Auto Tuning Method for Deciding Block Size Parameters in Dynamically Load-Balanced BLAS

*Yuta SAWA and Reiji SUDA
The University of Tokyo

iWAPT 2009 October 1-2

*Now in Central Research Laboratory, Hitachi, Ltd.

Background
Multi-Core CPUs in personal computers

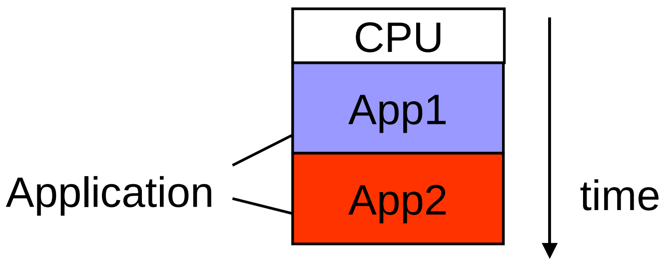
CPUs on personal computers

Year:	1995	2000	2005	2009
Fastest Intel CPU (for PC)	Pentium	Pentium 4	Pentium D	Core i7
Cores (thread)	1	1	2	4(8)
GFLOPS	0.4	4	12.8	51.2
How many dimensions of matrix multiplications can be solved in a second	580	1250	1800	2900

About 5 times in the dimension, and about 125 times in FLOPS

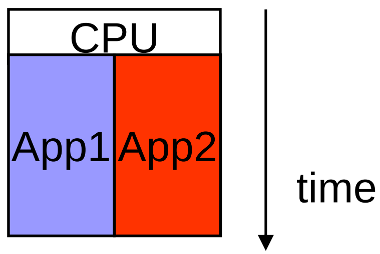
Users' paradigm shift about usage of computers

- A decade ago, users ran applications sequentially, or the performance of the application was decreased



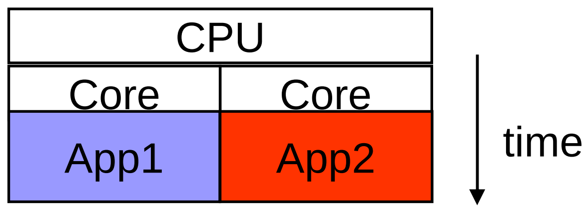
Running sequentially

or



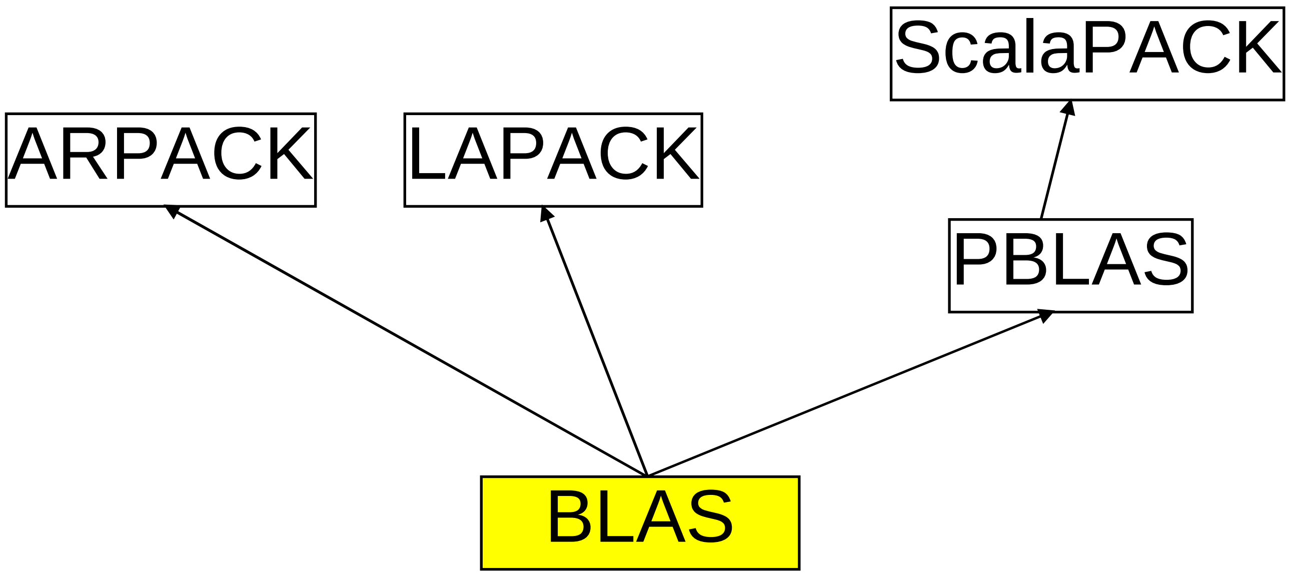
Running concurrently disturbing each other

- Today, users can run a few applications concurrently without much overhead



BLAS (Basic Linear Algebra Subprograms)

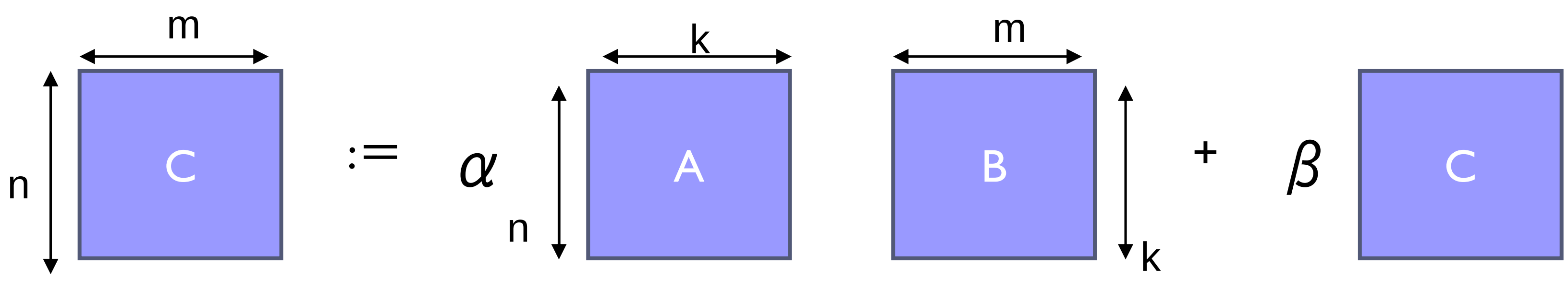
- BLAS are interfaces of multiplication routines of vectors and matrices
  - BLAS are basic routine in numerical calculations
  - BLAS are called from many other libraries such as LAPACK



DGEMM routine in BLAS

- DGEMM is the simplest double-precision matrix multiplication written as follows:

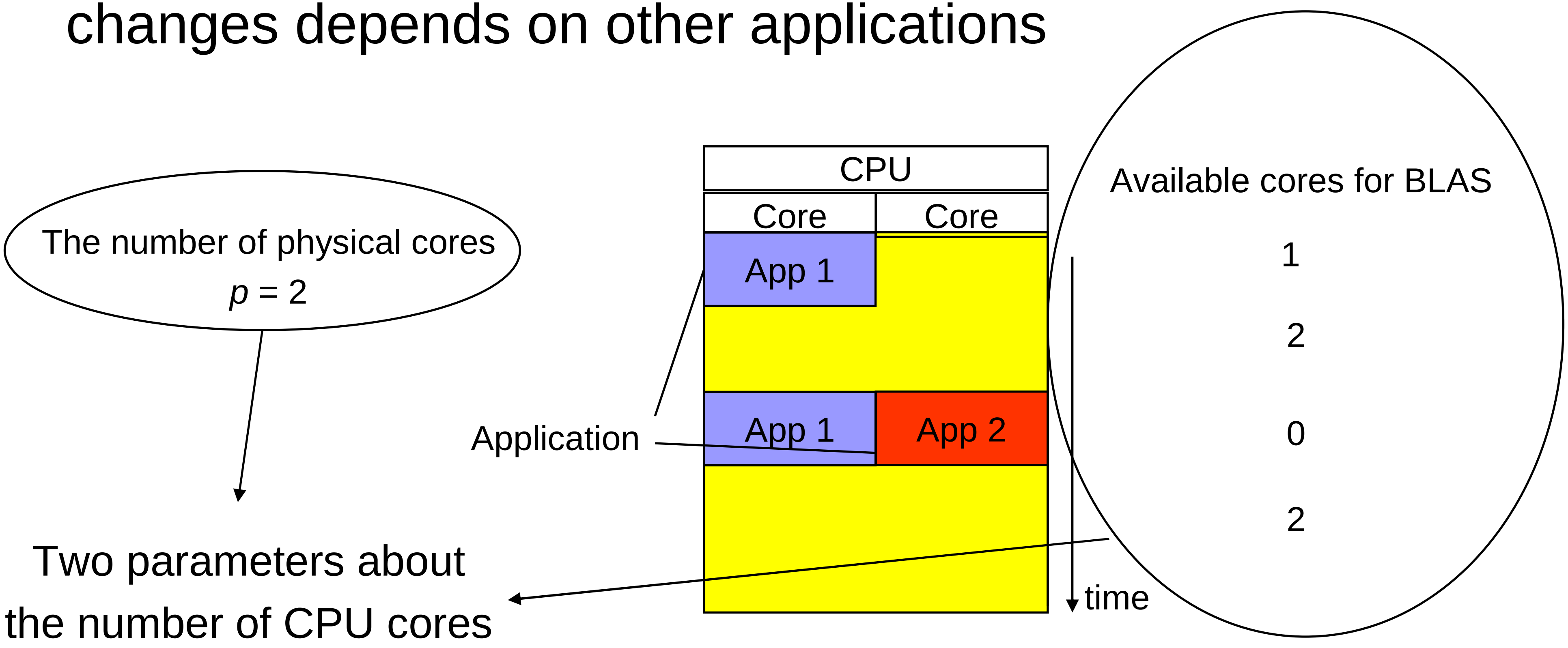
$$C := \alpha AB + \beta C$$



- Computation complexity of DGEMM routine is O(*mnk*)
- This DGEMM routine is our target problem in this paper

BLAS on personal computers

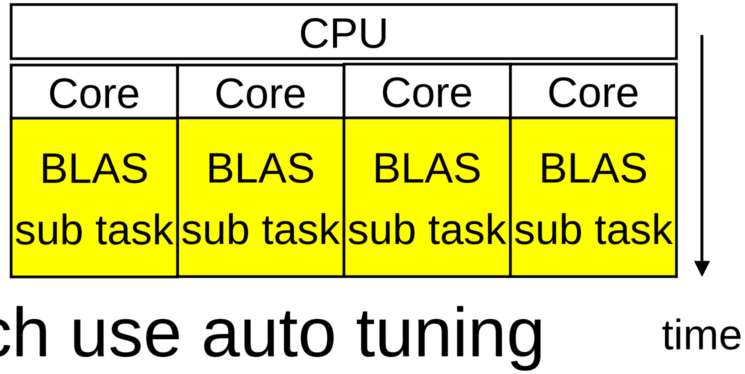
- DGEMM routine have much parallelism
- In personal computers, the number of available CPU cores changes depends on other applications



High-performance BLAS implementations

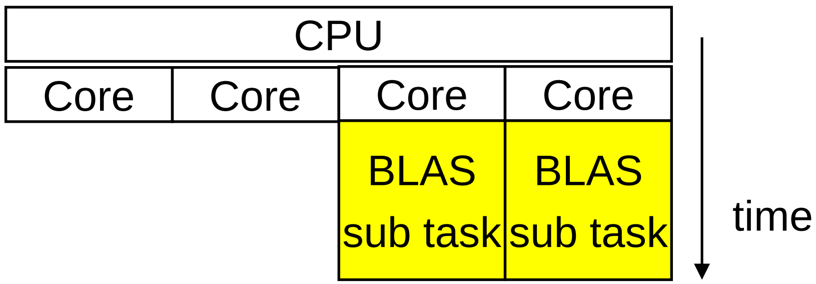
- GotoBLAS is the fastest BLAS implementation in the world today
  - Users can set the number of threads

GotoBLAS with NUM_THREAD=4

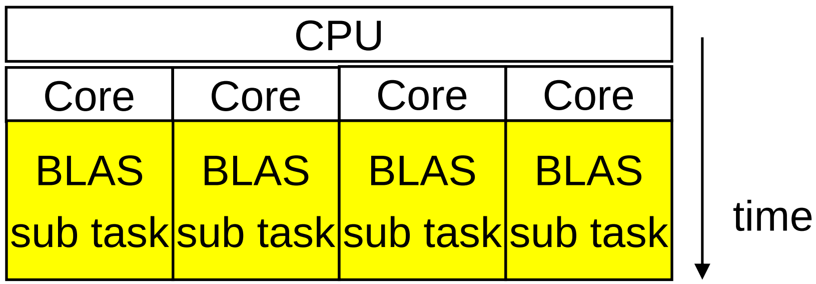


- ATLAS is BLAS implementation which use auto tuning
  - ATLAS decides the number of threads automatically depending on the problem size

ATLAS for small problems

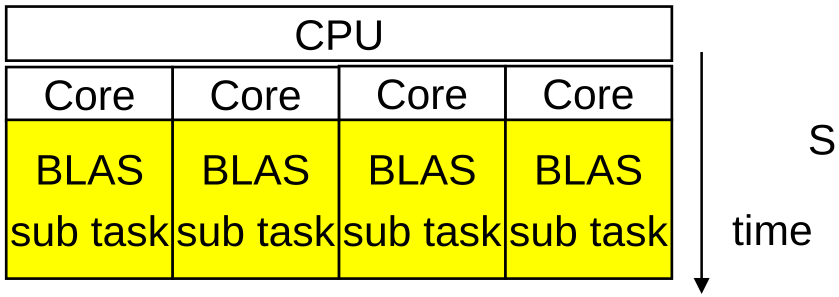


ATLAS for large problems



Importance of Dynamic Load Balancing

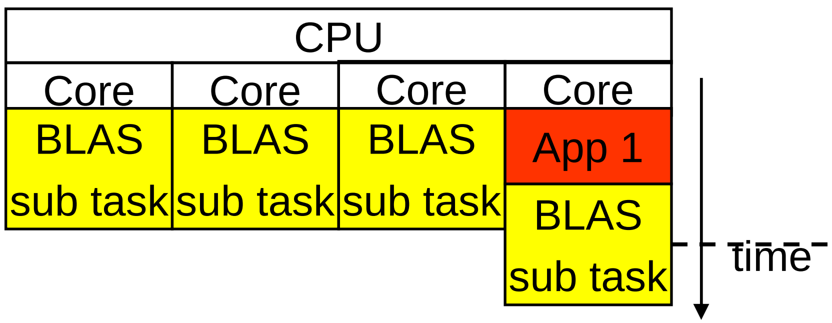
- In traditional BLAS implementations, the size of each sub task was almost the same



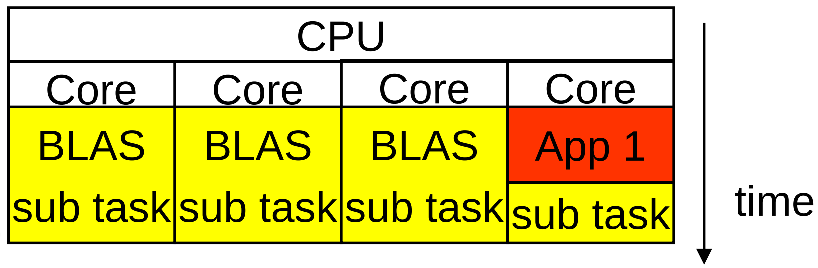
Sub tasks with the same size

- If there are other tasks running concurrently, using dynamic load balancing is better

Sub tasks with the same size



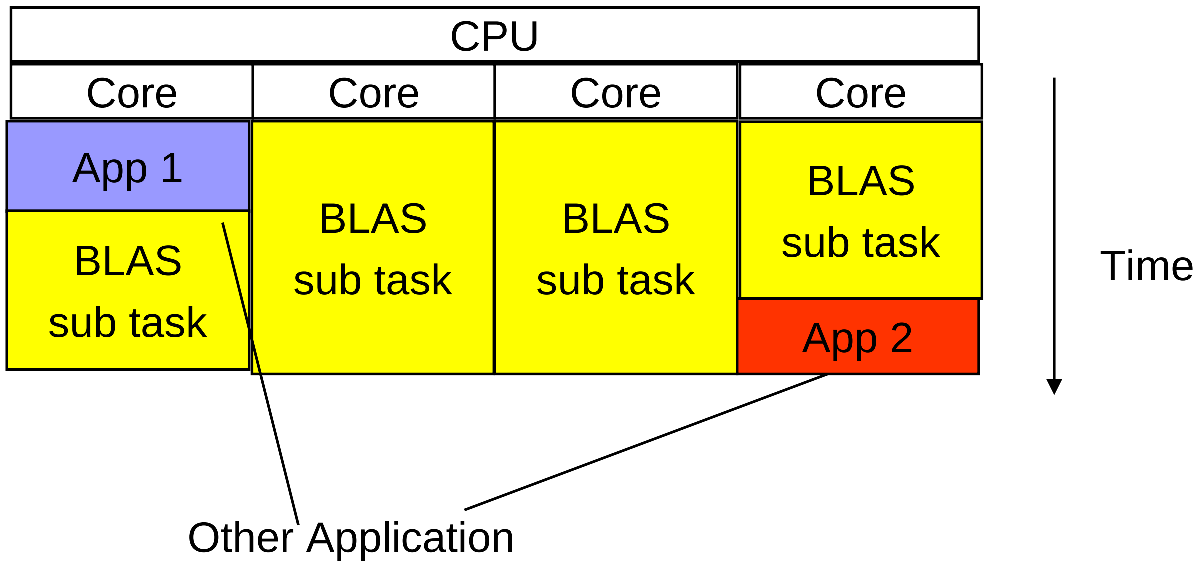
Dynamic load balancing



Dynamically Load-Balanced BLAS (DL-BLAS)

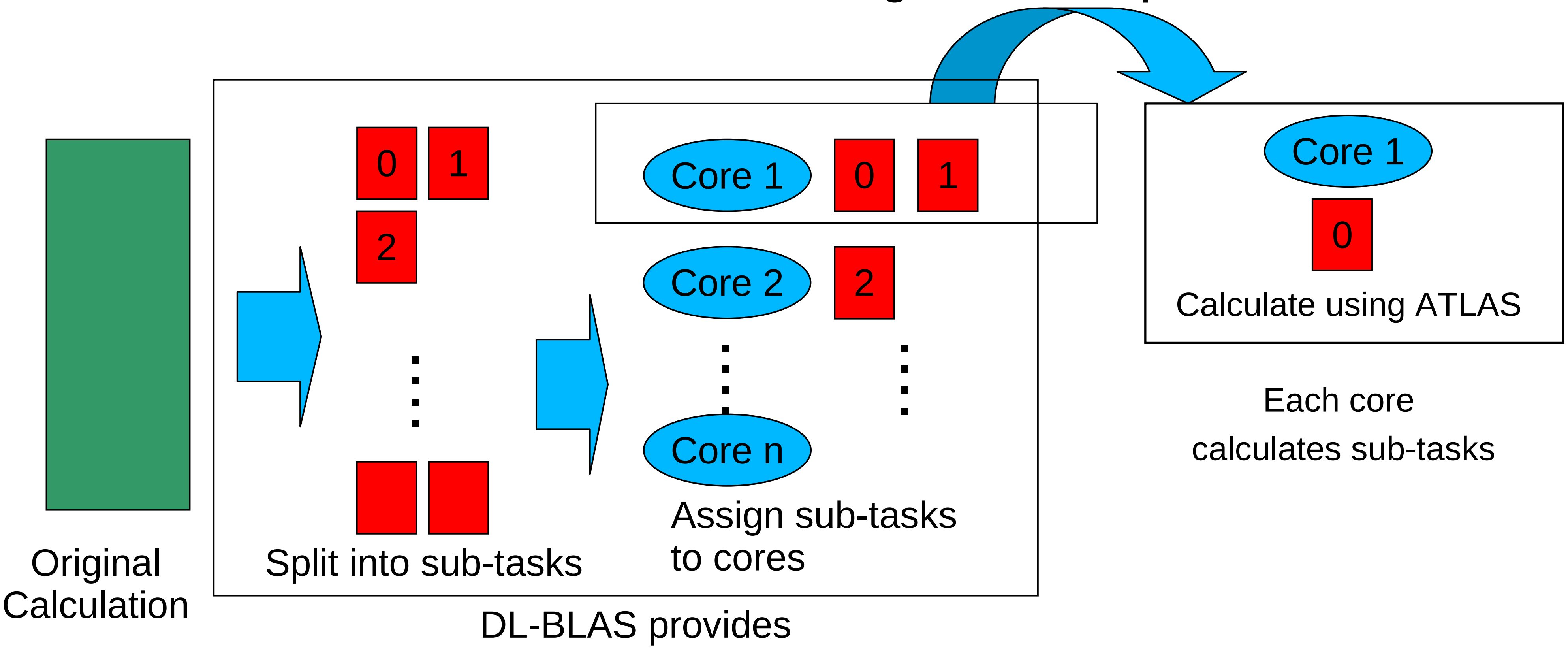
What is DL-BLAS?

- DL-BLAS is one of BLAS implementations
- DL-BLAS is well parallelized even if there are other applications running concurrently



DL-BLAS parallelization algorithm

- DL-BLAS split a calculation into some tasks, and assign them to CPUs using dynamic load balancing
- Each CPU calculates tasks using BLAS implementation



DL-BLAS overview

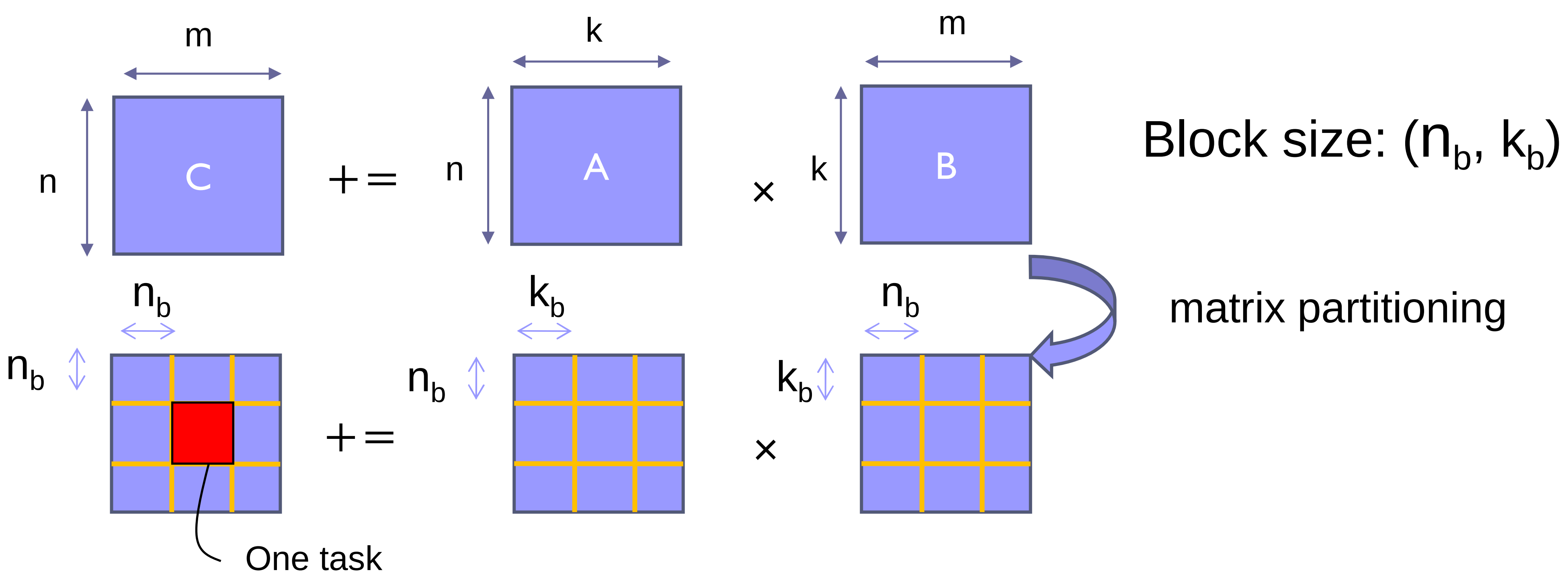
	Problem	Our solution
STEP1	Static scheduling is inefficient on personal computers	Tile matrices by sub-matrices with **block size** and use dynamic load balancing
STEP2	We have to decide **block size**	Use Diagonal Searching Algorithm When the **problem size are fixed**
STEP3	We want efficient parameters **for every problem sizes?**	Use Reductive Searching Algorithm and Parameter Selection Algorithm

DL-BLAS overview

	Problem	Our solution
STEP1	Static scheduling is inefficient on personal computers	Tile matrices by sub-matrices with **block size** and use dynamic load balancing
STEP2	We have to decide **block size**	Use Diagonal Searching Algorithm When the **problem size are fixed**
STEP3	We want efficient parameters **for every problem sizes?**	Use Reductive Searching Algorithm and Parameter Selection Algorithm

DL-BLAS task splitting

- DGEMM calculation can be written as
  $C = \alpha AB + \beta C$
- A is partitioned by (n_b, k_b) matrices, B is by (k_b, n_b) and C is be (n_b, n_b)



Parameter Tuning Algorithms in DL-BLAS

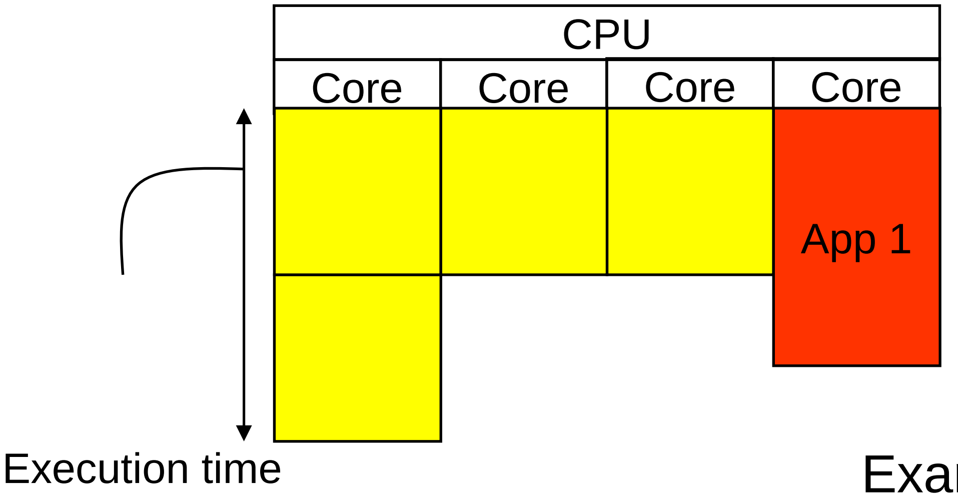
DL-BLAS overview

	Problem	Our solution
STEP1	Static scheduling is inefficient on personal computers	Tile matrices by sub-matrices with **block size** and use dynamic load balancing
STEP2	We have to decide **block size**	Use Diagonal Searching Algorithm When the **problem size are fixed**
STEP3	We want efficient parameters **for every problem sizes?**	Use Reductive Searching Algorithm and Parameter Selection Algorithm

DL-BLAS performance modeling

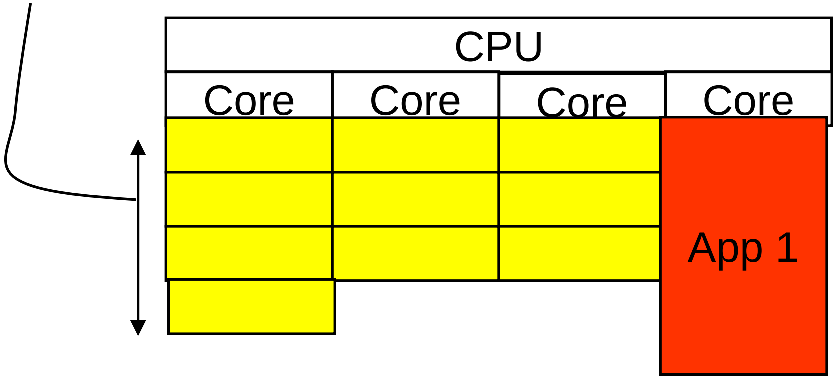
- DL-BLAS performance =
  single-thread performance × multi-thread speed-up
- We modeled the lower bound of multi-thread speed-up
- We created an algorithm to find quasi-maximum value of single-thread performance

Multi-thread speed-up and the number of sub tasks



Example 1: sub-tasks are large

If there are 3 available CPU cores and 4 sub tasks, they need 2 unit time.
The speed-up rate is 4 / 2 = 2
Parallel efficiency: 2 / 3 = 0.667



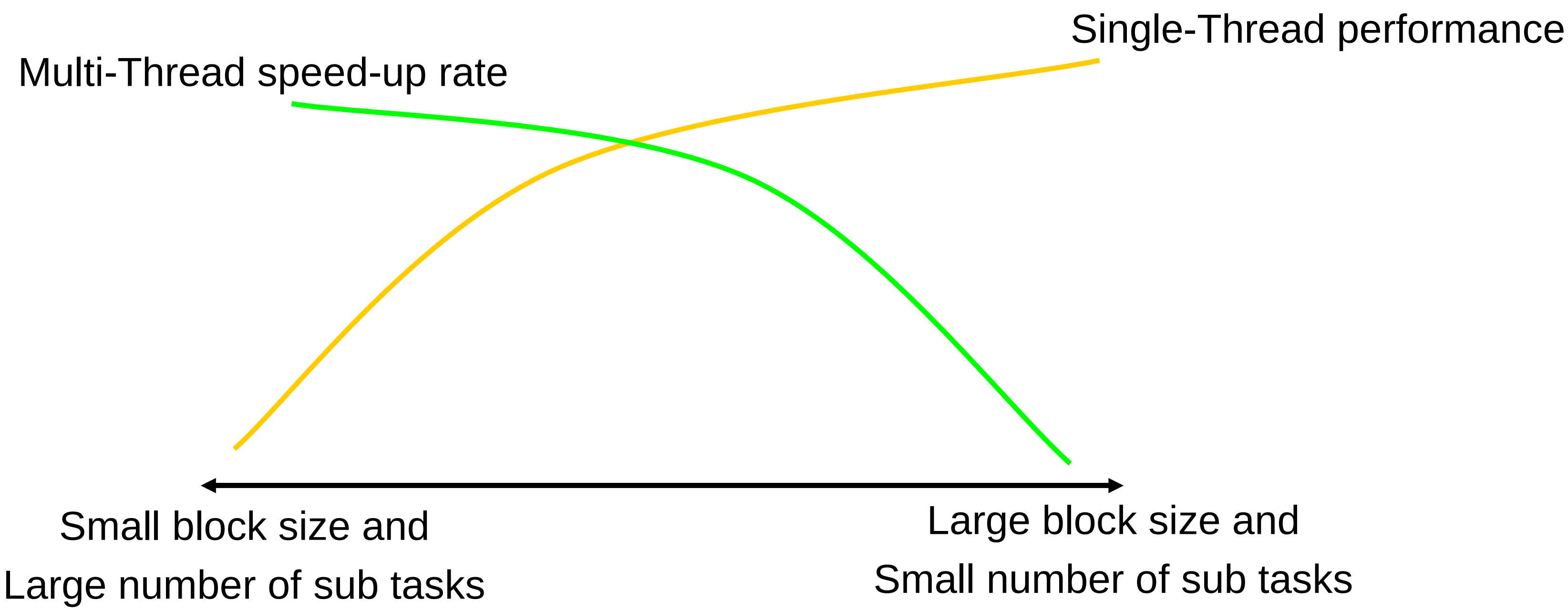
Example 2: sub-tasks are small

If there are 3 available CPU cores and 10 sub tasks, they need 4 unit time.
the speed-up rate is 10 / 4 = 2.5
Parallel efficiency: 2.5 / 3 = 0.833

If there are only small number of tasks, the speed-up rate can be small

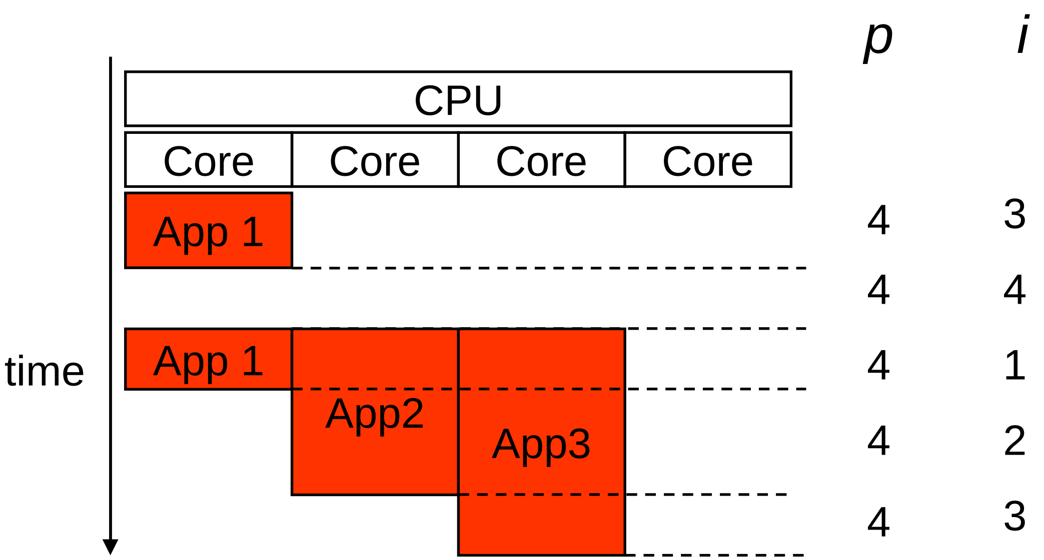
Trade-off between multi-thread speedup and single-thread performance

- Generally there are following trends
  - Larger block size provides higher single-thread calculation performance
  - Larger block size make the number of tasks smaller, and then load balance mechanism do not work well



Multi-Thread speed-up rate (1/2)

- We denote parameters, *p* and *i*



p: The number of physical cores
(NOT be changed in a machine)

i: available CPU cores
(may be changed by other applications)

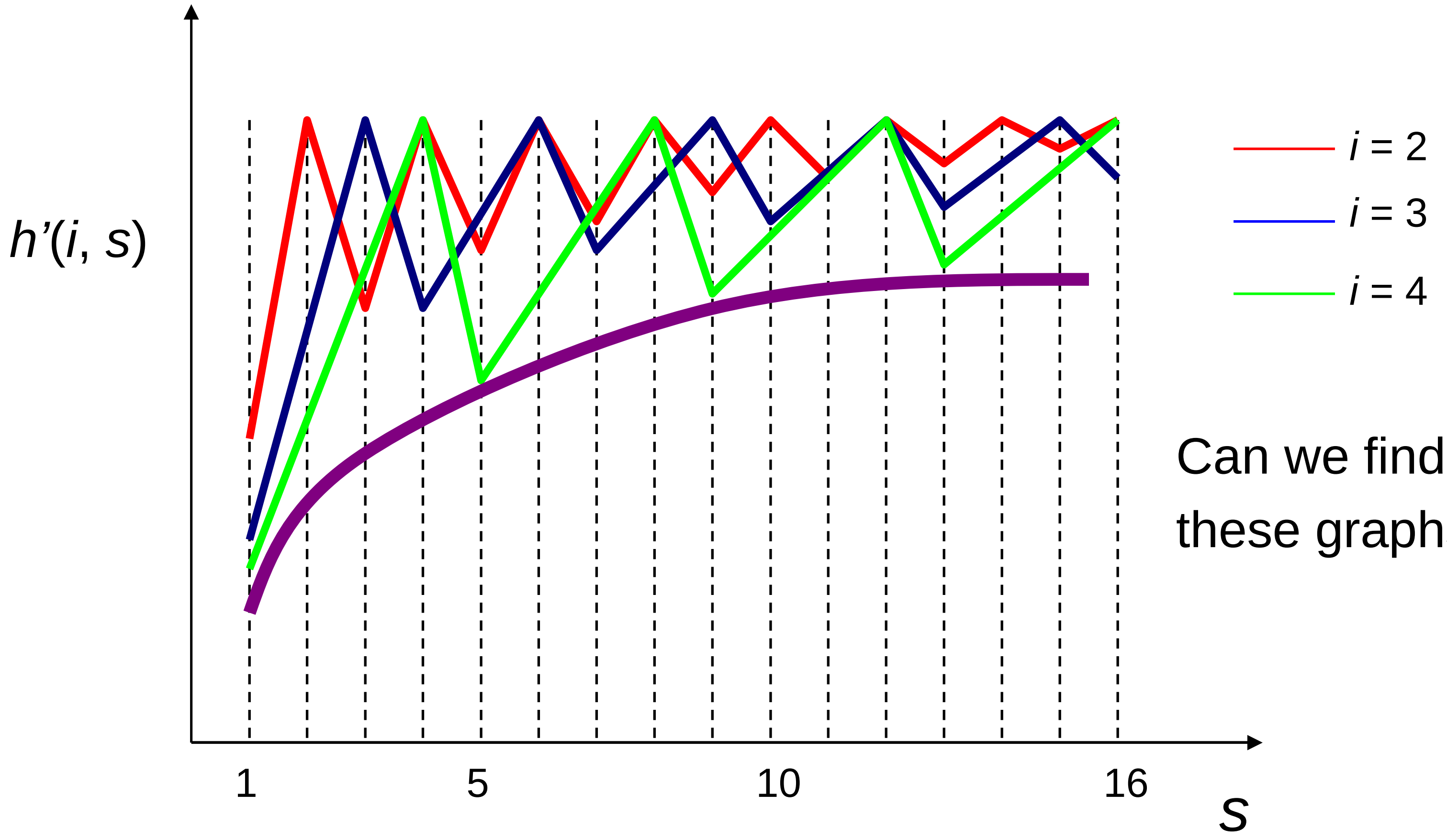
Clearly, i ≦ p

- Letting the number of sub tasks be *s*, we define the speed-up rate *h*(*i*, *s*) and *h'*(*i*, *s*) as follows

$$h(i,s) = ih'(i,s) = \frac{\text{GFLOPS value of multi-threads (using } i \text{ cores) calculation}}{\text{GFLOPS value of single-thread calculation}}$$

Multi-Thread speed-up rate (2/2)

$$h(i,s) = ih'(i,s) = \frac{\text{GFLOPS value of multi-threads (using } i \text{ cores) calculation}}{\text{GFLOPS value of single-thread calculation}}$$



Can we find the lower bound of these graphs?

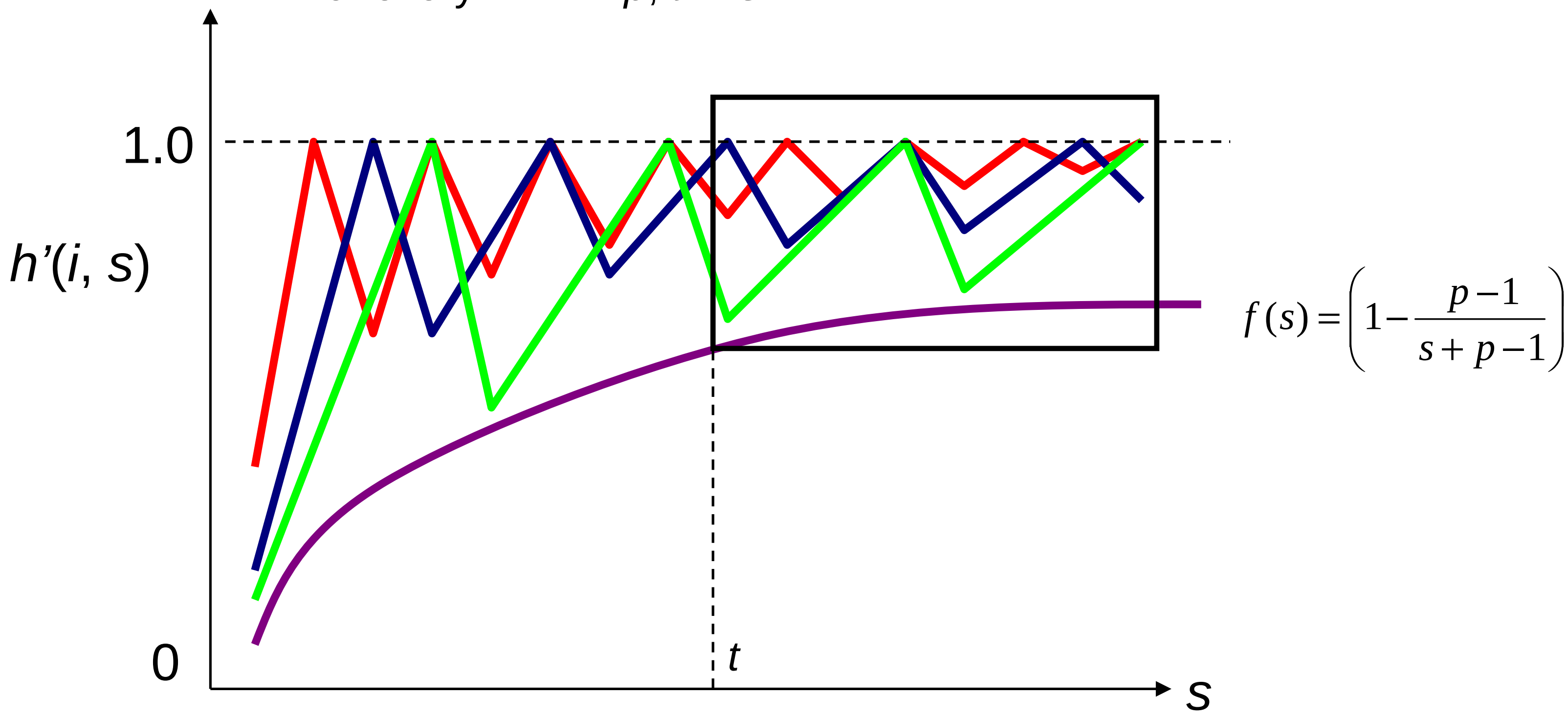
Theorem about speed-up rate

- We proved following theorem

$$h'(i,s) \geq \left(1 - \frac{p-1}{t+p-1}\right)$$

For every $1 < i \leqq p$, $t \leqq s$

p: number of physical CPU cores
s: number of tasks
i: number of CPU cores which can be used on that time



Example of speed-up rate

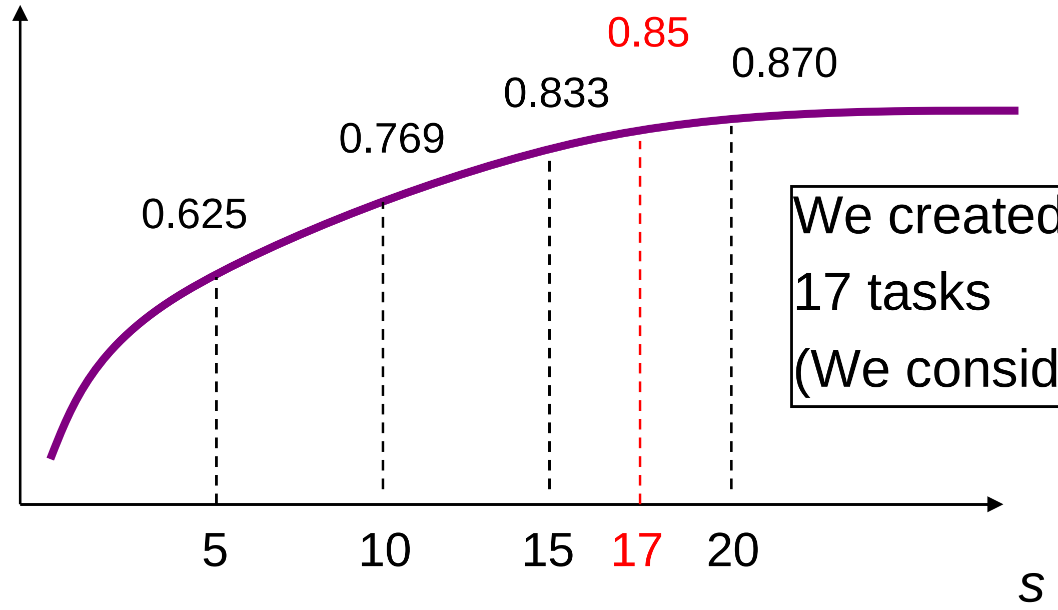
- Case: $p = 4$

In the evaluations, we used machines which have 4 physical CPU cores or 4 CPU threads

$$h'(i,s) \geq \left(1 - \frac{p-1}{t+p-1}\right)$$

p: number of physical CPU cores
s: number of tasks
i: number of CPU cores which can be used on that time
$1 < i \leqq p$, $t \leqq s$

$$f(s) = \left(1 - \frac{p-1}{s+p-1}\right)$$



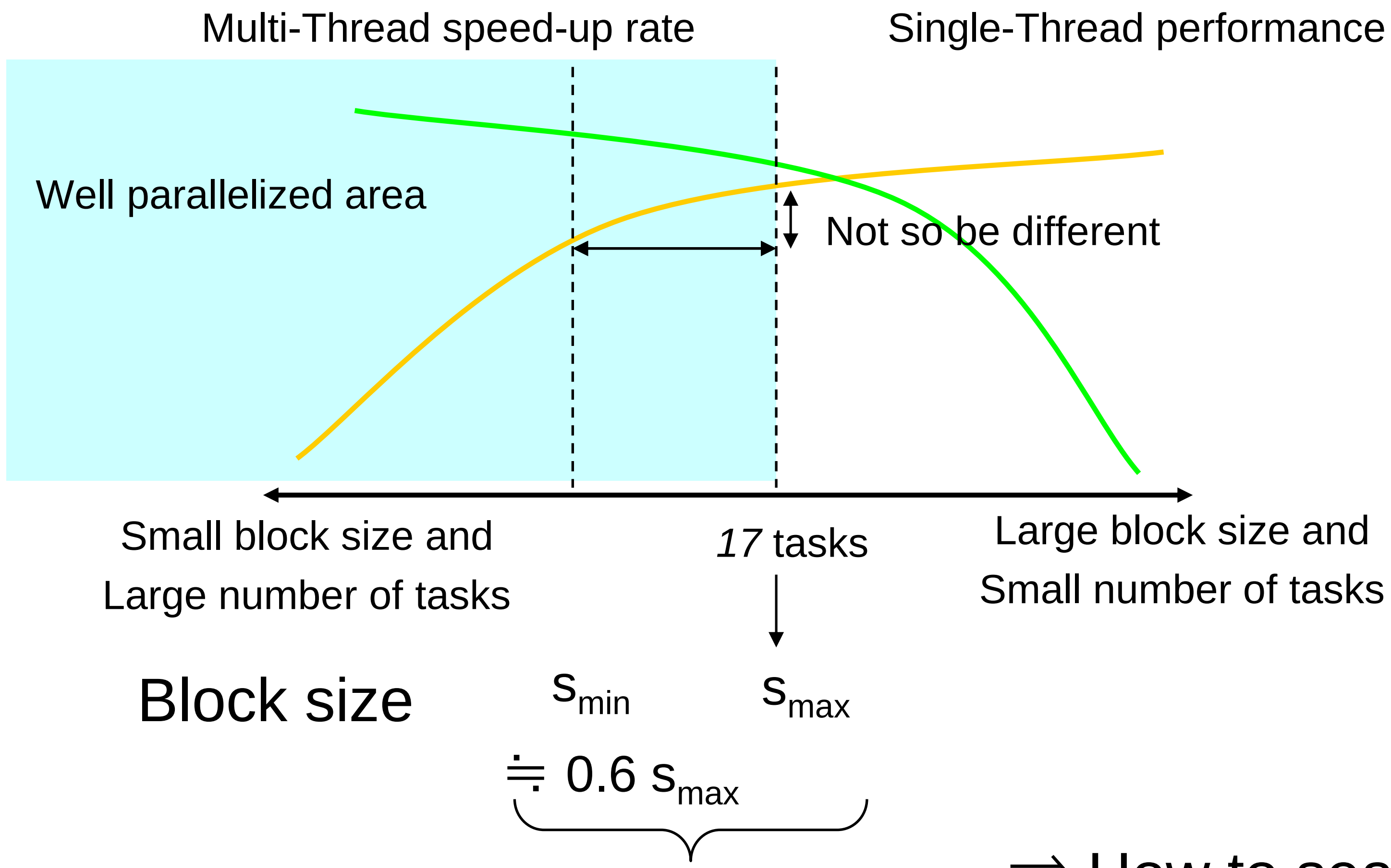
We created more than or equal to 17 tasks
(We considered 85% is well parallelized)

DL-BLAS overview

	Problem	Our solution
STEP1	Static scheduling is inefficient on personal computers	Tile matrices by sub-matrices with **block size** and use dynamic load balancing
STEP2	We have to decide **block size**	Use Diagonal Searching Algorithm When the **problem size are fixed**
STEP3	We want efficient parameters **for every problem sizes?**	Use Reductive Searching Algorithm and Parameter Selection Algorithm

Searching space of block size

- Basic idea is follows

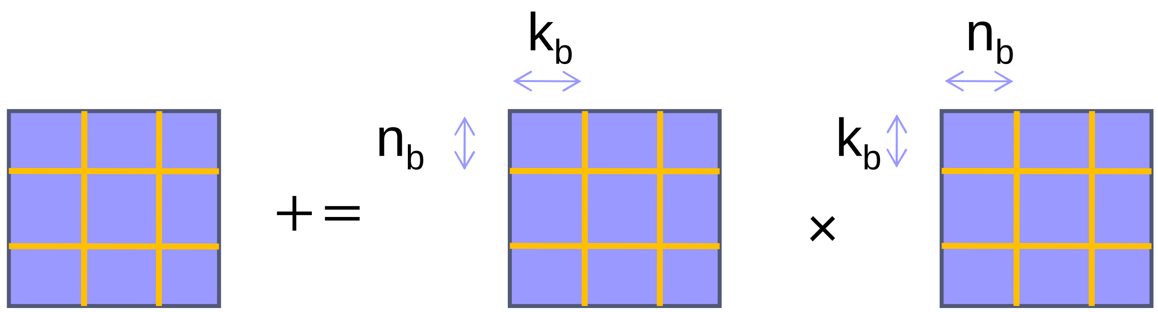


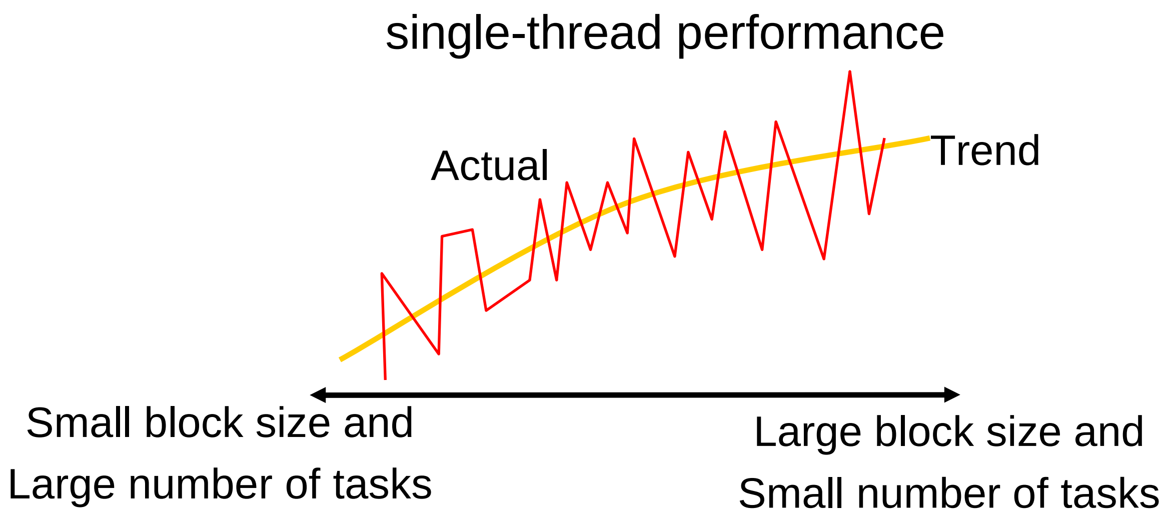
Search parameters in this range

⇒ How to search?

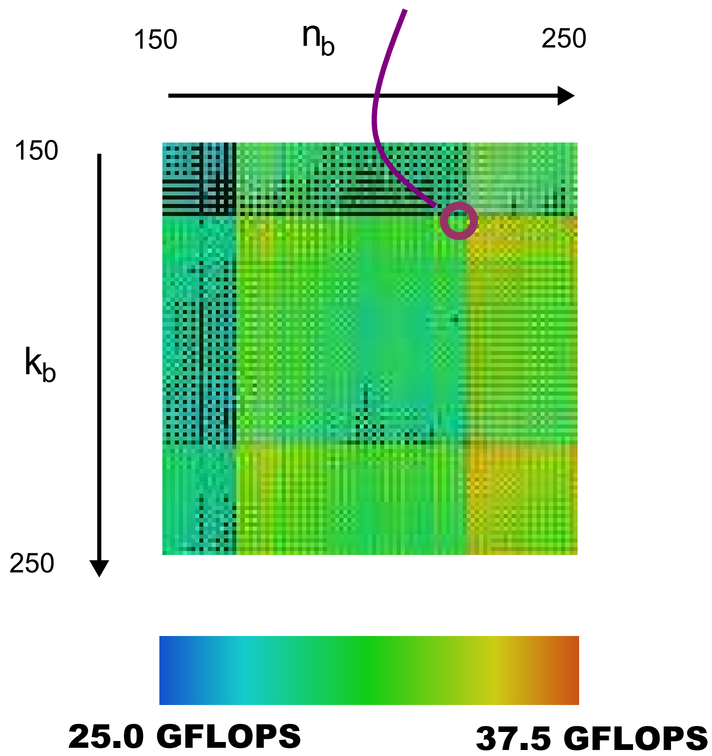
Relation between block sizes and single-thread performance

- Exhaustive search results of calculating 1000 by 1000 square matrix multiplication on Q6600 (Intel Core 2 Quad)





Highest performance block size

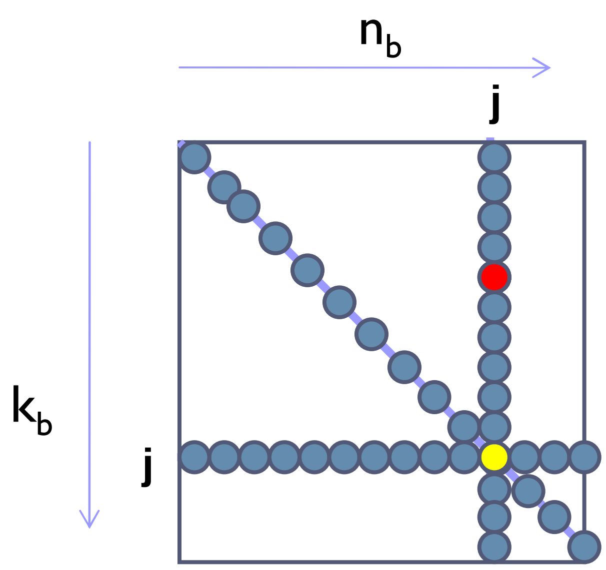


Diagonal Searching Algorithm

- Evaluating all parameters $s_{min} \leqq n_b, k_b \leqq s_{max}$ need much time to calculate
  - Diagonal Searching Algorithm decrease the calculation complexity

Algorithm

1. Evaluate all paramters $s_{min} \leqq n_b = k_b \leqq s_{max}$, the highest performance block size be *j*
2. Search parameters in $n_b = j$ or $k_b = j$
3. Choose the highest performance block size found in STEP 1 and STEP 2



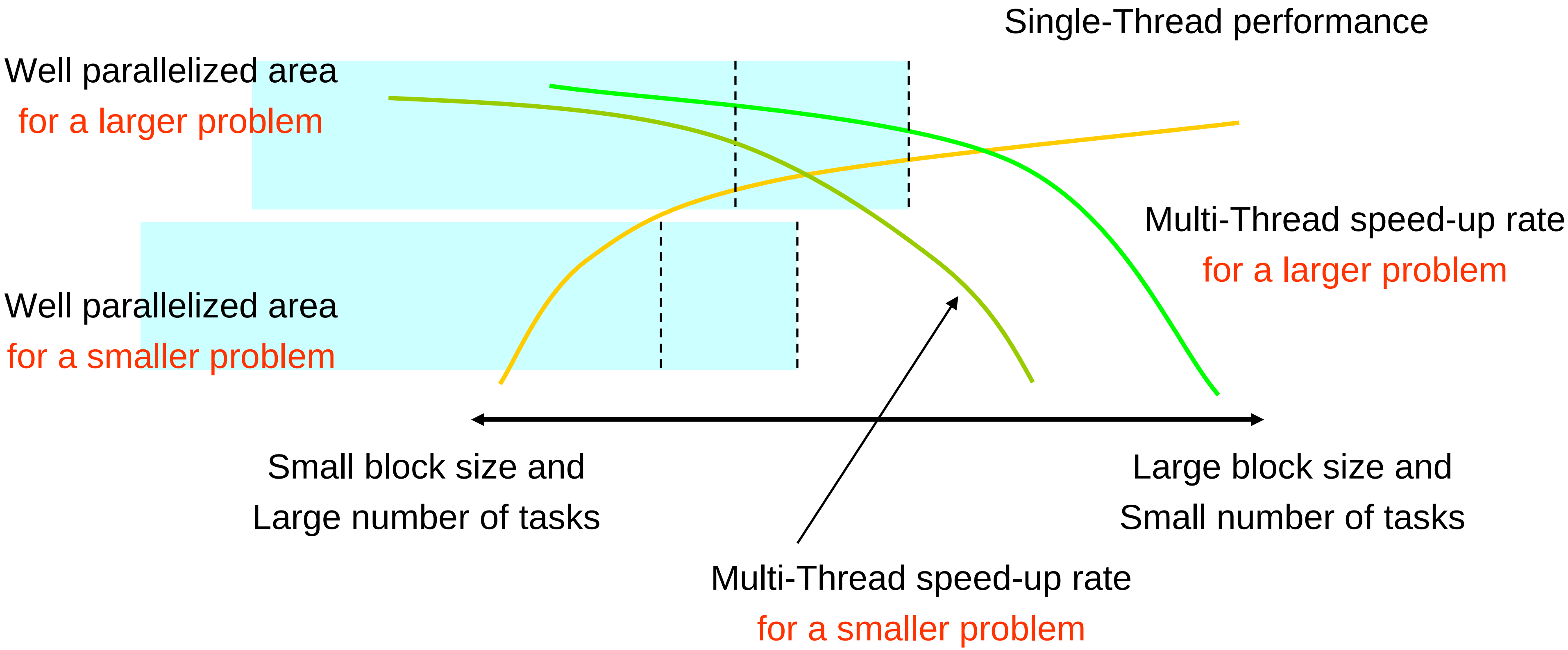
This algorithm returned the parameters which gave us the highest performance on many architectures (Intel Core 2 Extreme, Intel Core i7, AMD Phenom, AMD Phenom II)

DL-BLAS overview

	Problem	Our solution
STEP1	Static scheduling is inefficient on personal computers	Tile matrices by sub-matrices with **block size** and use dynamic load balancing
STEP2	We have to decide **block size**	Use Diagonal Searching Algorithm When the **problem size are fixed**
STEP3	We want efficient parameters **for every problem sizes?**	Use Reductive Searching Algorithm and Parameter Selection Algorithm

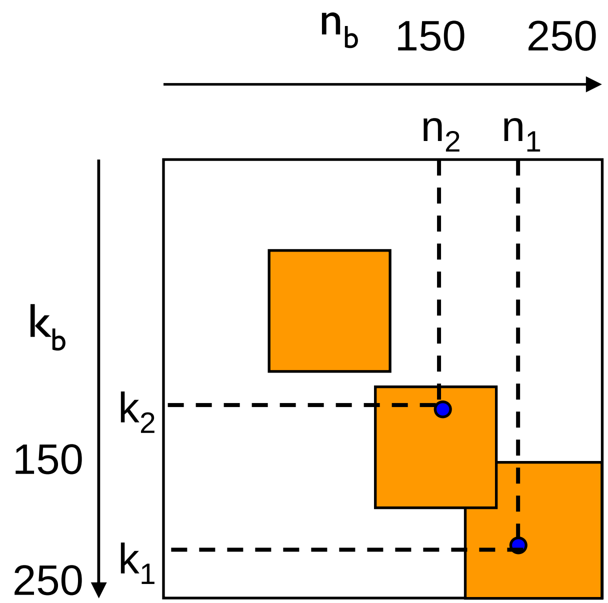
Relation between block size and problem size

- Multi-Thread speed-up rate is changed when the problem size is changed



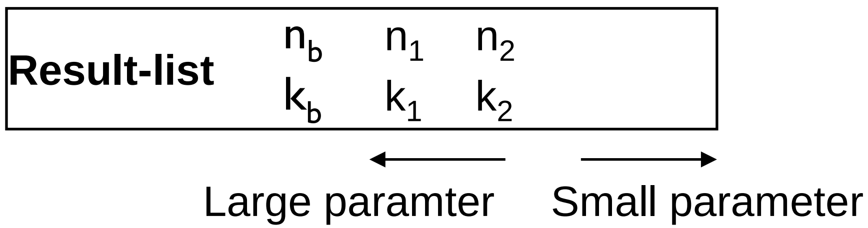
Reductive Searching Algorithm

- We want to get multiple $[n_b, k_b]$ pairs



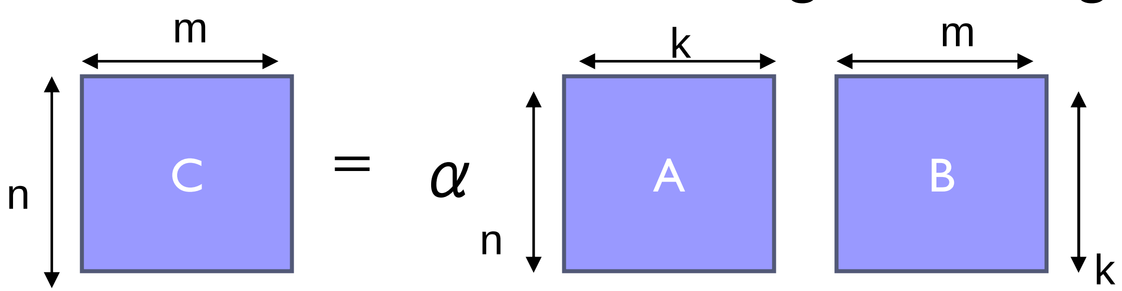
Algorithm:

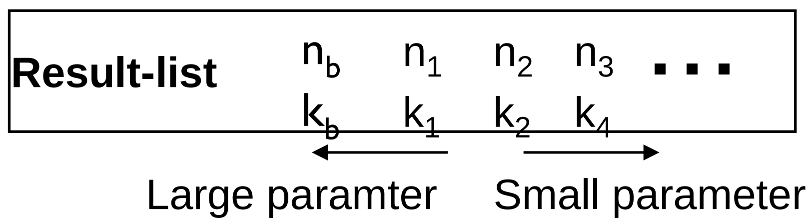
1. Initialize s_{min} and s_{max} be [150, 250], and a = 1
2. In the range $[s_{min}, s_{max}]$, run Diagonal Searching Algorithm, and let the results of the solution be $[n_a, k_a]$
3. If single-thread calculation is faster than multi-thread calculation, then last the algorithm
4. let $[s_{min}, s_{max}] = [0.5\ n_a, 0.9\ n_a]$ and add $[n_a, k_a]$ to **result-list**
5. a := a + 1 and Go to STEP 2



Parameter Selection

For each problem, we select one pair of the parameters (n_a, k_a) from the **result-list** using following algorithm





Algorithm

For i = 1 to length(**Result-list**)

num-tasks = $\lceil n / n_i \rceil \lceil m / n_i \rceil$

if num-tasks > 16

return (n_i, k_i);

end

return 0 // single-thread calculation

End for

Select largest pair of parameters which make more than 16 tasks

Algorithms’ work-flow

Installation time	Prepare hardware and BLAS routine ↓ Tune DL-BLAS using Diagonal Searghing Algorithm and Reductive Searching Algorithm ↓ Store **result-list** on the disc
Calculation time	When DL-BLAS routines are called, get result-list from the disc ↓ Using Parameter Selection Algorithm, decide the block size and calculate

Result-list	n_b	n_1	n_2	n_3	・・・・・
	k_b	k_1	k_2	k_4	

Performance Evaluation Results

Algorithms' work-flow

Installation time

Prepare hardware and BLAS routine

↓

Tune DL-BLAS using
Diagonal Searghing Algorithm and Reductive Searching Algorithm

↓

Store **result-list** on the disc

Result-list	n_b	n_1	n_2	n_3	……
	k_b	k_1	k_2	k_4	

Calculation time

When DL-BLAS routines are called, get result-list from the disc

↓

Using Parameter Selection Algorithm, decide the block size and calculate

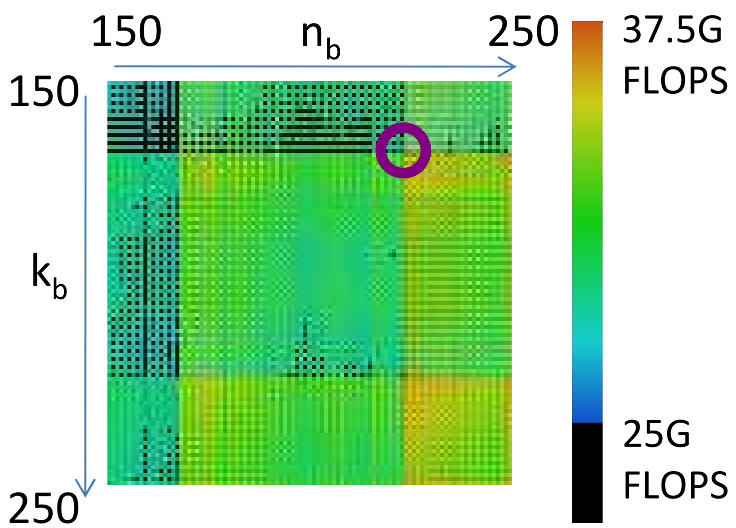
Hardware and Problems

- We used some CPU models
  - Intel Core 2 Extreme, QX9650
  - Intel Core i7 965
  - Intel ATOM 330
  - AMD Phenom 9600
  - AMD Phenom II X4 940

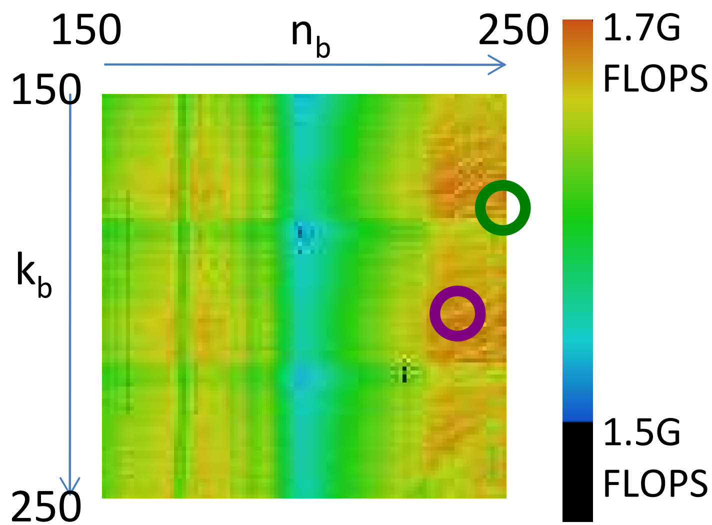
- As first, we evaluate Diagonal Searching Algorithm using 1000 by 1000 square matrix multiplications

Results of Diagonal Searching Algorithm

Intel Core 2 Extreme (QX9650)



Intel ATOM 330



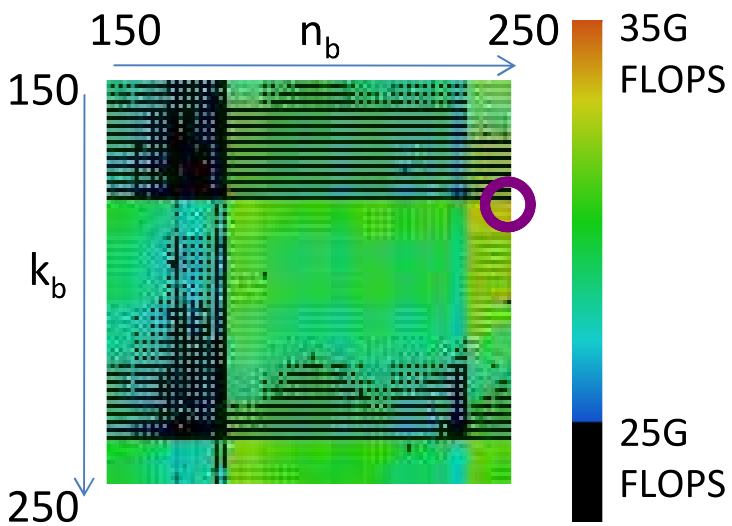
○ Results of Diagonal Searching Algorithm
= Highest performance parameters

○ Results of Diagonal Searching Algorithm
1.69 GFLOPS

○ Highest performance parameters
1.70 GFLOPS

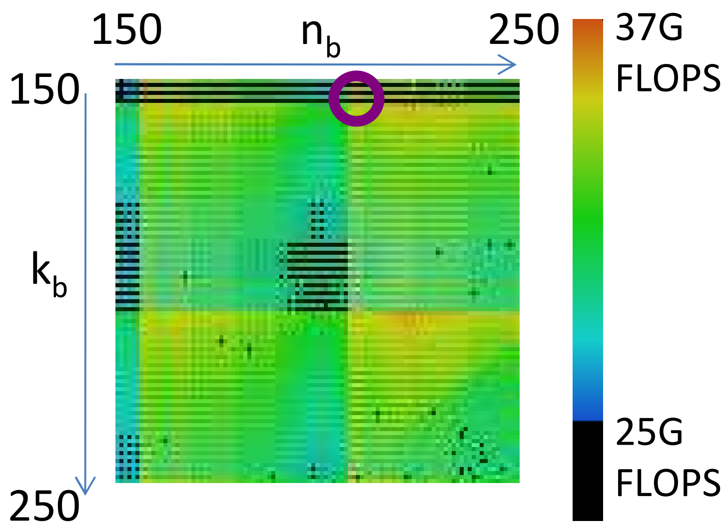
Results of Diagonal Searching Algorithm

Intel Core i7 965 **with** hyper-threading



Results of Diagonal Searching Algorithm
= Highest performance parameters

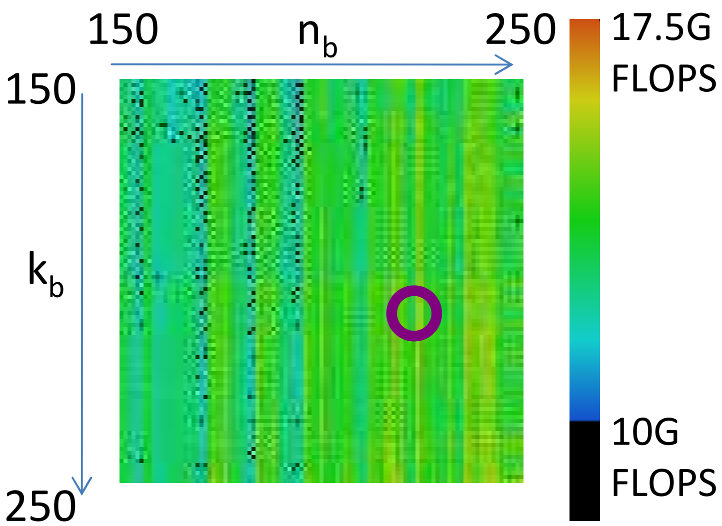
Intel Core i7 965 **without** hyper-threading



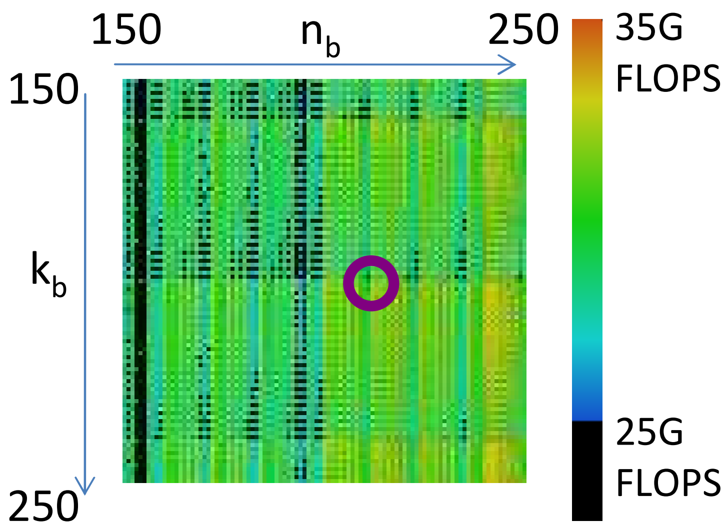
Results of Diagonal Searching Algorithm
= Highest performance parameters

Results of Diagonal Searching Algorithm

AMD Phenom 9600



AMD Phenom II X4 940



○ Results of Diagonal Searching Algorithm
= Highest performance parameters

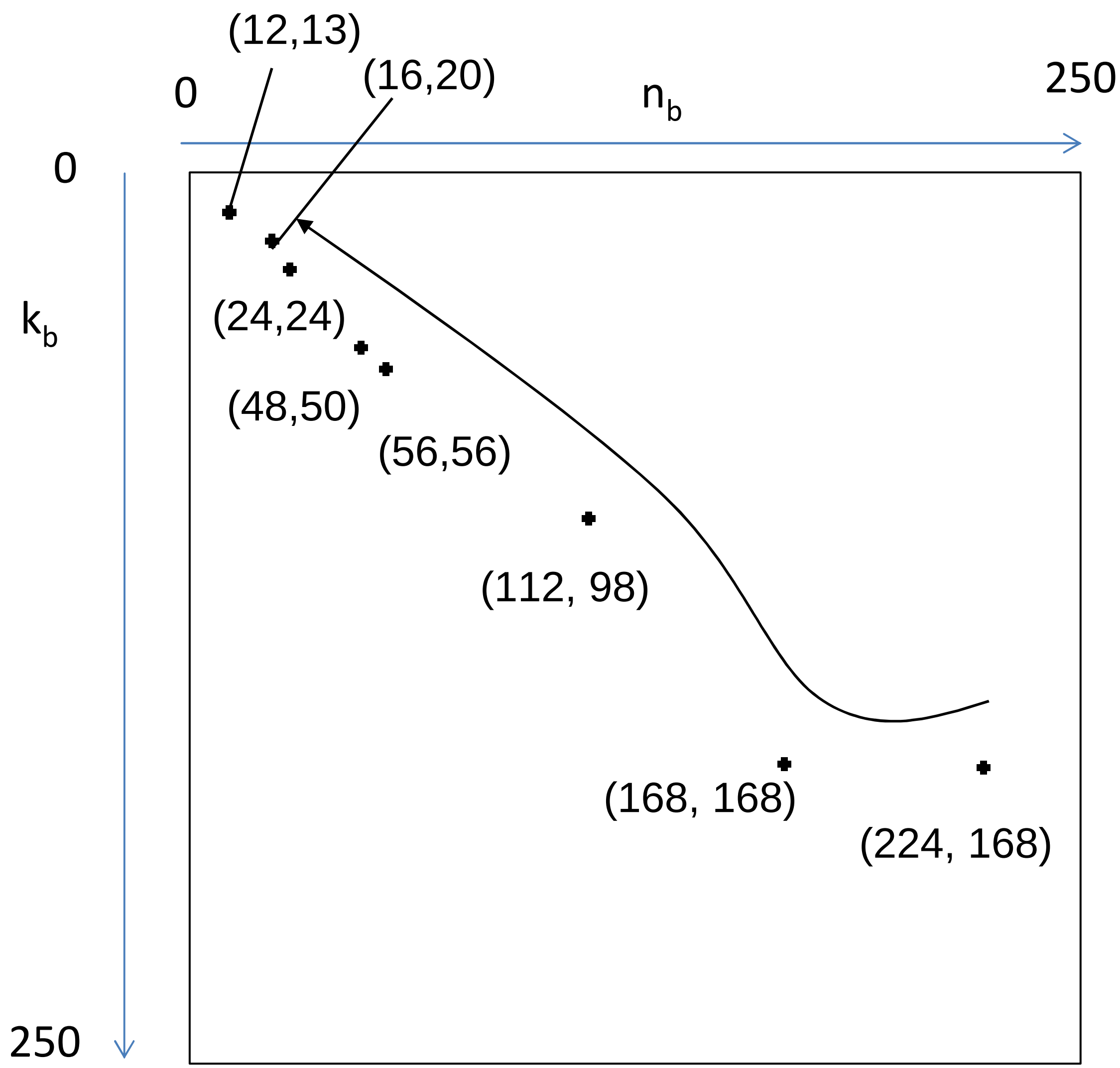
○ Results of Diagonal Searching Algorithm
= Highest performance parameters

Results of Reductive Searching Algorithm

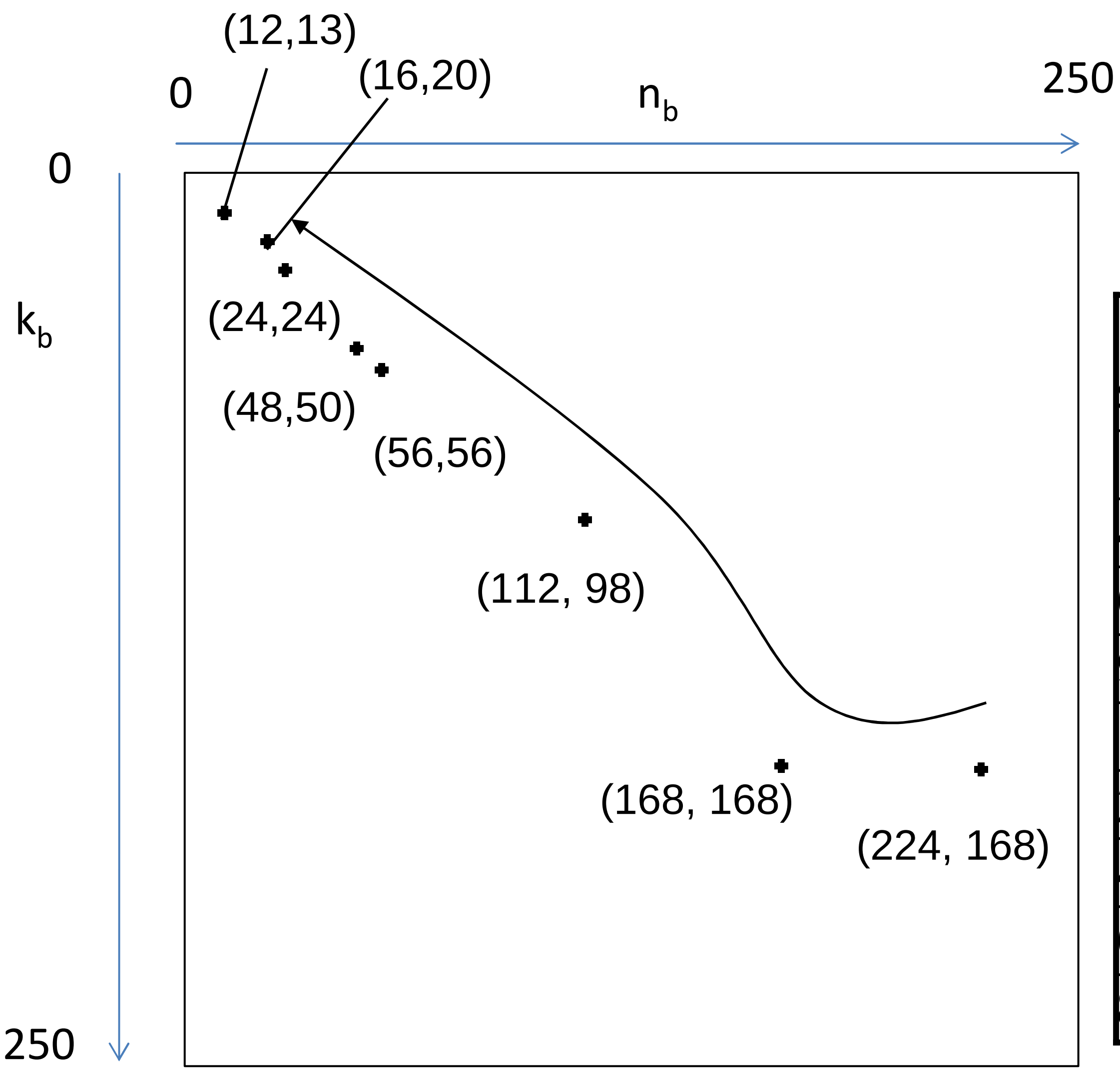
Intel Core 2 Extreme (QX9650)

Result-list is shown in right

This calculation need less than half an hour



Results of Parameter Selection Algorithm



Intel Core 2 Extreme (QX9650)

In square matrix multiplication, following parameters are used

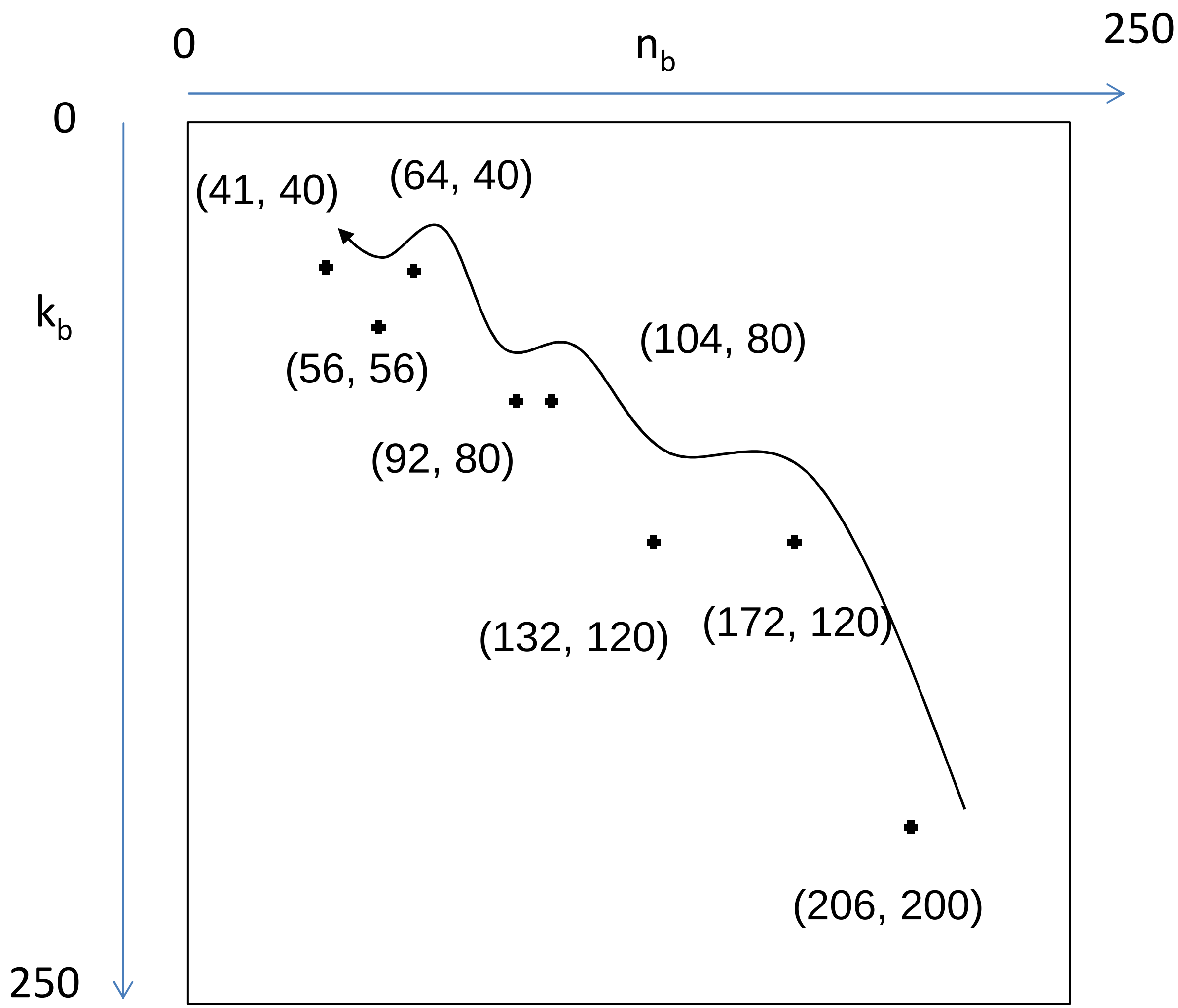
Dimension of square matrix	Used parameter (n_b, k_b)
～48	Single Thread
49～64	(12,13)
65～96	(16, 20)
97～192	(24, 24)
193～224	(48, 50)
225～448	(56, 56)
449～672	(112, 98)
673～892	(168, 168)
893～	(224, 168)

Results of Reductive Searching Algorithm

AMD Phenom II X4 940

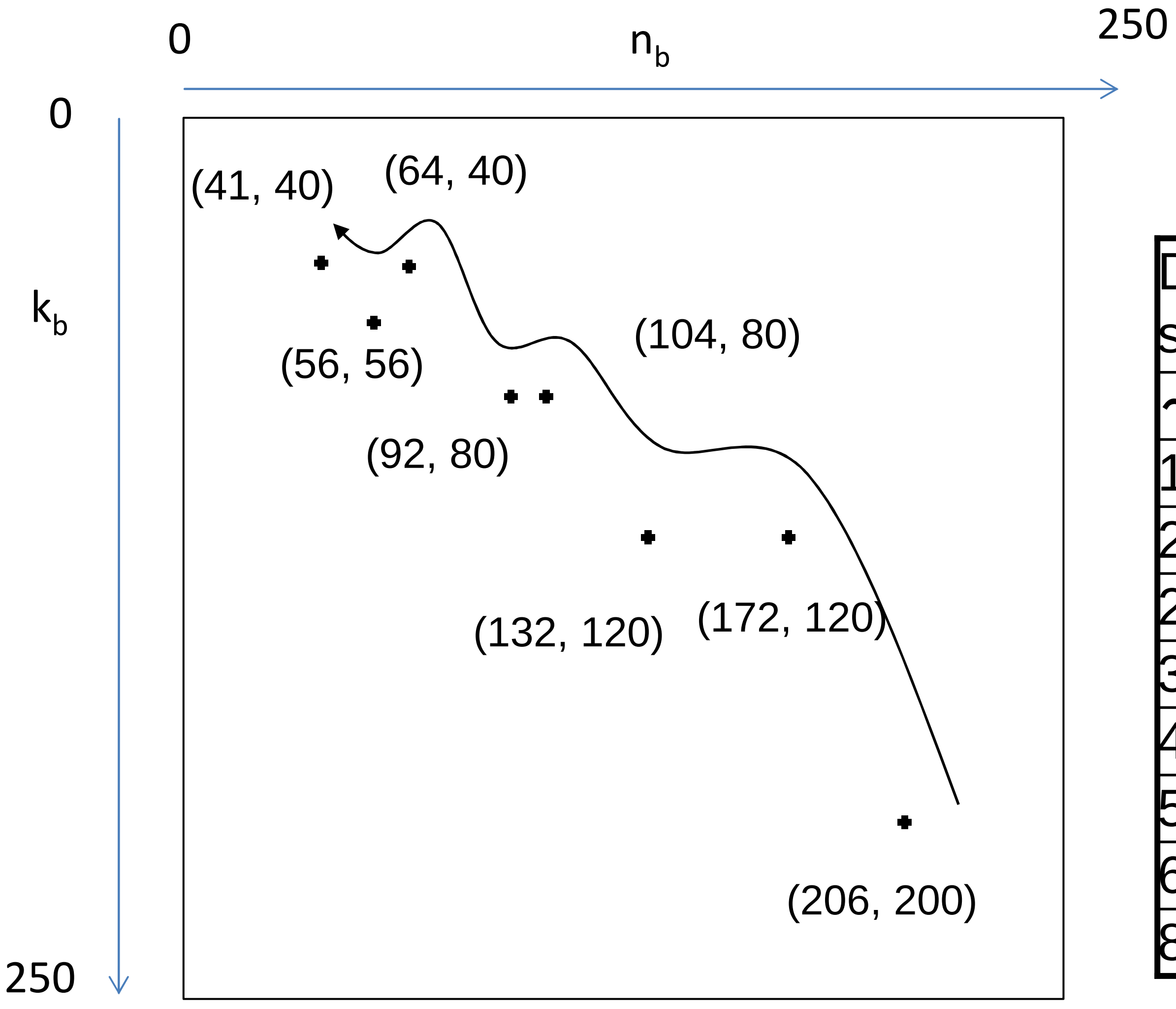
Result-list is shown in right

This calculation need less than half an hour



Results of Parameter Selection Algorithm

AMD Phenom II X4 940



In square matrix multiplication, following parameters are used

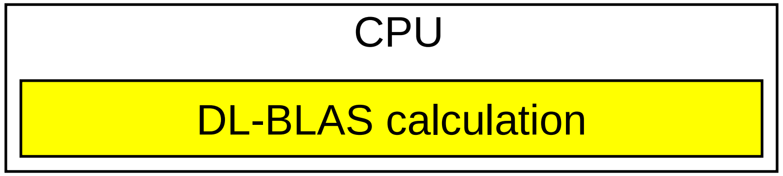
Dimension of square matrix	Used parameter (n_b, k_b)
～164	Single Thread
165～224	(41, 40)
225～256	(56, 56)
257～368	(64, 40)
369～416	(92, 80)
417～528	(104, 80)
529～684	(132, 120)
685～892	(172, 120)
825～	(206, 200)

Evaluation Machines

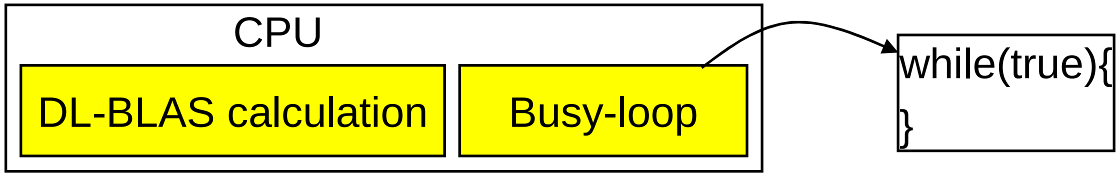
- We solved square matrix multiplication problem with the following hardware and software
- CPU models
  - Intel Core 2 Extreme QX9650
  - Intel Core i7 965
  - Intel ATOM 330
  - AMD Phenom 9600
  - AMD Phenom II X4 940
- BLAS implementations
  - DL-BLAS
  - ATLAS
  - GotoBLAS (version 1.26)

Evaluation Circumstances

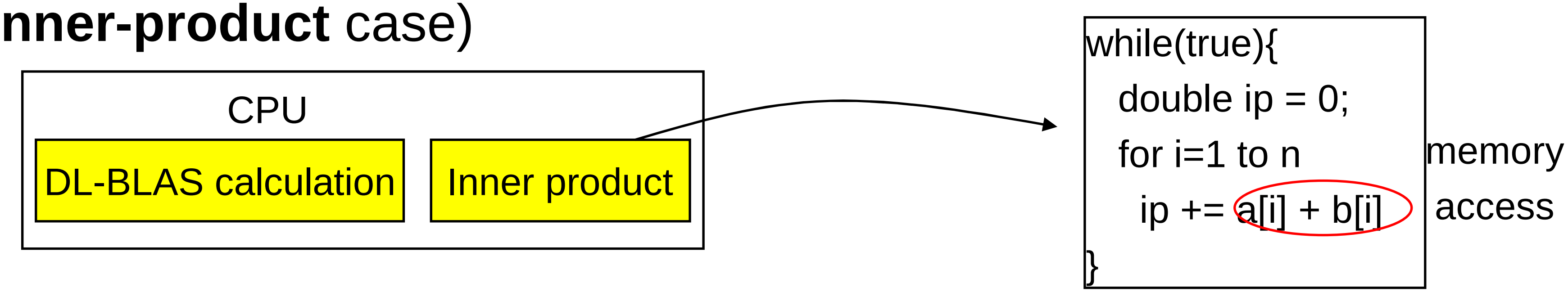
- We make following three circumstances
  - No other tasks are running (**no-task** case)



  - Busy loop program is running concurrently (**busy-loop** case)

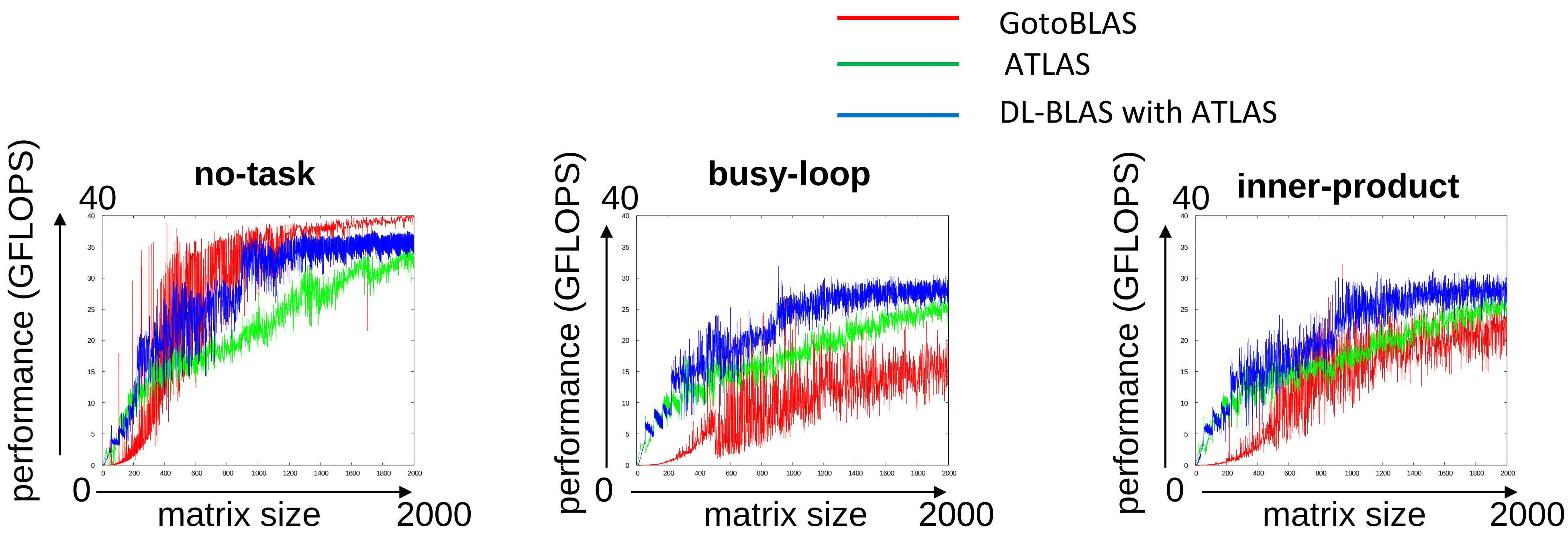


  - Inner product (dot product) program is running concurrently (**inner-product** case)



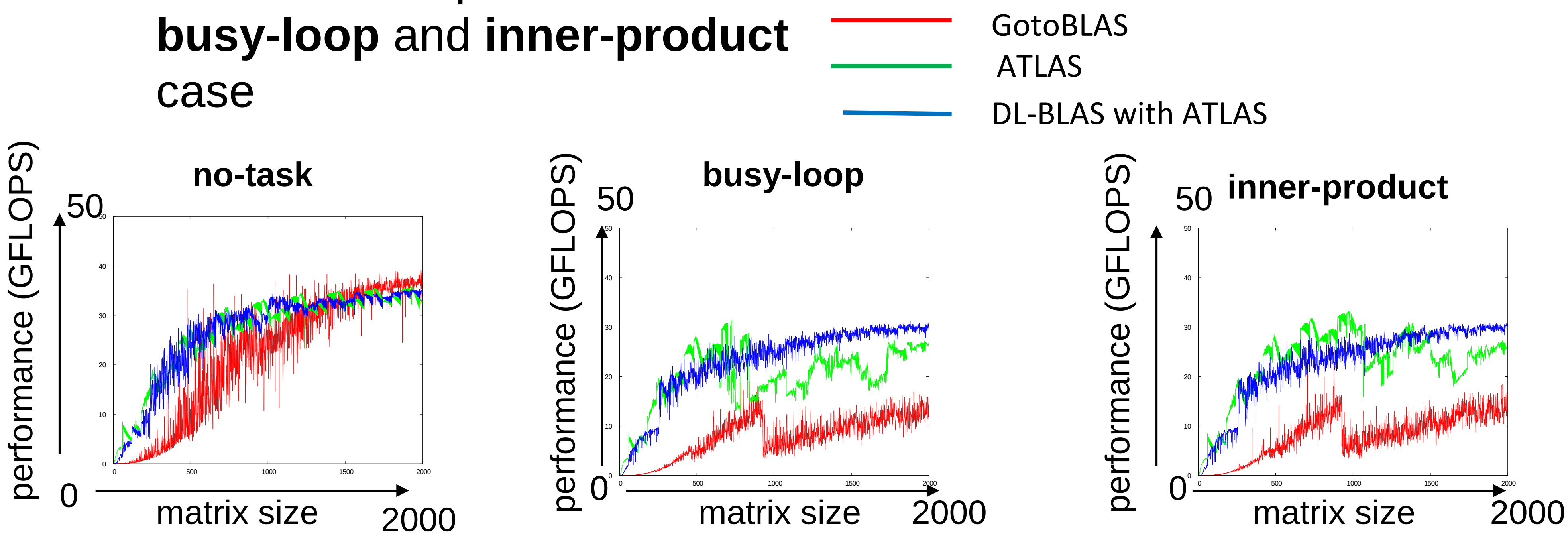
DGEMM calculation on QX9650 (Intel Core 2 Extreme)

- GotoBLAS is fastest in **no-task** case, but slowest in **busy-loop** case and **inner-product** case
- DL-BLAS is faster than ATLAS in all cases
  - DL-BLAS is fastest in **busy-loop** case and **inner-product** case



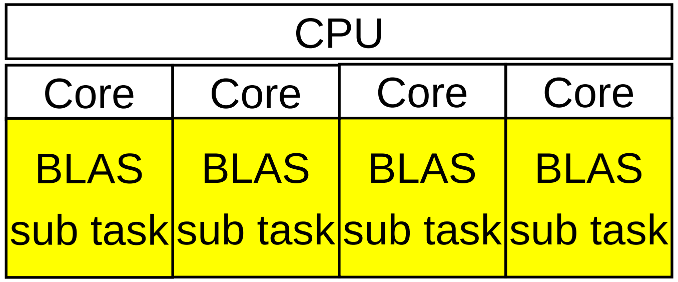
DGEMM calculation on 965 (Intel Core i7) hyper-threading

- GotoBLAS is not so fast on this architecture
  - "Hyper-Thread is harmful" K.Goto said, developer of GotoBLAS
- ATLAS is fastest in some cases
  - Middle size of problems in **busy-loop** and **inner-product** case



High-performance BLAS implementations

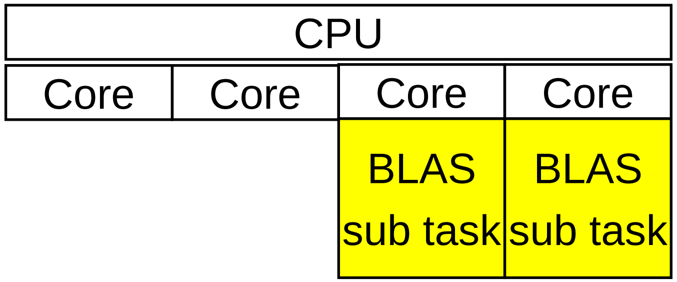
- GotoBLAS is the fastest BLAS implementation in the world today
  - Users can set the number of threads



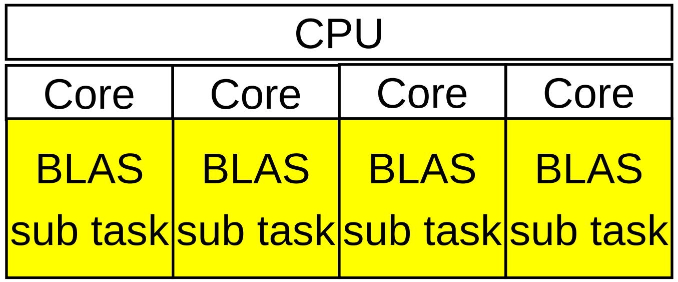
GotoBLAS with NUM_THREAD=4

- ATLAS is BLAS implementation which use auto tuning
  - ATLAS decides the number of threads automatically depending on the problem size

ATLAS for small problems

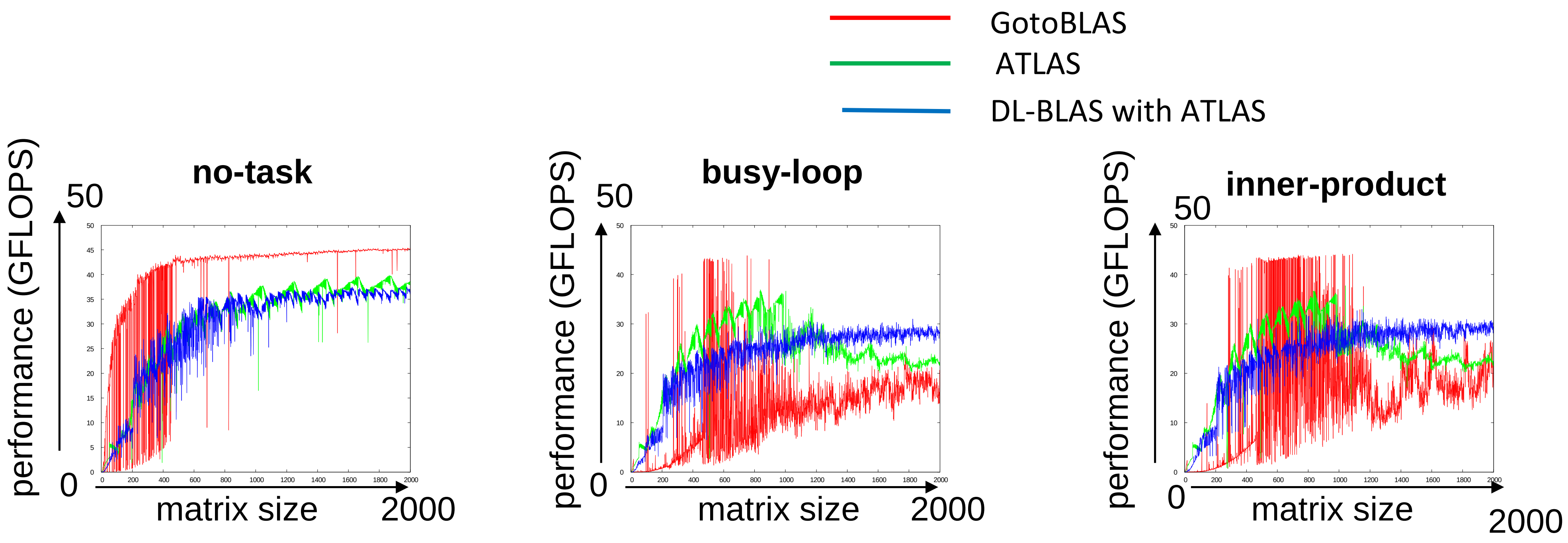


ATLAS for large problems



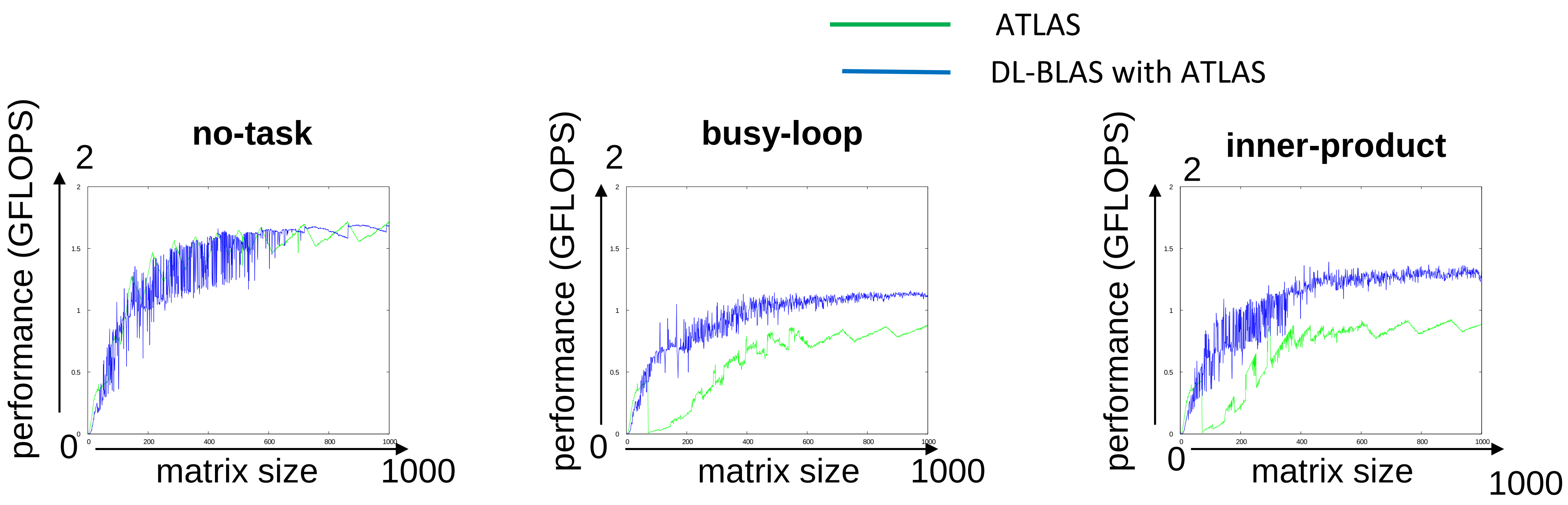
DGEMM calculation on 965 (Intel Core i7) without hyper-threading

- GotoBLAS is fastest in **no-task** case
- In other cases, the performance of GotoBLAS is unstable
  - Other tasks disturb GotoBLAS



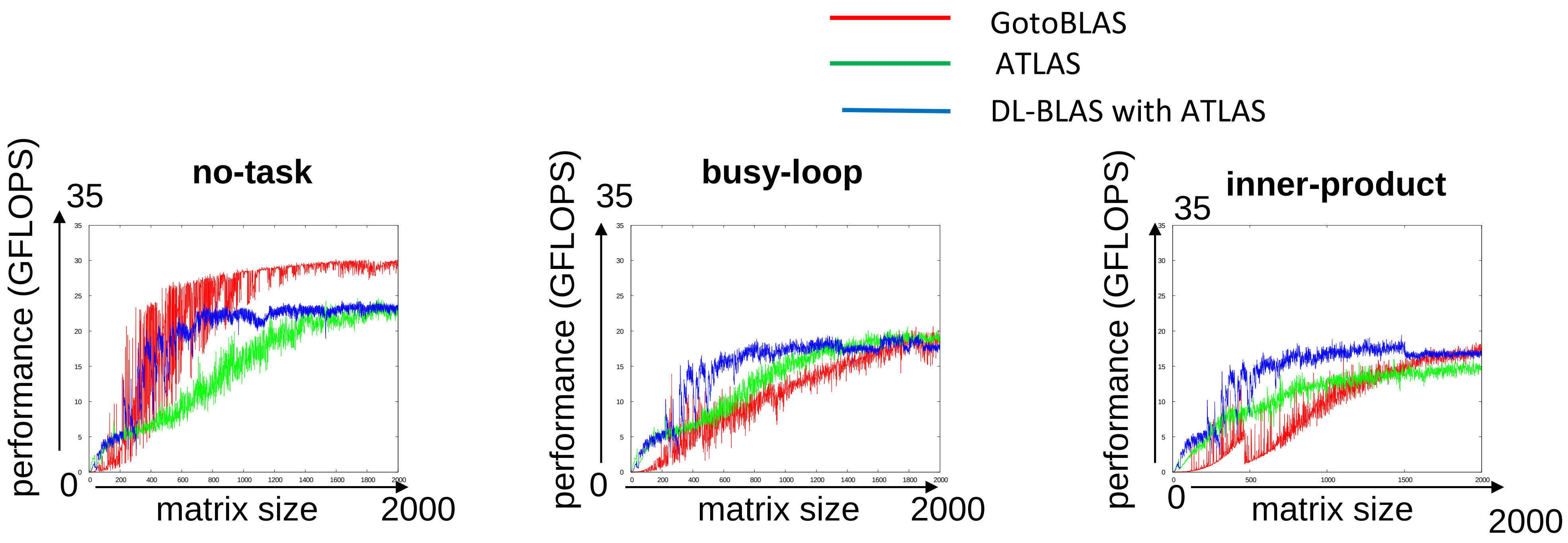
DGEMM calculation on Intel ATOM 330 with hyper-threading

- DL-BLAS is faster in **busy-loop** case and **inner-product** case
- The performance of DL-BLAS is not faster than ATLAS in **no-task** case
  - It is considered that DL-BLAS is disturbed by hyper-threading



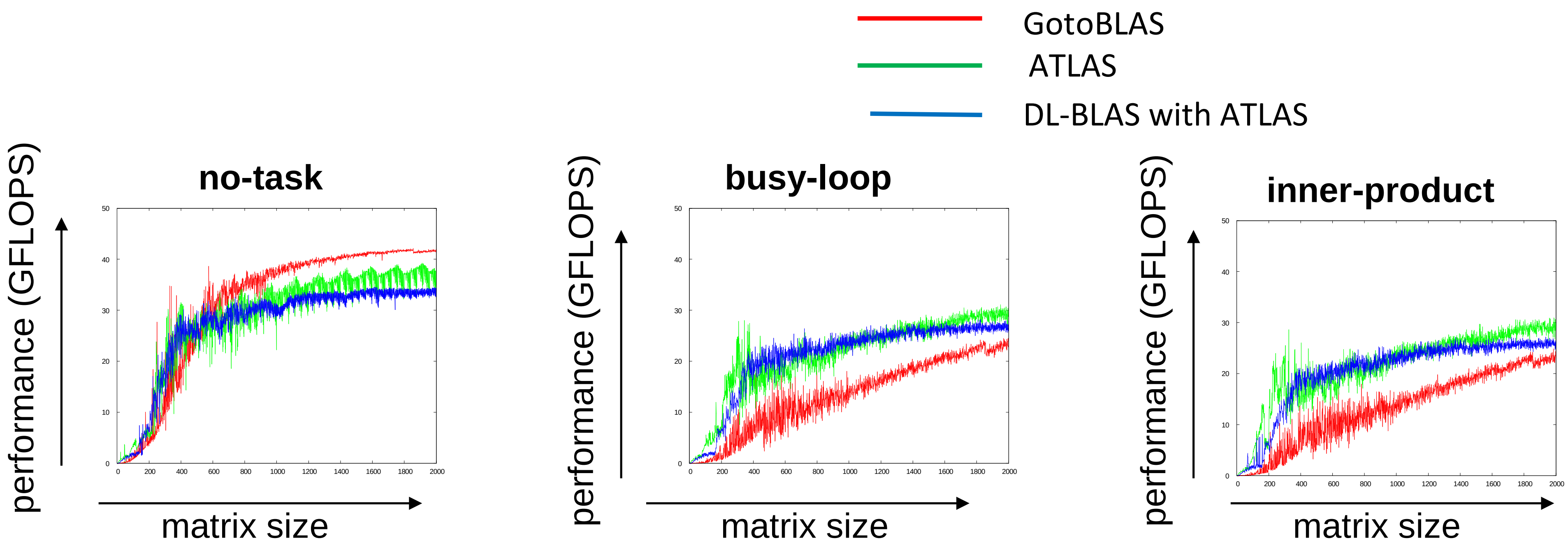
DGEMM calculation on AMD Phenom 9600

- GotoBLAS is fastest in **no-task** case
- DL-BLAS is the fastest for small problems in **busy-loop** case and **inner-product** case
  - For large problems, the performances are not so different



DGEMM calculation on X4 940 (AMD Phenom II)

- GotoBLAS is a little faster than ATLAS and DL-BLAS in **no-task** case
- The performance of DL-BLAS is similar to that of ATLAS



Conclusion

- We implemented a BLAS implementation, which use dynamic load balancing
  - We call the implementation DL-BLAS
- Our implementations need auto tuning technique to get block size parameters
  - We call the techniques Diagonal Searching Algorithm, Reductive Searching Algorithm, and Parameter Selection
- In some cases, performance of DL-BLAS was better than ATLAS and GotoBLAS
  - The performance of DL-BLAS is constantly not so wrong