libraries
 - BLAS(Basic Linear Algebra Subprograms) are the fundamental routines for basic vector and matrix operations.
 - Examples: ScaLAPACK, ATLAS, SPIRAL
- Our motivation:
 - Phylanx, a platform to run your python code in parallel and distributed with machine learning as the target application.

Background

Background: HPX

- HPX is a general purpose C++ runtime system for parallel and distributed applications of any scale.
- HPX is the first open source software runtime system implementing the concepts of the ParalleX execution model, on conventional systems including Linux clusters, Windows, Macintosh, Android, XeonPhi, and the Bluegene/Q.
- Fine-grained parallelism instead of heavyweight threads.

Background: HPX, Execution Model

Four major factors for performance degradation: SLOW

- Starvation
- Latency
- Overheads
- Waiting for contention resolution

Background: Blaze C++ Library



Blaze is a high performance C++ linear algebra library based on Smart Expression Templates.

- Expression Templates:
 - Creates a parse tree of the expression at compile time and postpone the actual evaluation to when the expression is assigned to a target
- Smart:
 - Integration with highly optimized compute kernels
 - Selecting optimal evaluation method automatically for compound expressions

Background: Blaze, Parallelization

Depending on the operation and the size of operands, the assignment could be parallelized through four different backends

- HPX
- OpenMP
- C++ threads
- Boost

Background: Blaze, Backend Implementation

In the current implementation, the work is equally divided between the cores at compile time.

- Parallel for loop

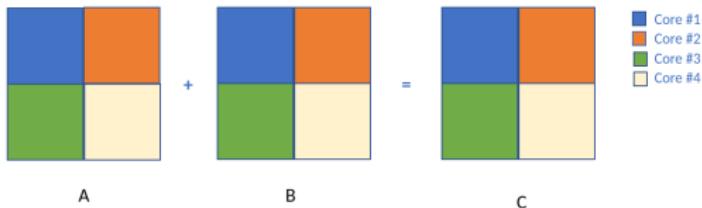


Figure 1: An example of how $C=A+B$ is performed in parallel in Blaze with 4 cores

Background: Loop Scheduling

- Chunk size: Number of loop iterations executed by one thread

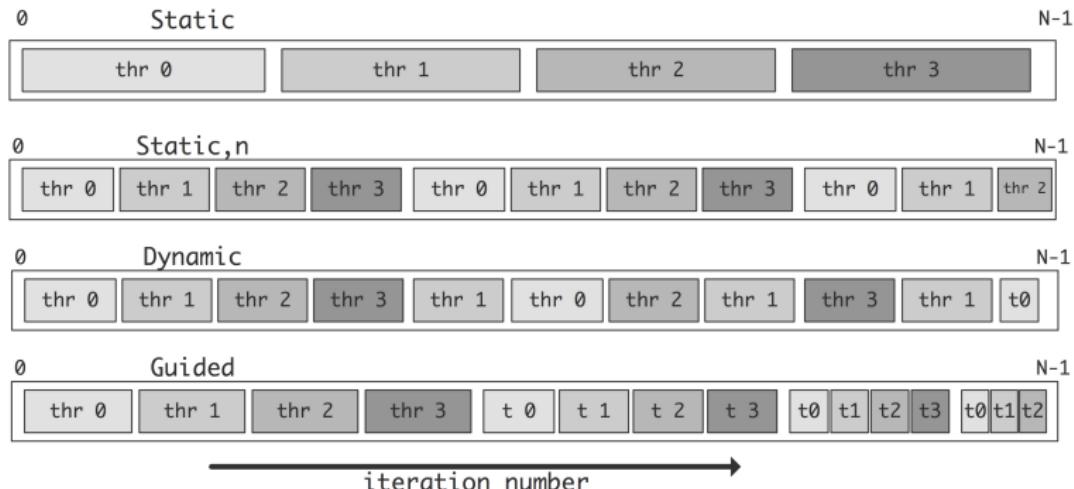


Figure 2: An example of different loop scheduling methods¹

¹<http://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-loop.html>

Background: Task Granularity

Grain size: The amount of work performed by one task

- What causes performance degradation?
 - Overheads
 - Starvation

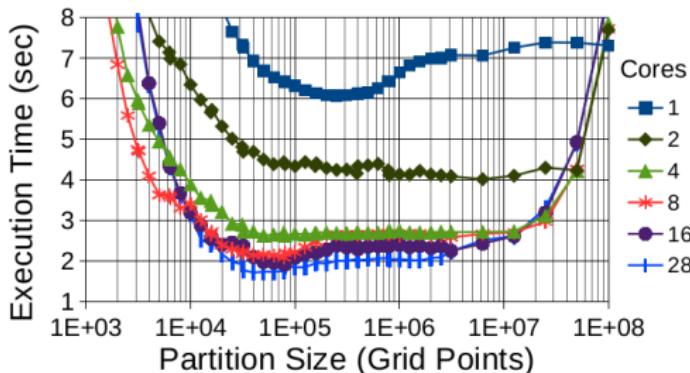


Figure 3: The effect of task size on execution time for Stencil application²

²Grubel, Patricia, et al. "The performance implication of task size for applications on the hpx runtime system." 2015 IEEE International Conference on Cluster Computing. IEEE, 2015.

Background: Modeling Performance based on number of cores

- Amdahl's Law

$$S(p) = \frac{p}{1 + \sigma(p - 1)}$$

- Universal Scalability Law(USL)

$$S(p) = \frac{p}{1 + \sigma(p - 1) + \kappa p(p - 1)}$$

- Models the effects of linear speedup, contention delay, and coherency delay due to crosstalk

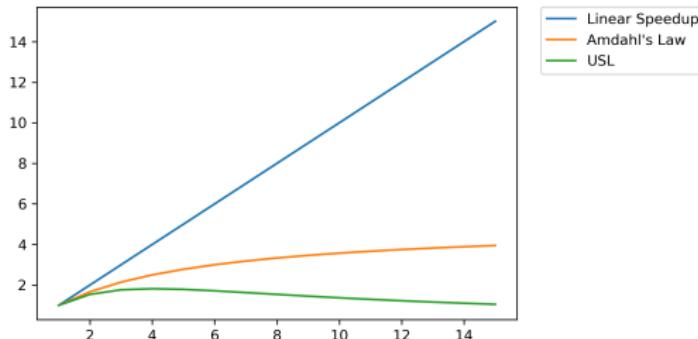


Figure 4: An example of speedup based on Amdahl's law and USL compared to the ideal linear speedup where $\sigma = 0.2$ and $\kappa = 0.05$.

Background: Modeling Performance based on number of cores: Other Models

- Quadratic model

$$S(p) = p - \gamma p(p - 1)$$

- Exponential model

$$S(p) = p(1 - \alpha)^{(p-1)}$$

- Geometric model

$$S(p) = \frac{1 - \phi^p}{1 - \phi}$$

Method

Method: Objective

Dynamically divide the work among the cores based on number of cores, matrix size, complexity of the operation, machine architecture. For this purpose two parameters have been introduced:

- `block_size`: at each loop iteration the assignment is performed on one block
- `chunk_size`: the number of loop iterations included in one task

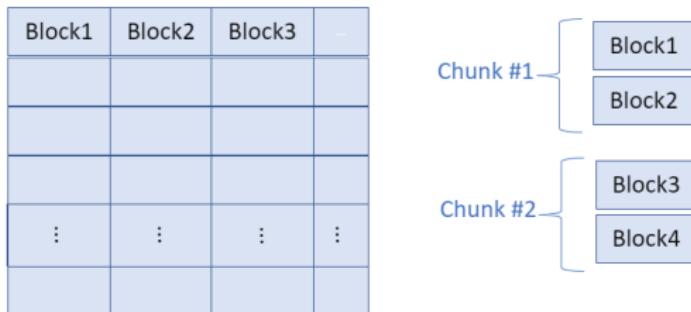


Figure 5: An example of blocking a matrix and creating chunks for `chunk_size = 2`

Method: Data Collection

- Starting from DMATDMATADD benchmark: $C = A + B$

Category	Configuration
Matrix sizes	200, 230, 264, 300, 396, 455, 523, 600, 690, 793, 912, 1048, 1200, 1380, 1587
Number of cores	1, 2, 3, 4, 5, 6, 7, 8
Number of rows in the block	4, 8, 12, 16, 20, 32
Number of columns in the block	64, 128, 256, 512, 1024
Chunk size	Between 1 and total number of blocks (logarithmic increase)

Table 1: List of different values used for each variable for running the *DMATDMATADD* benchmark

Method: Data Analysis

- For simplicity we look at each matrix size individually, one number of core at a time

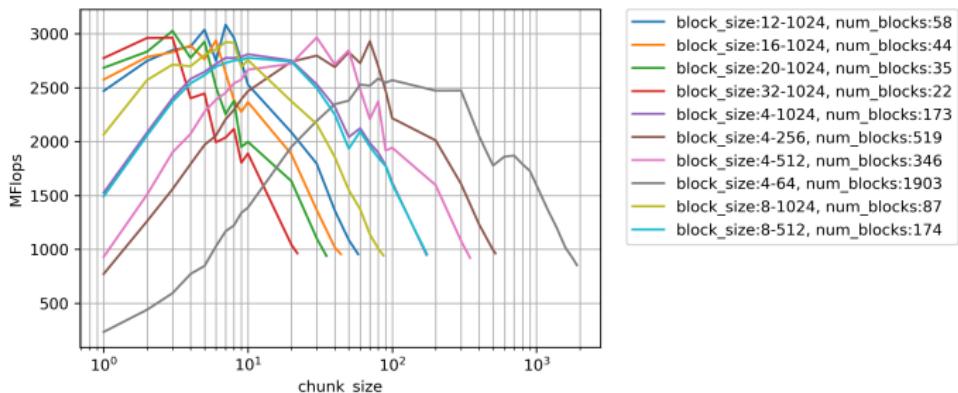
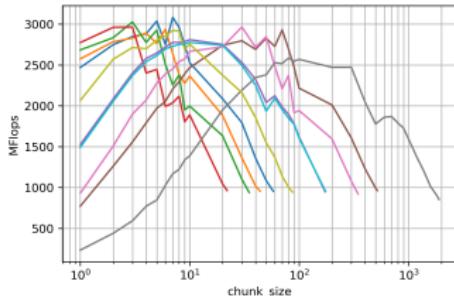


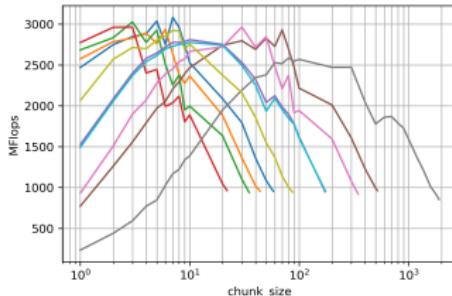
Figure 6: The results obtained from running *DMATDMATADD* benchmark for matrix sizes 690×690 with different combinations of block size and chunk size on 4 cores

Method: Observation



- For each selected block size, there is a range of chunk sizes that gives us the best performance.
- Except for some uncommon cases, no matter which block size we choose, we are able to achieve the maximum performance if we select the right chunk size.

Method: Observation

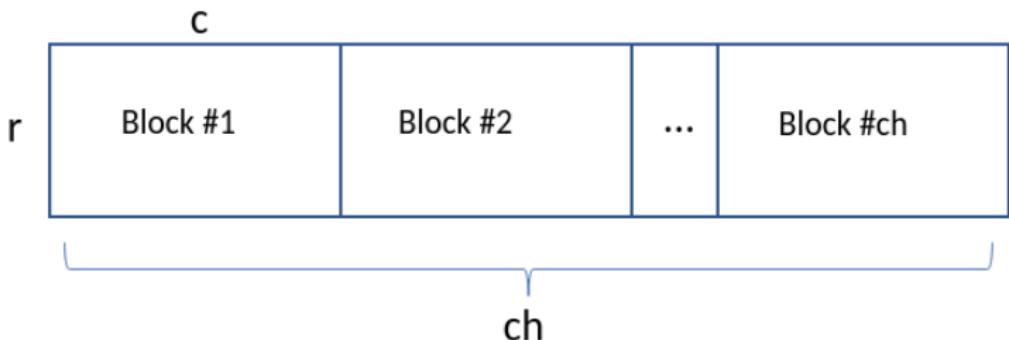


- For each selected block size, there is a range of chunk sizes that gives us the best performance.
- Except for some uncommon cases, no matter which block size we choose, we are able to achieve the maximum performance if we select the right chunk size.
- Instead of looking at block size and chunk size individually, look at grain size.

Method: Throughput vs. Grain Size

Grain size: The number of floating point operations performed by one thread

- Grain size represents the complexity of the expression
- For *DMATDMATADD*, with $block_size = r \times c$ and $chunk_size = ch$
 $Grain_size = r \times c \times ch$



Method: Throughput vs. Grain Size

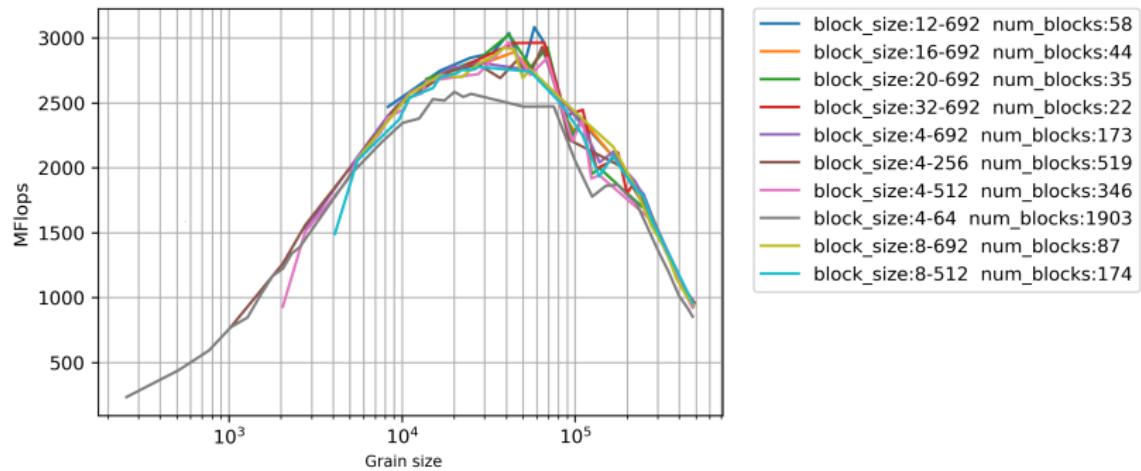


Figure 7: The results obtained from running *DMATDMATADD* benchmark through Blazemark for matrix size 690×690 on 4 cores.

Method: Throughput vs. Grain Size

The range of grain size for maximum performance

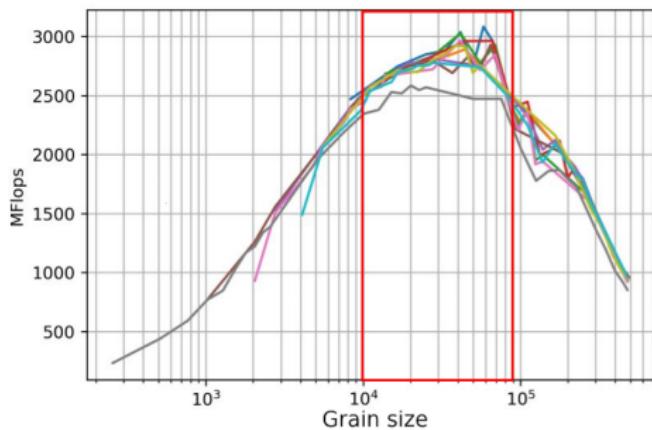


Figure 8: The results obtained from running *DMATDMATADD* benchmark through Blazemark for matrix size 690×690 on 4 cores.

Throughput vs. Grain Size and Number of Cores

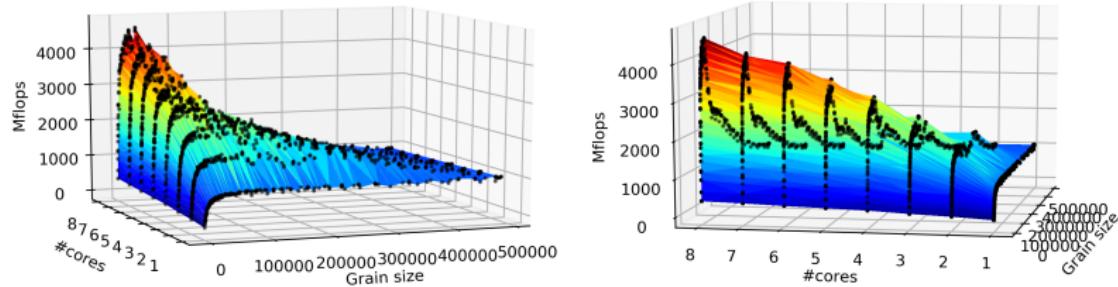
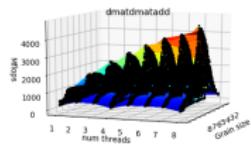


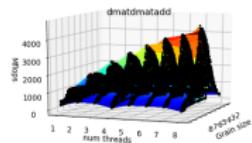
Figure 9: The results obtained from running *DMATDMATADD* benchmark through Blazemark for matrix size 690×690 based on grain size and number of cores.

Throughput vs. Grain Size and Number of Cores



Can we model the relationship between the **throughput** and the **grain size** and the **number of cores**?

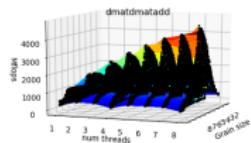
Throughput vs. Grain Size and Number of Cores



Can we model the relationship between the **throughput** and the **grain size** and the **number of cores**?

- First we try to model the relationship between throughput and grain size.

Throughput vs. Grain Size and Number of Cores



Can we model the relationship between the **throughput** and the **grain size** and the **number of cores**?

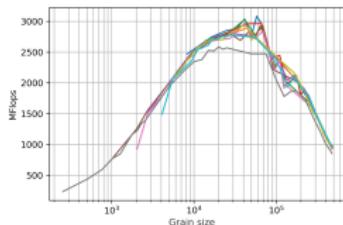
- First we try to model the relationship between throughput and grain size.
 - Polynomial Model
 - Bathtub Model

Method: Polynomial Model

In order to simplify the process and eliminate the effect of different possible factors, we started with limiting the problem to a fixed matrix size.

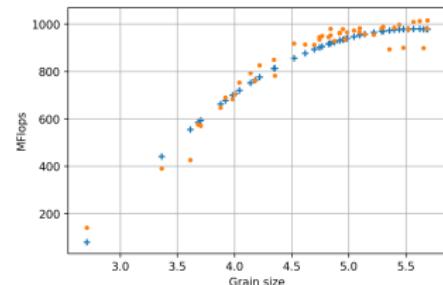
- Used a second order polynomial to model the relationship between the throughput and the grain size when number of cores is fixed.

$$P = ag^2 + bg + c$$

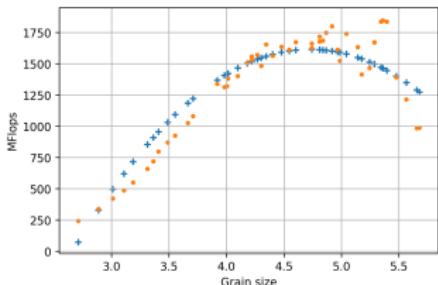


- Divide the data into training(60%) and test(40%)

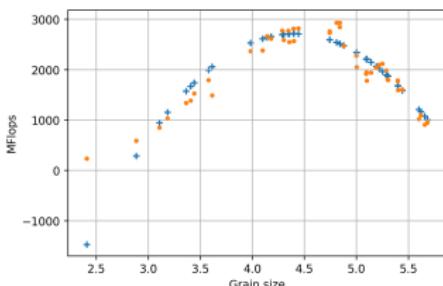
Method: Modeling Performance based on Grain Size



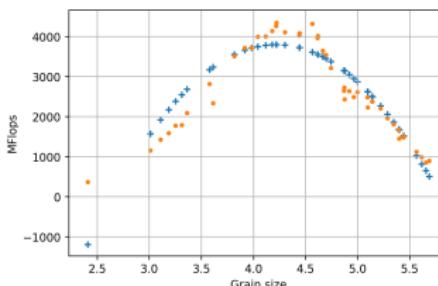
(a) 1 core



(b) 2 cores



(c) 4 cores



(d) 8 cores

Figure 10: The results of fitting the throughput vs grain size data into a 2d polynomial for *DMATDMATADD* benchmark for matrix size 690×690 with different number of cores on the test data.

Method: Modeling Performance based on Grain Size

$$\text{Mean_absolute_error} = \frac{1}{n} \sum_{i=1}^n \text{abs}(t_i - p_i)$$

n is the number of samples

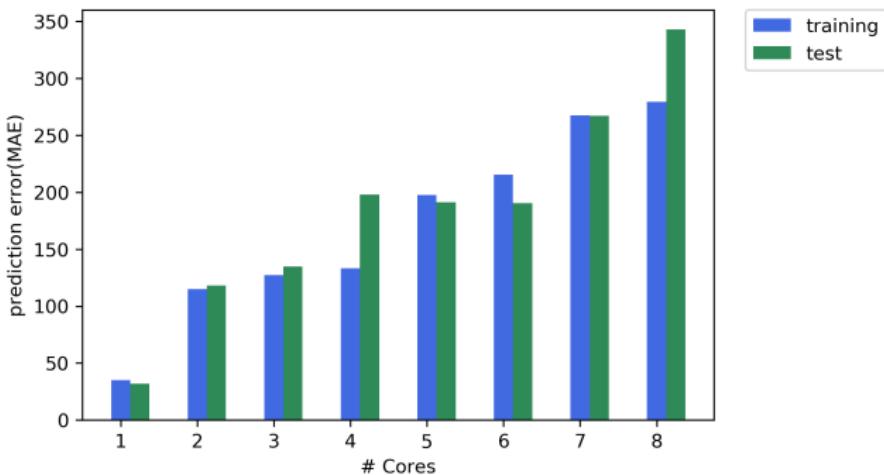


Figure 11: The training and test error for fitting data obtained from the *DMATDMATADD* benchmark for matrix size 690×690 against different number of cores cores.

Method: Modeling Performance based on Grain Size

$$R_{\text{_squared}} = 1 - \frac{\frac{1}{n} \sum_{i=1}^n (t_i - p_i)^2}{\text{Var}(t)}$$

n is the number of samples

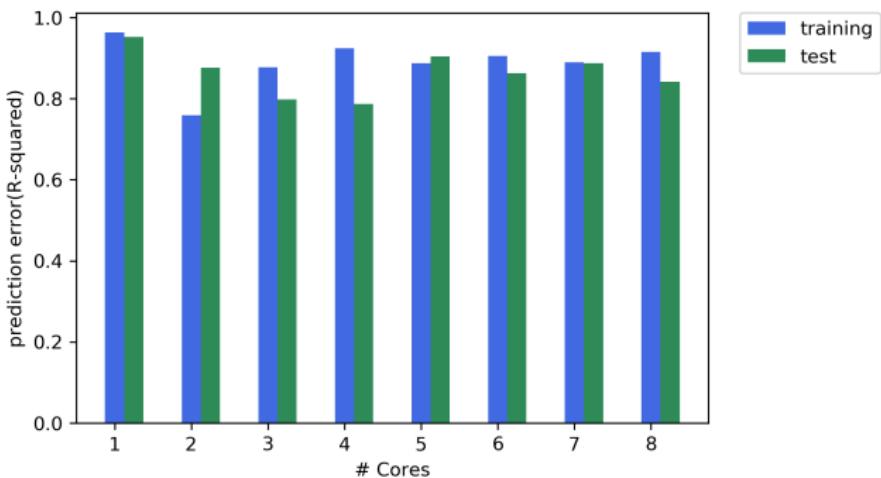
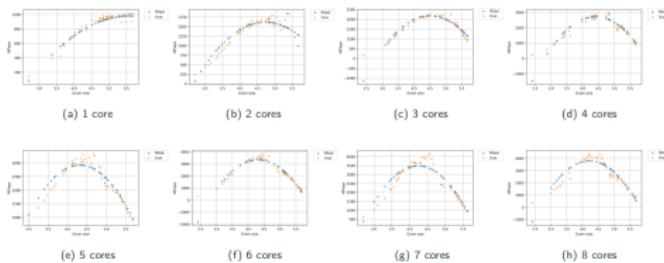


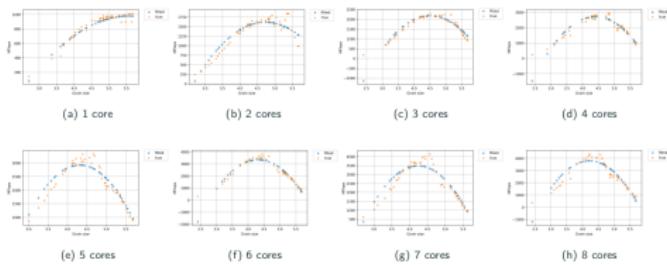
Figure 12: The training and test error for fitting data obtained from the *DMATDMATADD* benchmark for matrix size 690×690 against different number of cores cores.

Method: Modeling Performance based on Grain Size



- We have developed a model for each number of cores: 1, 2, .., 8
- $$P = ag^2 + bg + c$$

Method: Modeling Performance based on Grain Size



- We have developed a model for each number of cores: 1, 2, .., 8
$$P = ag^2 + bg + c$$
- Can we somehow integrate number of cores into the model?

Method: Modeling Performance based on Grain Size

- For $P = ag^2 + bg + c$, how do a , b , and c change with the number of cores?

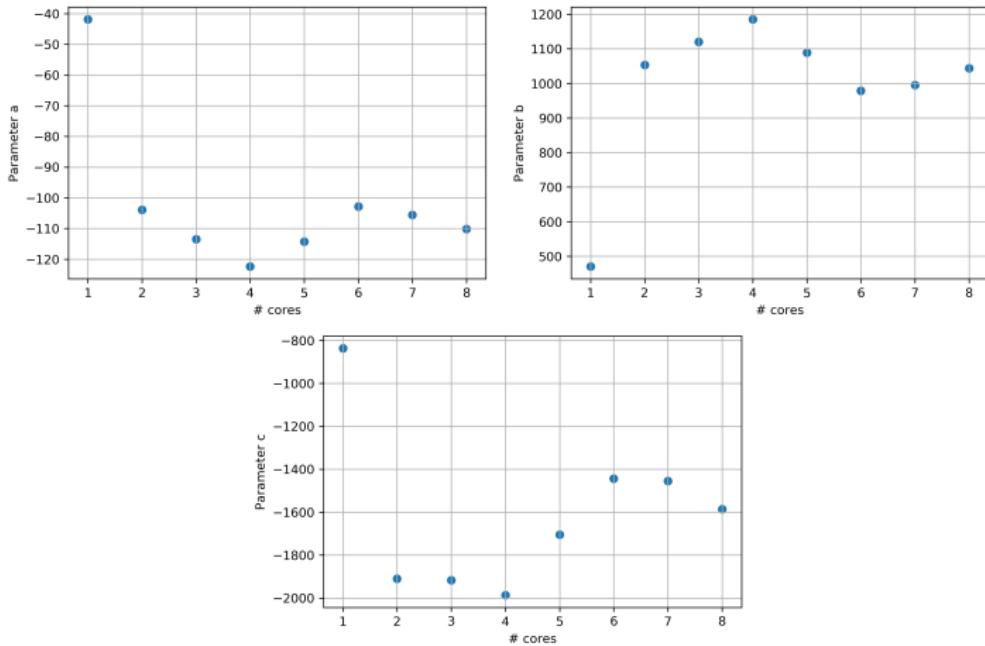


Figure 13: The parameters of the polynomial fit from the *DMATDMATADD* benchmark for matrix size 1587×1587 against different number of cores.

Method: Modeling Performance based on Grain Size

- Model the relationship with a 3rd degree polynomial

$$a = a_0 N^3 + a_1 N^2 + a_2 N + a_3, \quad b = b_0 N^3 + b_1 N^2 + b_2 N + b_3,$$

$$c = c_0 N^3 + c_1 N^2 + c_2 N + c_3$$

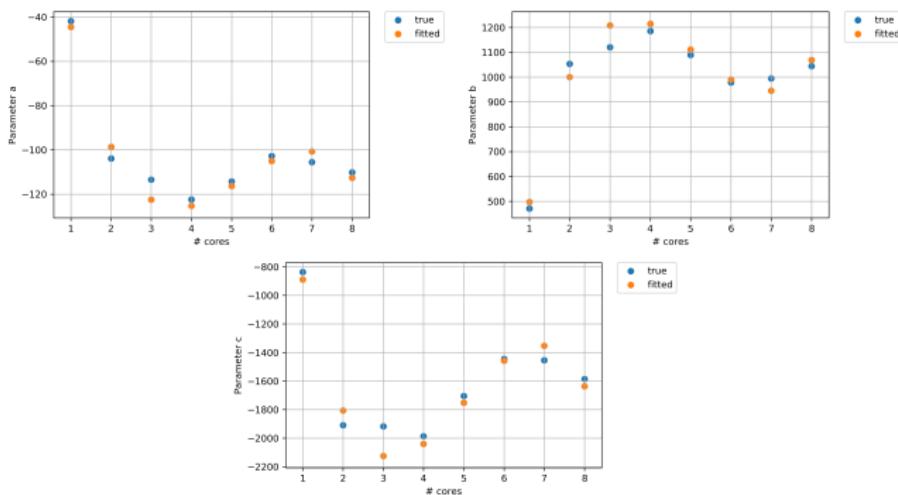


Figure 14: Fitting the parameters of the polynomial function with a 3rd degree polynomial from the *DMATDMATADD* benchmark for matrix size 1587×1587 against different number of cores.

Method: Modeling Performance based on Grain Size

The final model:

$$P = a_{11}g^2N^3 + a_{10}g^2N^2 + \dots + a_1N + a_0$$

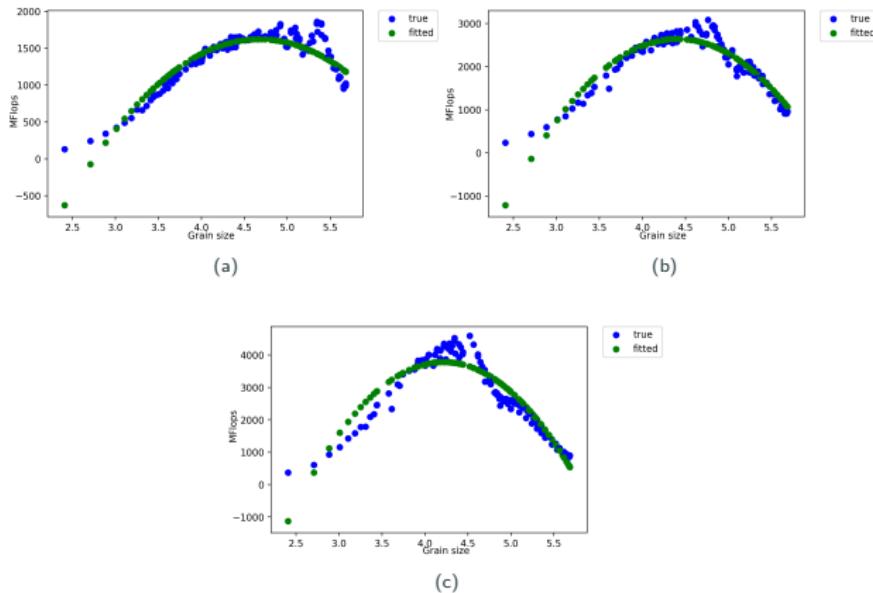
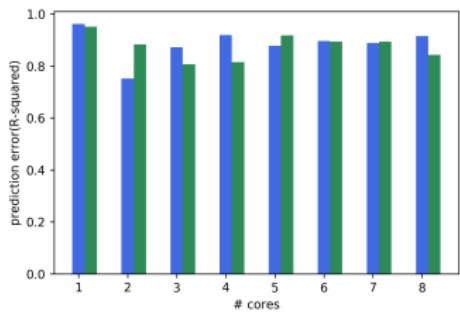
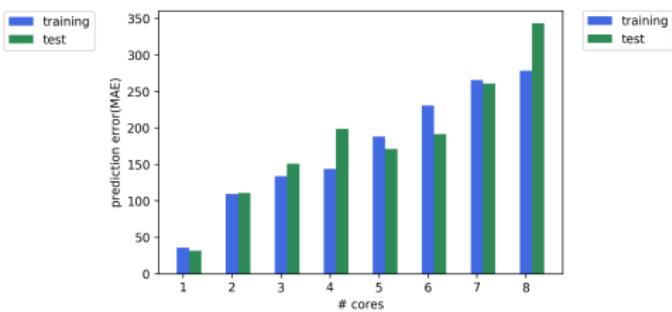


Figure 15: matrix size 690×690 for (a) 2 core, (b) 4 cores, (c) 8 cores.

Method: Modeling Performance based on Grain Size



(a)



(b)

Figure 16: All the data points are included in calculation of error,(a) R^2 squared error (b) Mean Absolute Error(MAE) .

Method: Finding the Grain Size Range for Maximum Performance

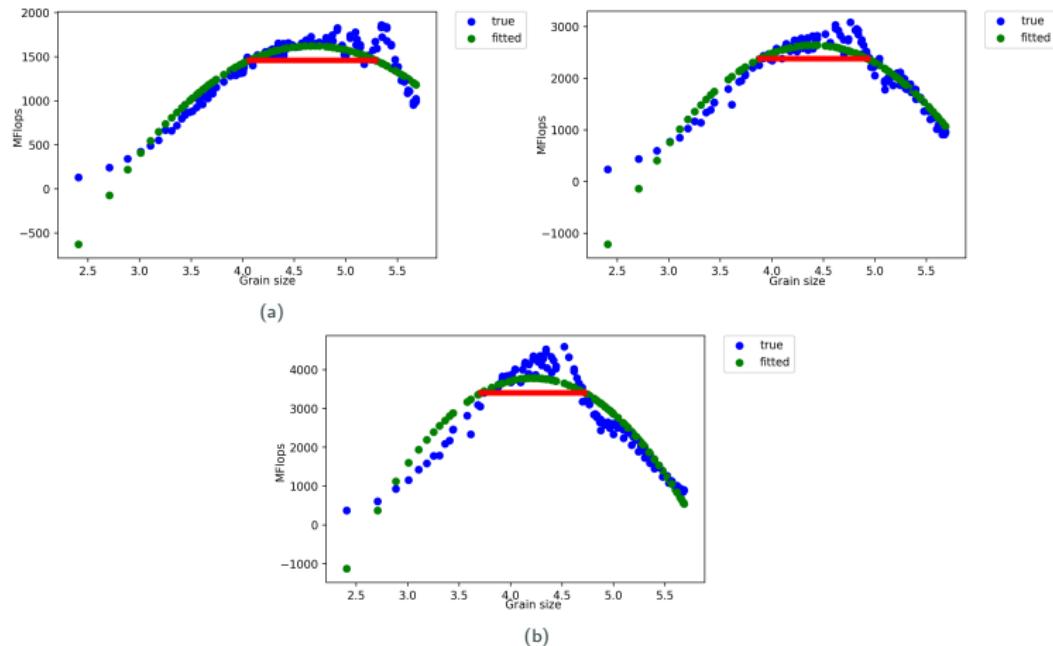
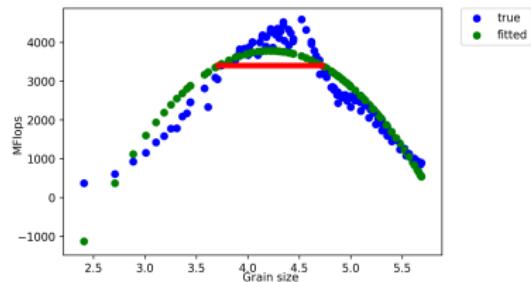


Figure 17: The range of grain size (shown as the red line) that leads to a performance within 10% of the maximum performance for (a) 2 cores, (b) 4 cores and (b) 8 cores.

Method: Finding the Grain Size Range for Maximum Performance



How do we use the calculated range?

- Select a reasonable block size, e.g. 4×256
- Find the range of chunk size that results in the calculated range of grain size, $\text{Grain_size} = r \times c \times ch$

Method: Finding the Grain Size Range for Maximum Performance

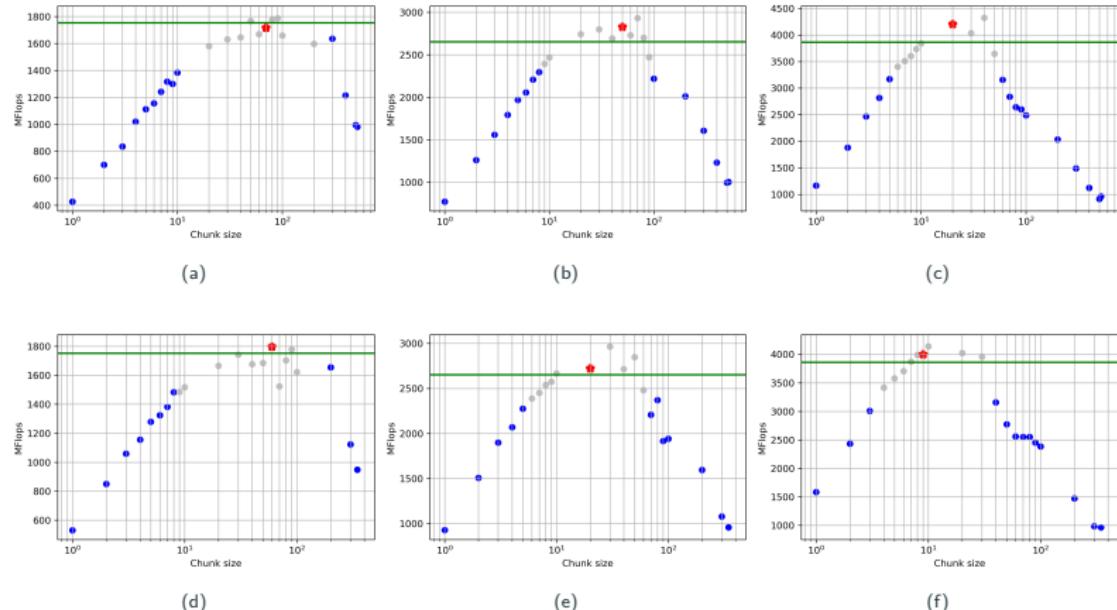


Figure 18: matrix size 690×690 with block size of 4×256 on (a) 2 cores, (b) 4 cores, and (c) 8 cores, and block size of 4×512 on (d) 2 cores, (e) 4 cores, and (f) 8 cores.

Method: Polynomial Model, Wrap up

- Simple model, can easily find the maximum.
- It is not physical.

Method: Bathtub Model

Can we create an analytic model for execution time based on grain size?

Method: Bathtub Model

- Overheads of creating tasks
- Starvation

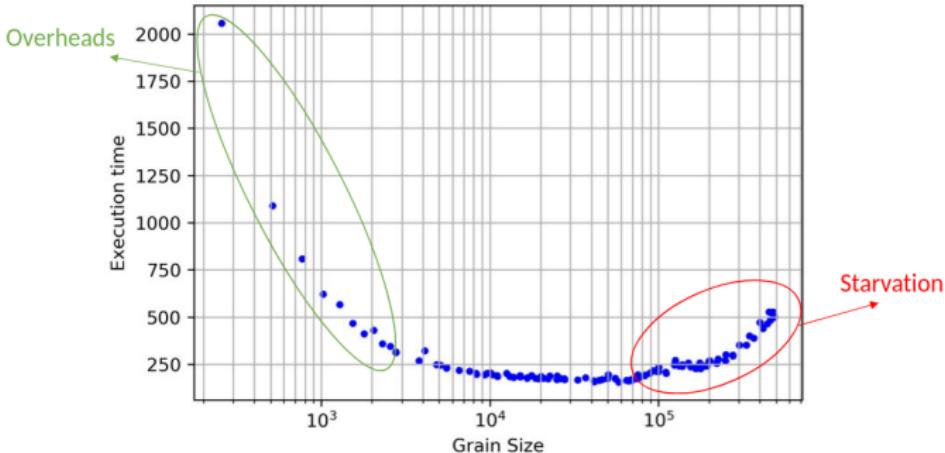


Figure 19: Results of running the *DMATDMATADD* benchmark on 8 cores matrix size 690×690 (time unit is microseconds)

Method: Modeling Execution Time based on Grain Size

N : Number of cores

n_t : Number of created tasks

$$n_t = \frac{\text{Total amount of work}}{\text{grain_size}}$$

t_s : sequential execution time

M : Number of cores actually doing the work

$$M = \begin{cases} n_t & \text{if } n_t < N \\ N & \text{otherwise} \end{cases}$$

Method: Modeling Execution Time based on Grain Size

N : Number of cores

n_t : Number of created tasks

$$n_t = \frac{\text{Total amount of work}}{\text{grain_size}}$$

t_s : sequential execution time

M : Number of cores actually doing the work

$$M = \begin{cases} n_t & \text{if } n_t < N \\ N & \text{otherwise} \end{cases}$$

$$\text{Execution_time} = \frac{t_s}{M}$$

Method: Modeling Execution Time based on Grain Size

N : Number of cores

n_t : Number of created tasks

$$n_t = \frac{\text{Total amount of work}}{\text{grain_size}}$$

t_s : sequential execution time

M : Number of cores actually doing the work

$$M = \begin{cases} n_t & \text{if } n_t < N \\ N & \text{otherwise} \end{cases}$$

$$\text{Execution_time} = \frac{t_s}{M} + \alpha \frac{n_t}{M}$$

Method: Modeling Execution Time based on Grain Size

N : Number of cores

n_t : Number of created tasks

$$n_t = \frac{\text{Total amount of work}}{\text{grain_size}}$$

t_s : sequential execution time

M : Number of cores actually doing the work

$$M = \begin{cases} n_t & \text{if } n_t < N \\ N & \text{otherwise} \end{cases}$$

$$\text{Execution_time} = \frac{t_s}{M} + \alpha \frac{n_t}{M} + \gamma$$

Method: Modeling Execution Time based on Grain Size

$$t = \begin{cases} \alpha + \frac{t_s}{n_t} + \gamma & \text{if } n_t < N \\ \frac{\alpha n_t + t_s}{N} + \gamma & \text{otherwise} \end{cases}$$

Method: Modeling Execution Time based on Grain Size

- Fixed matrix size, and number of cores
- Training set and test set (%60, %40)

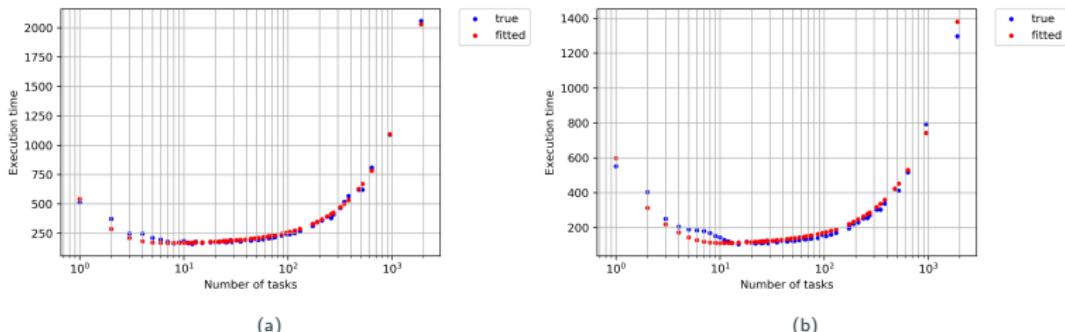


Figure 20: The prediction of execution time based on grain size using the bathtub model, for (a)4 cores and (b)8 cores for *DMATDMATADD* benchmark for matrix size 690×690 .

Method: Modeling Performance based on Grain Size

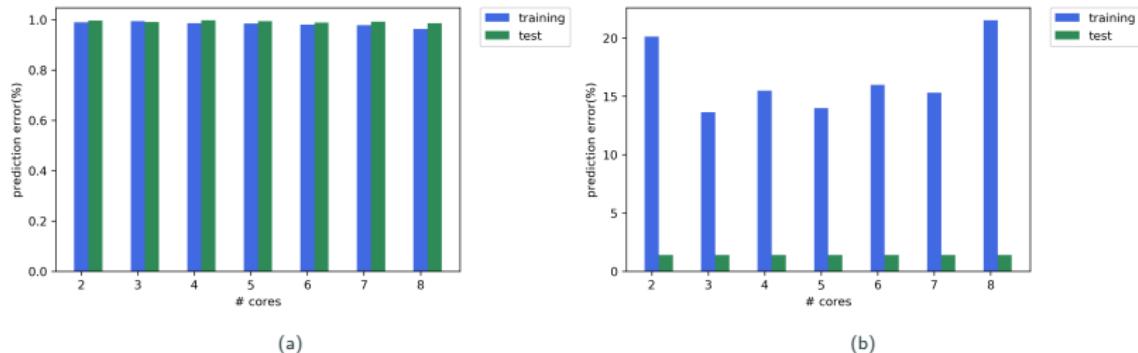


Figure 21: The error in fitting execution time with the bathtub formula for *DMATDMATADD* benchmark for matrix size 690×690 with different number of cores,(a) R_squared error (b) Mean Absolute Error(MAE).

Method: Modeling Execution Time based on Grain Size

- How do t_s , α , and γ change with number of cores?

$$f(N) = \frac{m_0 + m_1(N - 1) + m_2(N - 1)N + m_3(N^2)(N - 1)}{N}$$

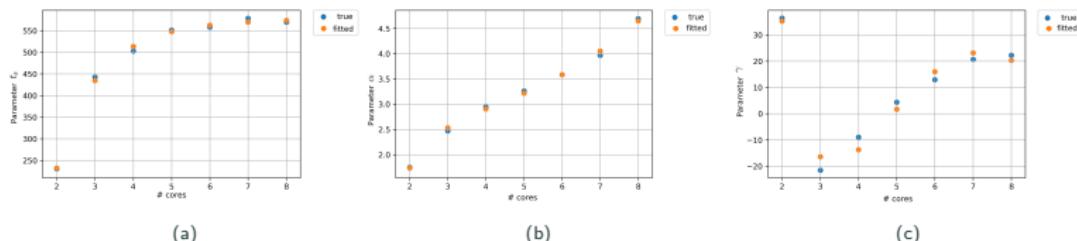


Figure 22: Fitting the three parameters (a) α , (b) t_s , and (c) γ for *DMATDMATADD* benchmark for matrix size 690×690 .

Method: Modeling Execution Time based on Grain Size

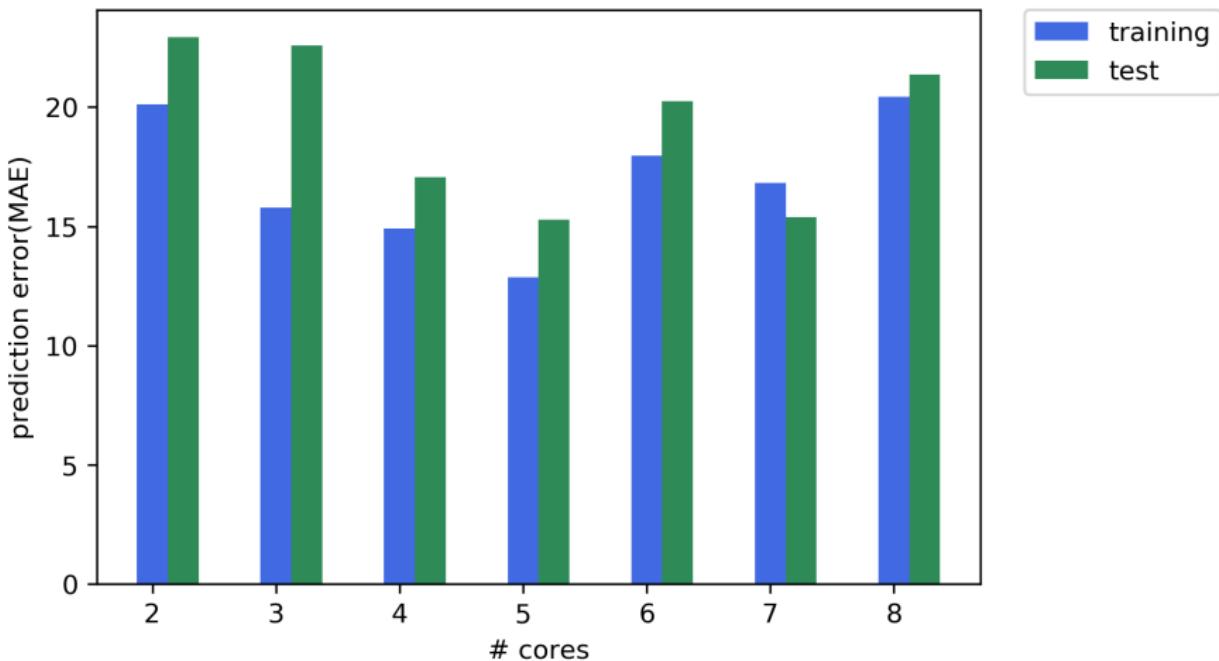


Figure 23: The error in fitting execution time with the bathtub formula integrated with the USL model for *DMATDMATADD* benchmark for matrix size 690×690 with different number of cores.

Method: Modeling Execution Time based on Grain Size

- The problem with the current model is that with this formula we know that the minimum occurs at $n_t = N$.
- Parameters t_s, α, γ do not behave the way we expect, they change with change of number of cores.
What is the missing factor?

Setup: Configuration

Category	Specification
CPU	2 x Intel(R) Xeon(R) CPU E5-2450 0 @ 2.10GHz
RAM	48 GB
Number of Cores	16
Hyperthreading	Off

Table 2: Specifications of the Marvin node from Rostam cluster at CCT.

Library	Version
HPX	1.3.0
Blaze	3.5

Table 3: Specifications of the libraries used to run our experiments.

Related Work

Related Work

- Liu et al. estimated the optimal number of cores to run the program on based on cache specific traces.
- Khatami et al. used logistic regression to find the best chunk size based on some static and dynamic features of the loop.
- Thoman et al. proposed a compile-time and runtime solution, using an effort estimation function set the chunk size.

Related Work

- Laberge et al. used machine learning to find the best chunk size to get the maximum performance, while block size was fixed statistically.
- Features included: matrix size, number of cores, number of floating point operations, number of iterations.

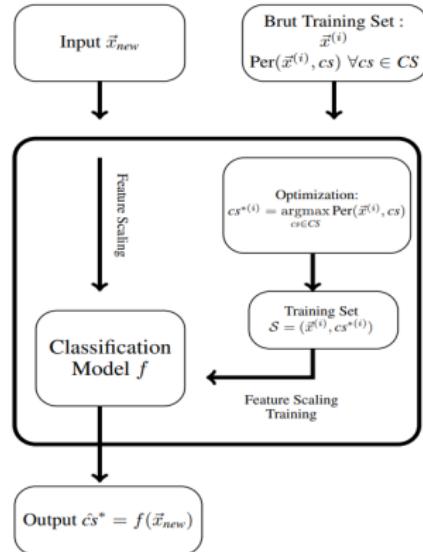


Figure 25: Laberge, G., Shirzad, S., Diehl, P., Kaiser, H., Prudhomme, S., Lemoine, A. (2019). Scheduling optimization of parallel linear algebra algorithms using Supervised Learning. arXiv preprint arXiv:1909.03947.

Our Contributions

- We propose a novel analytic model to represent how the execution time is expected to change based on grain size.
- To our knowledge, there has not been a work to create a 3D model of the throughput, grain size, and number of cores.
- We are proposing a method to apply the developed model to a linear algebra library, in a way specific to our application, and the machine architecture.

Proposed Study

Proposed Study

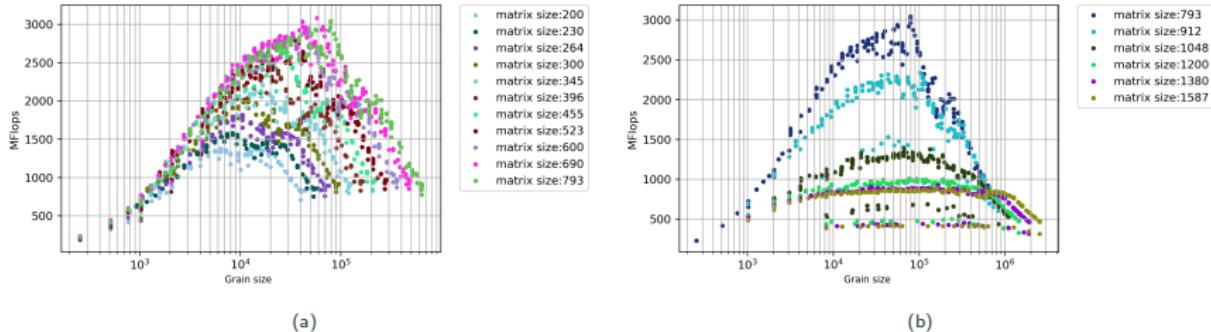


Figure 26: Throughput vs. grain size graph obtained from running *DMATDMATADD* benchmark on 4 cores for matrix sizes (a) smaller than 793×793 (b) larger than 793×793 .

- Studying the bathtub model to find the missing factor, also location of the minimum
- Generalization for matrix size, adding runs for larger matrix sizes
- Generalization for complex expressions
- Generalization for different architectures

Thank you!

Appendix

BLAS operations

Level 1	addition/scaling dot products, norms	$\alpha x, \quad \alpha x + y$ $x^T y, \quad \ x\ _2, \quad \ x\ _1$
Level 2	matrix/vector products rank 1 updates rank 2 updates triangular solves	$\alpha Ax + \beta y, \quad \alpha A^T x + \beta y$ $A + \alpha xy^T, \quad A + \alpha xx^T$ $A + \alpha xy^T + \alpha yx^T$ $\alpha T^{-1}x, \quad \alpha T^{-T}x$
Level 3	matrix/matrix products rank- k updates rank- $2k$ updates triangular solves	$\alpha AB + \beta C, \quad \alpha AB^T + \beta C$ $\alpha A^T B + \beta C, \quad \alpha A^T B^T + \beta C$ $\alpha AA^T + \beta C, \quad \alpha A^T A + \beta C$ $\alpha A^T B + \alpha B^T A + \beta C$ $\alpha T^{-1}C, \quad \alpha T^{-T}C$

Figure 27: <https://web.stanford.edu/class/ee392o/nlas-foils.pdf>

Appendix

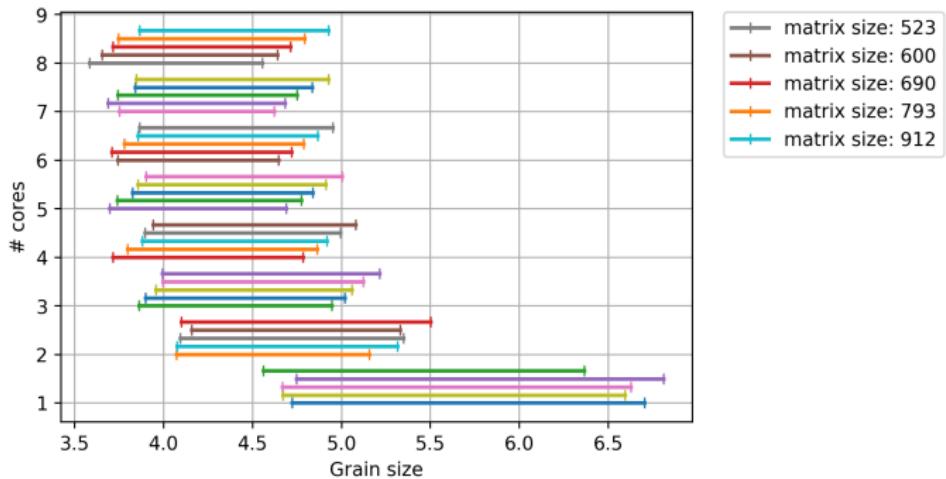


Figure 28: The range of grain size within 10% of the maximum performance of the fitted polynomial function for *DMATDMATADD* benchmark for different number of cores for matrix size 523×523 to 912×912 .