An Analytical Study of GPU Computation for Solving QAPs by Parallel Evolutionary Computation with Independent Run

Shigeyoshi Tsutsui
Hannan Univ., JAPAN

Noriyuki Fujimoto
Osaka Prefecture Univ., JAPAN

Outline of This Talk

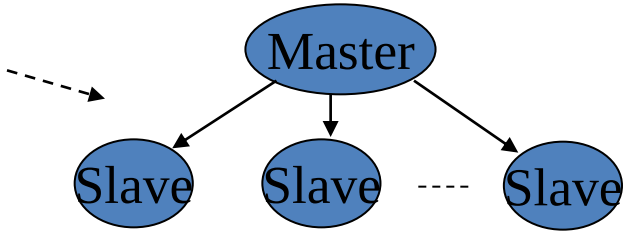
- Background of the research
- Effect of parallel independent run on GPU
- Quadratic Assignment Problem (QAP)
- Implementation Details on GPU
- Results
- Analytical study
- Conclusions and Future Work

Background

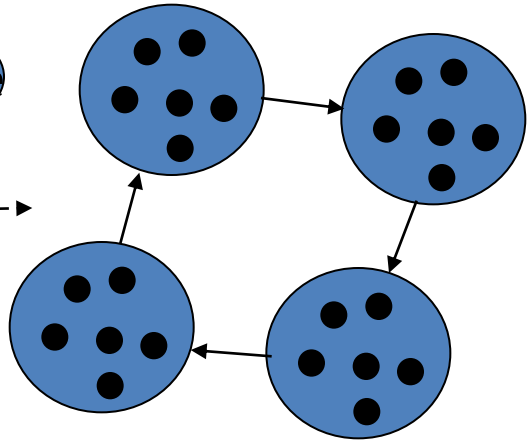
- In a previous study (CIGPU 2009), we applied GPU computation to solve quadratic assignment problems (QAPs) with parallel EC on a single GPU
- The results in that study showed that parallel EC with the GTX285 GPU produce a speedup of x3 to x12 compared to the i7 965 (3.2 GHz)
- However, the analysis of the results was postponed for future work
- In this study, we propose a simplified parallel EC model and analyze how the speedup is obtained using a statistical model of parallel runs of the algorithm

Parallel EC Models

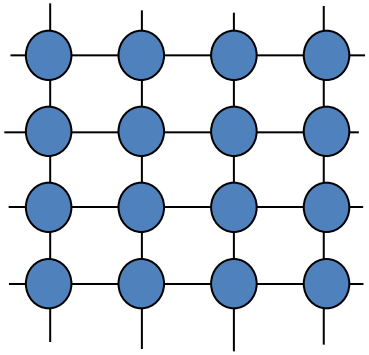
Master-Slave Model



Coarse-grained Model
(Distributed EC)

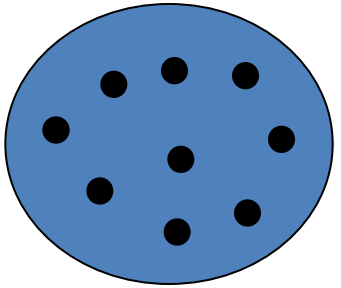


Fine-grained Model



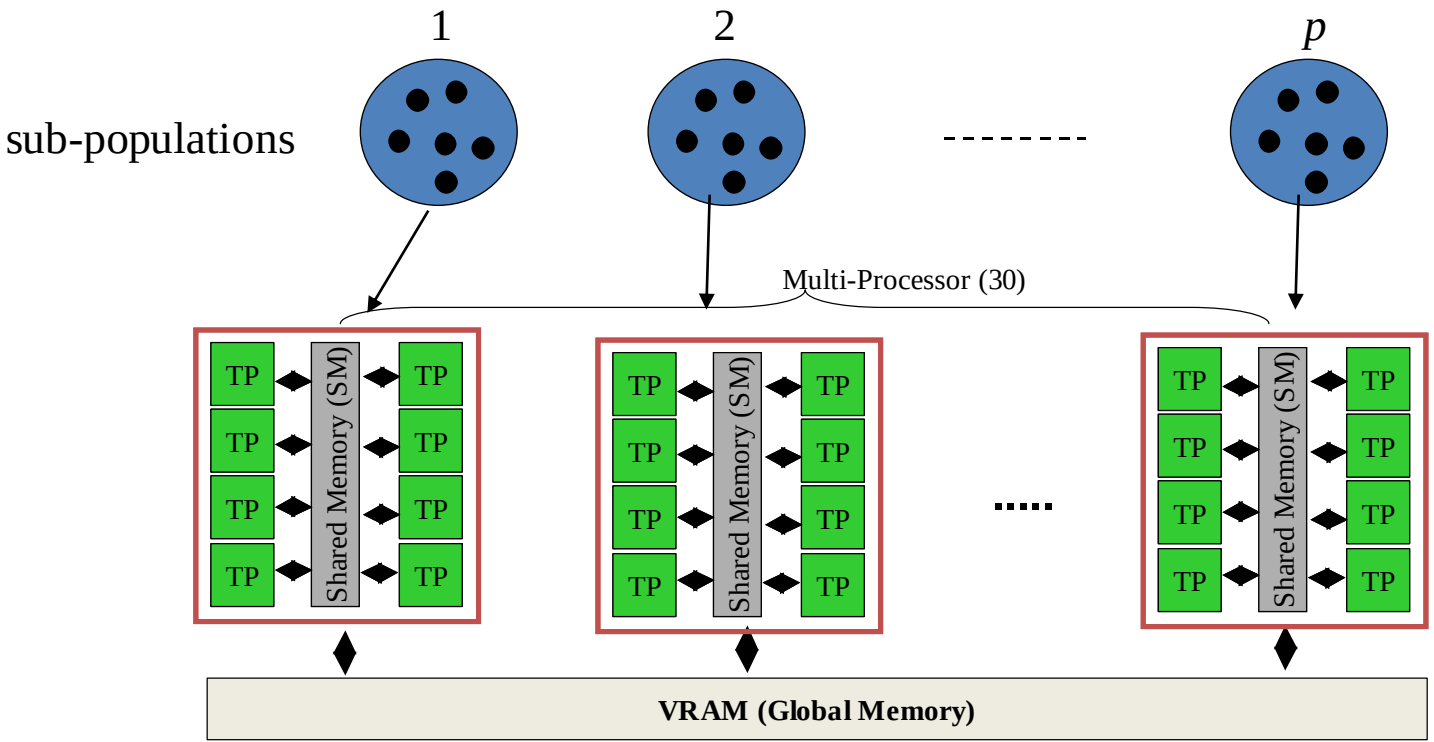
Hybrid Model

Individual-level Model

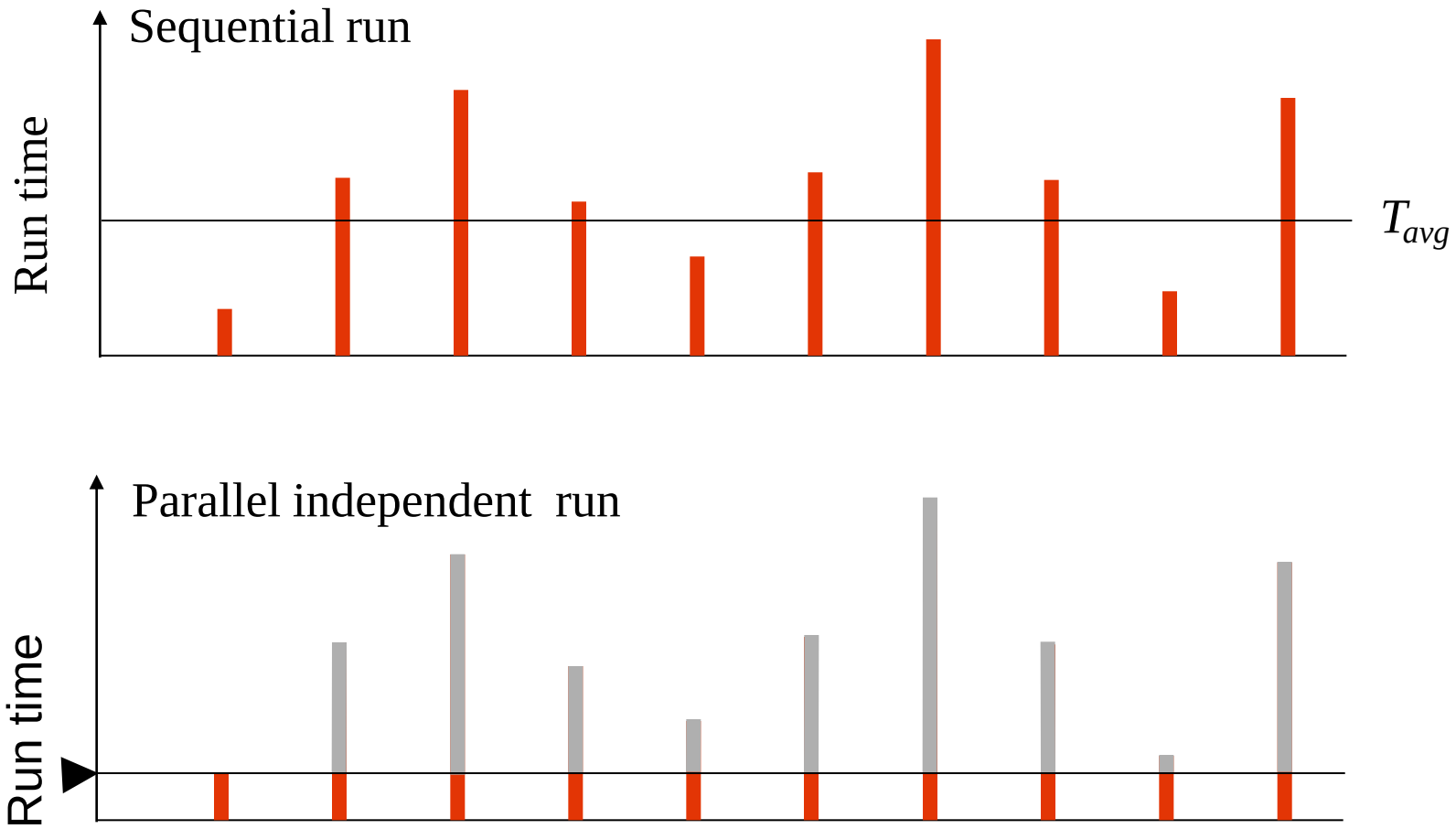


Parallel EC Model on GPU

- Parallel Independent Run Model
  - A variant of the coarse-grained model
  - Gives a lower bound performance of the coarse-grained model
  - Each sub-population runs on each MP independently
  - On an MP, individual level parallel run is performed



Effect of Parallel Independent Run



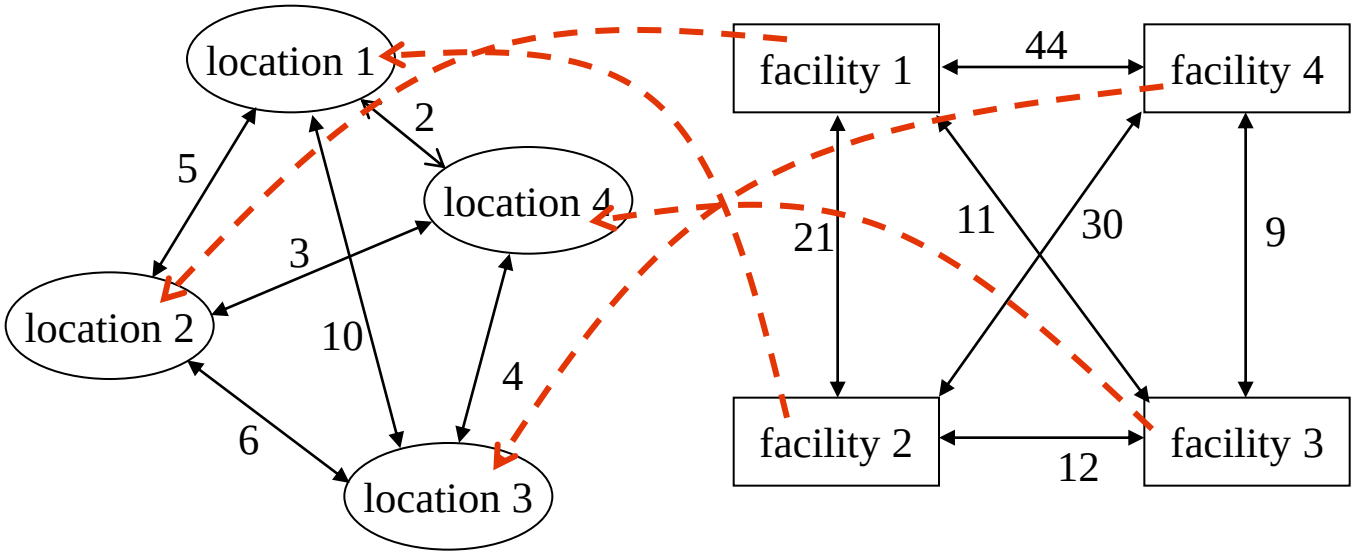
Obviously, $T_{avg} > T_{p,avg}$

Quadratic Assignment Problem (QAP)

- **One of the hardest** combinatorial optimization problem
- Problem size is at most 150
- Given *l* locations and *l* facilities, the task is to assign the facilities to the locations to minimize the cost
  - For each pair of locations *i* and *j*, the distance is d_{ij}
  - For each pair of facilities *r* and *s*, the flow is f_{rs}
  - The cost is defined as:

$$cost(\phi) = \sum_{i=1}^{l} \sum_{j=1}^{l} f_{ij} d_{\phi(i)\phi(j)}$$

An Example of QAP (l=4)



location

location	1	2	3	4
1	0	5	10	2
2	5	0	6	3
3	10	6	0	4
4	2	3	4	0

distance matrix d_{ij}

facility

facility	1	2	3	4
1	0	21	11	44
2	21	0	12	30
3	11	12	0	9
4	44	30	9	0

flow matrix f_{rs}

an assignment ϕ

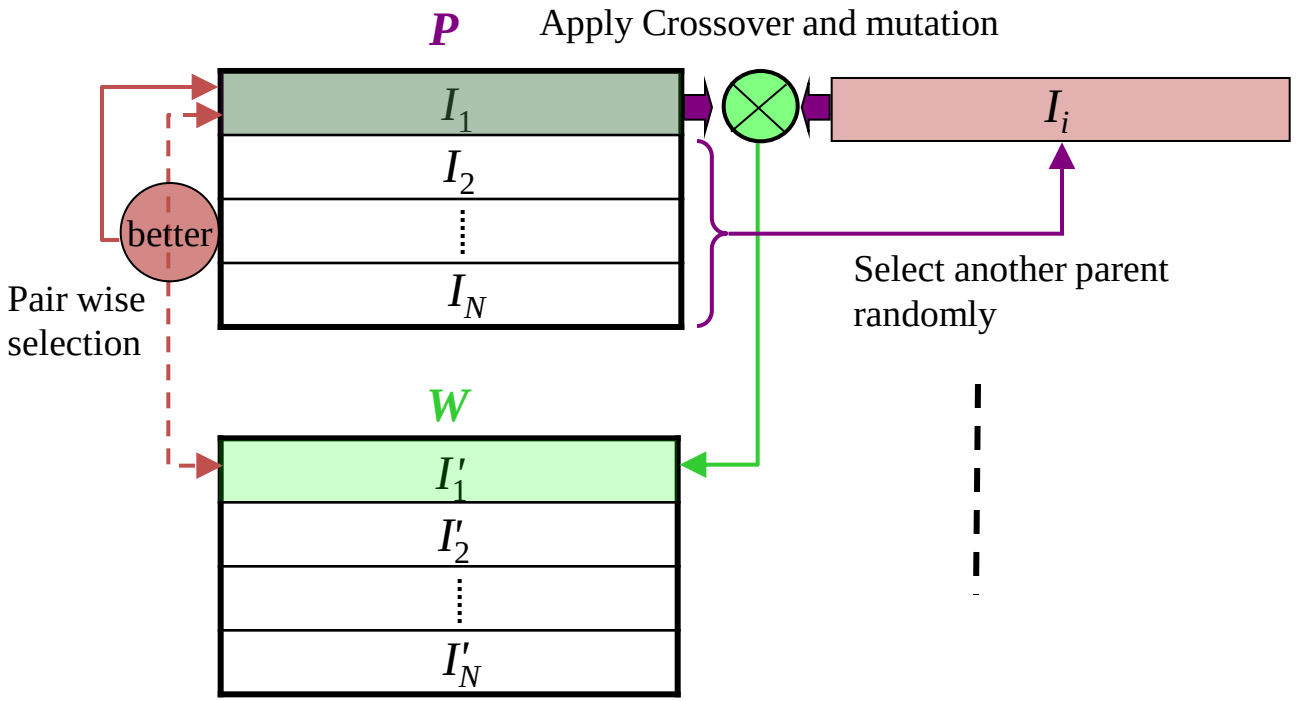
ϕ =	1	2	3	4
	2	1	4	3

$$cost(\phi) = \sum_{i=1}^{4}\sum_{j=1}^{4} f_{ij} d_{\phi(i)\phi(j)}$$

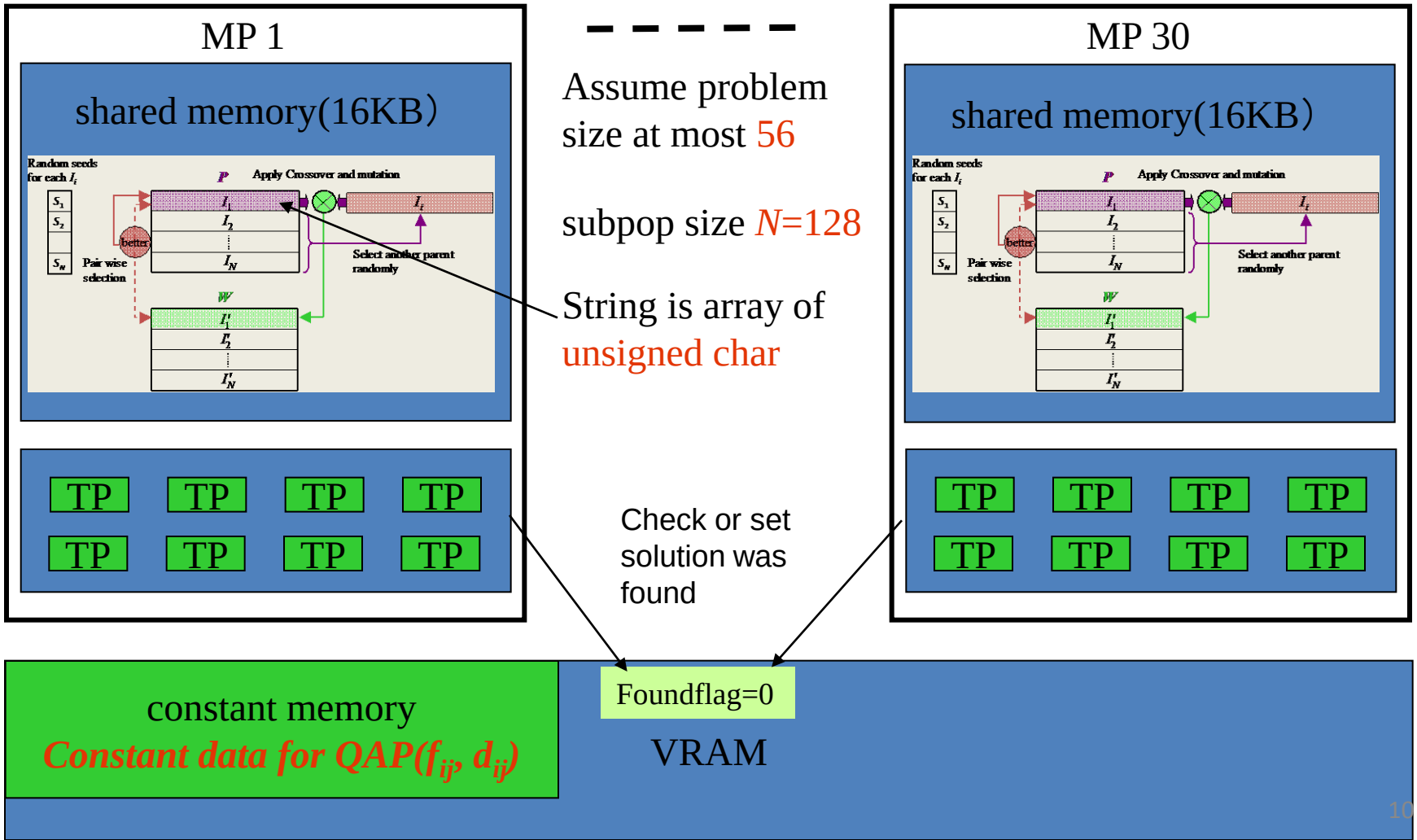
$$= 1524$$

The Base EC Model of a Sub-population

- We use population pool *P* and working pool *W*
- Each individual *i* (*i*=1,2,…, *N*) is processed independently of other individuals.
- Re-initialize if number of individuals which have current best functional value is greater than *N**0.6



Implementation Details on GPUs

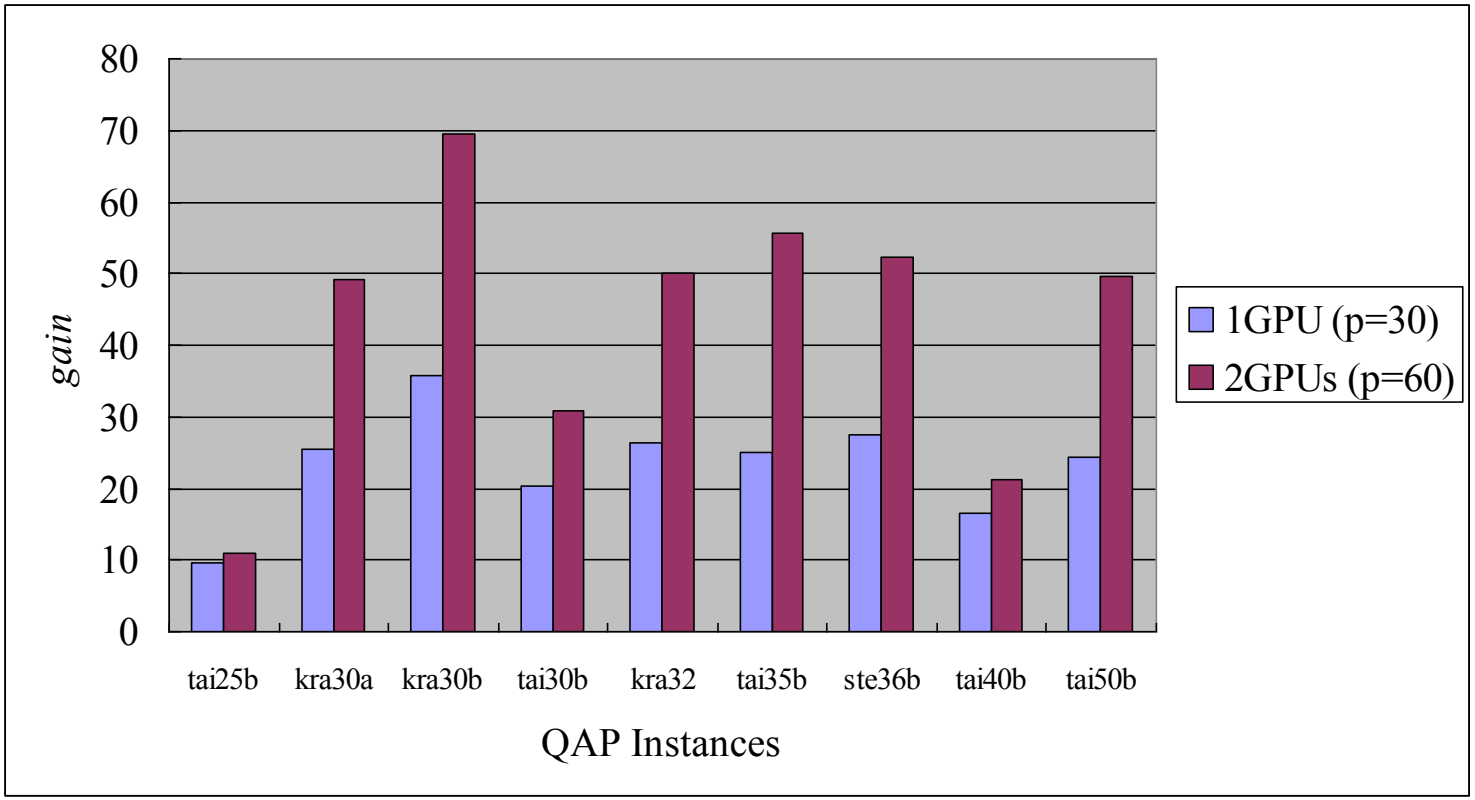


Experimental Conditions

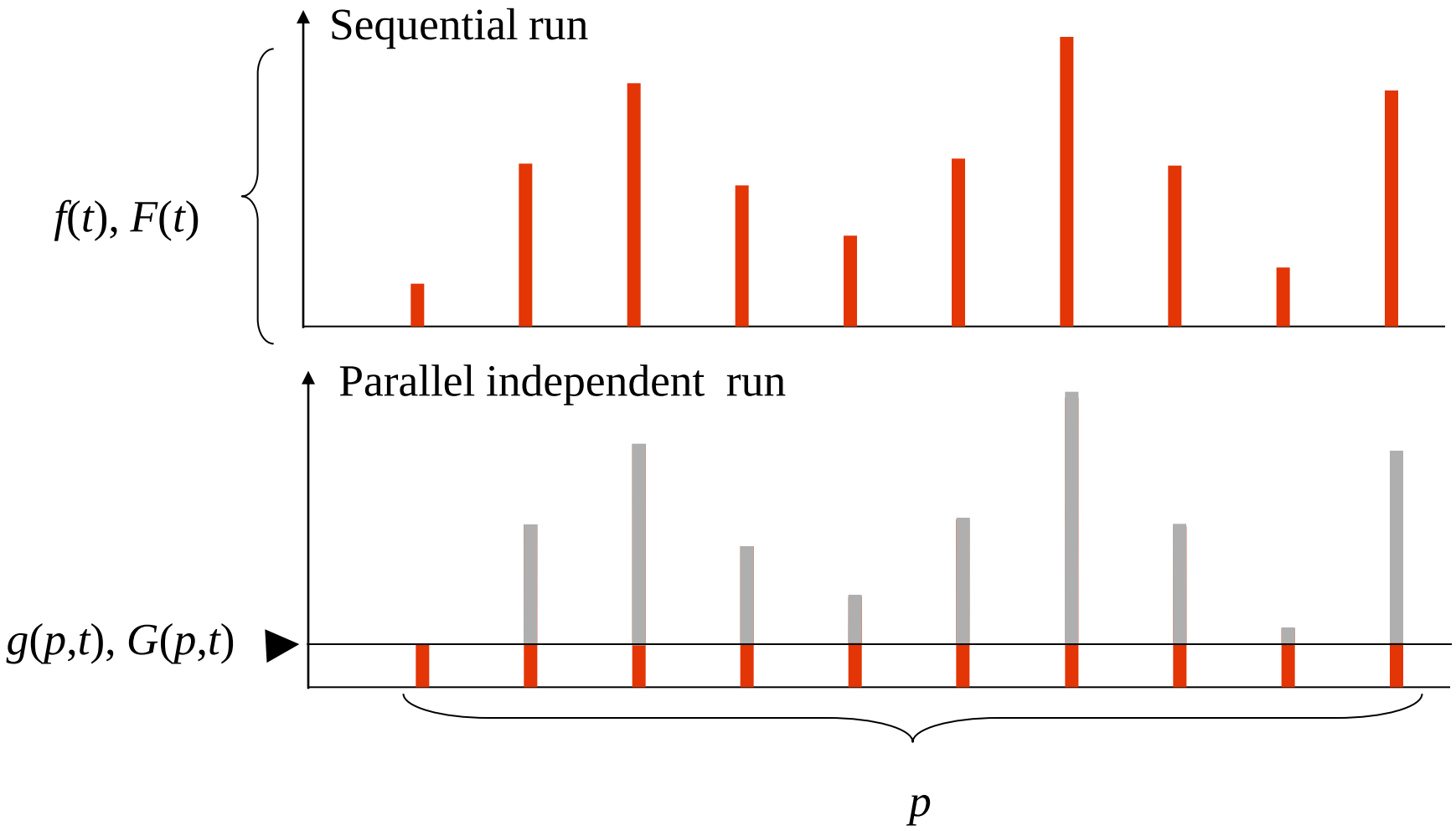
CPU	Intel Core i7 965
GPU	NVIDIA GeForce GTX285 (240 procs, VRAM 1GB)×２
OS	Windows XP
Compiler	Visual Studio 2005 with /O2
SDK	CUDA 2.3
Number of runs	30
Problem instances	tai25b, kra30a, kra30b, tai30b, kra32, tai35b, ste36b, tai40b, tai50b from QAPLIB

The run time gain obtained by *p*-block parallel runs to single block runs

- The values of *gain* are different from instance to instance
  - They are in the range [10, 35] for $p = 30$, and [10, 70] for $p = 60$,
  - and are nearly proportional to p, except for some instances



Run Time Estimation of Independent Parallel Run (1)



Run Time Estimation of Independent Parallel Run (2)

- Run time of parallel independent run with p blocks

$$F(t) = \int_{t=0}^{t} f(t)dt$$

$$G(p,t) = 1-(1-F(t))^{p}$$

$$g(p,t) = \frac{d}{dt}G(p,t)$$

$$= p(1-F(t))^{p-1} \cdot f(t)$$

$$M(p) = \int_{t=0}^{\infty} t \cdot p(1-F(t))^{p-1} \cdot f(t)dt$$

- Gain obtained by parallel independent run with p blocks

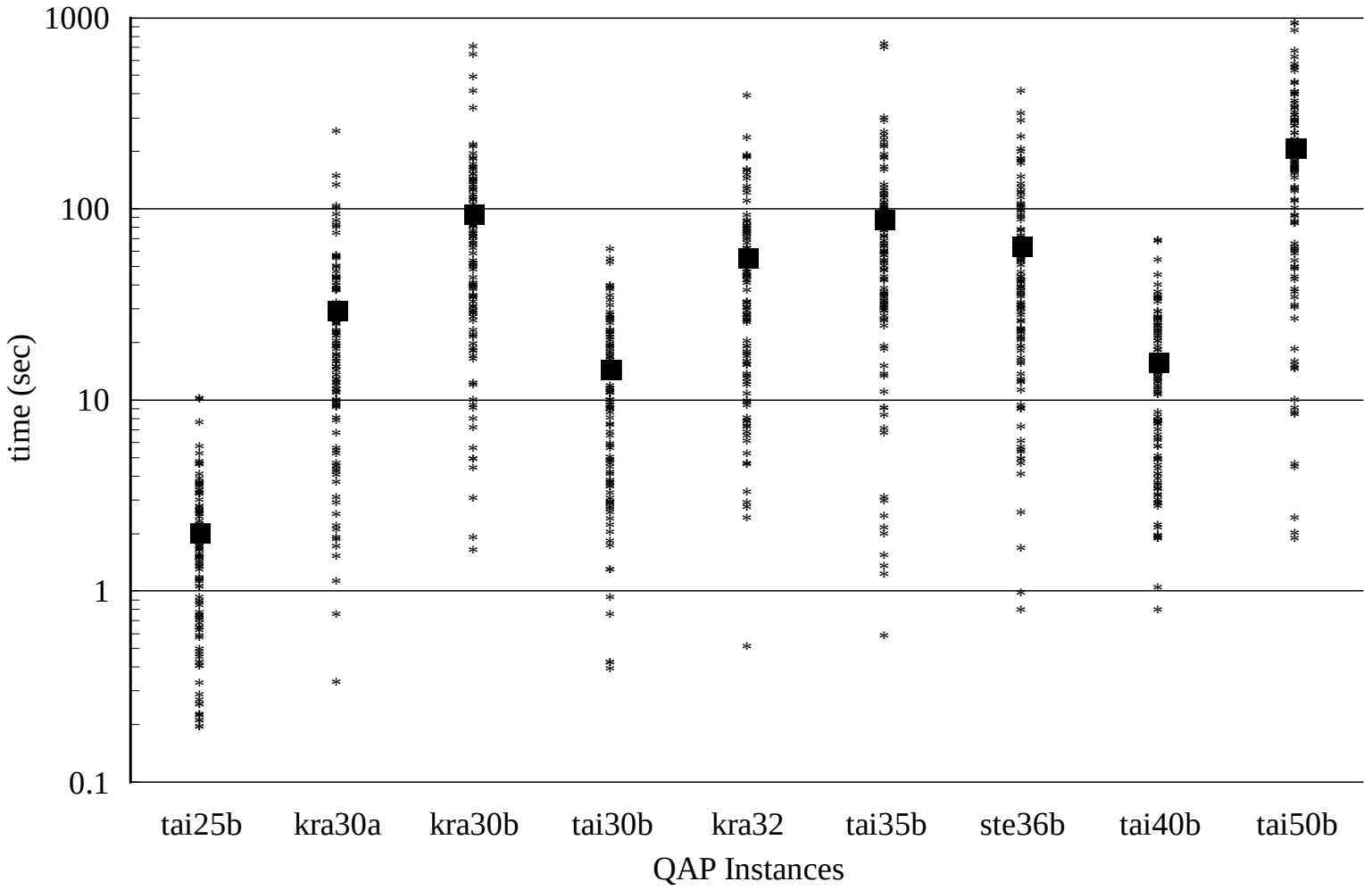
$$Gain_p = \frac{M(1)}{M(p)}$$

$$= \frac{\int_{t=0}^{\infty} t \cdot f(t)dt}{\int_{t=0}^{\infty} t \cdot p(1-F(t))^{p-1} f(t)dt}$$

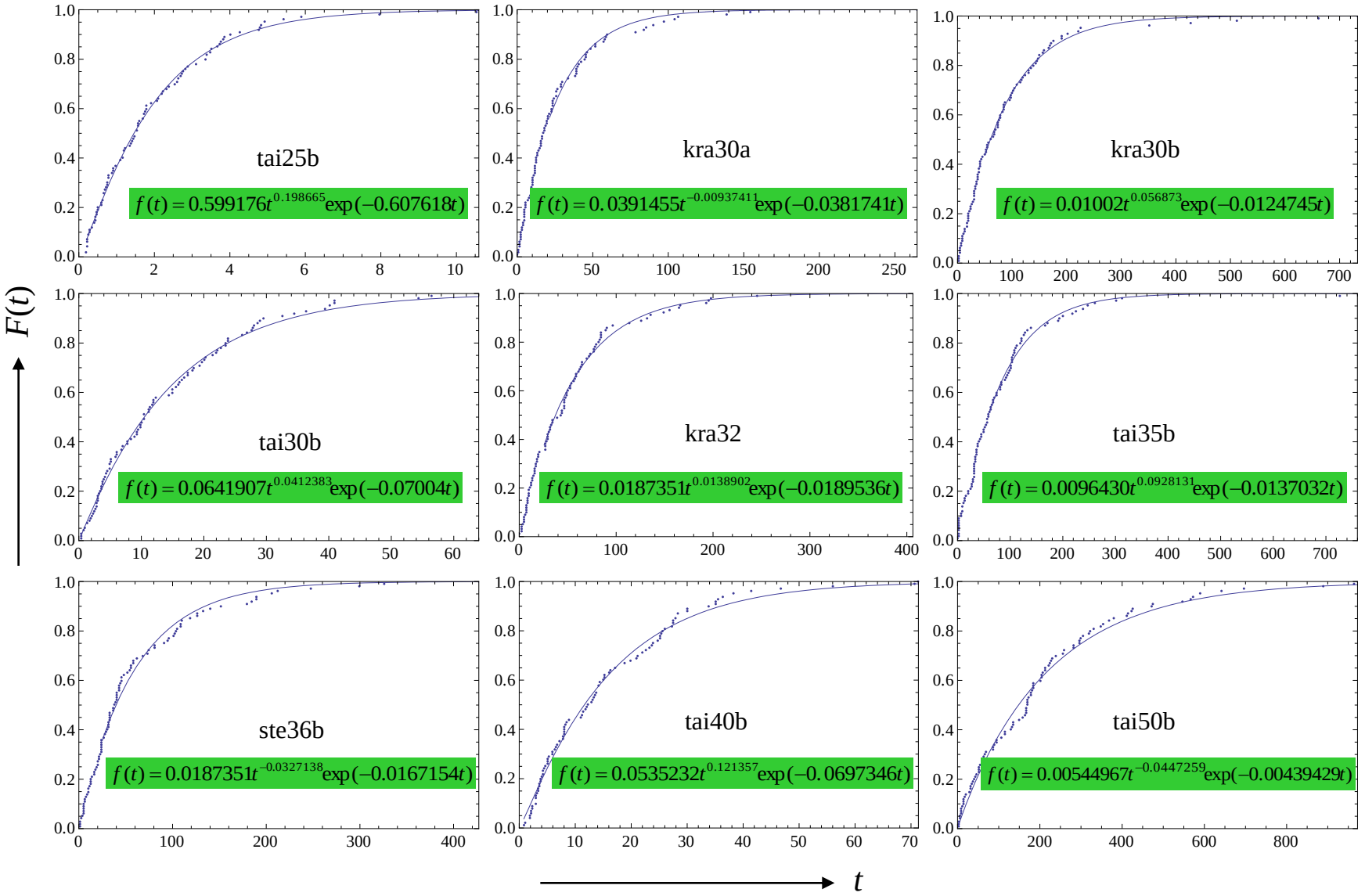
- Run time with single block run

$$M(1) = \int_{t=0}^{\infty} t \cdot f(t)dt$$

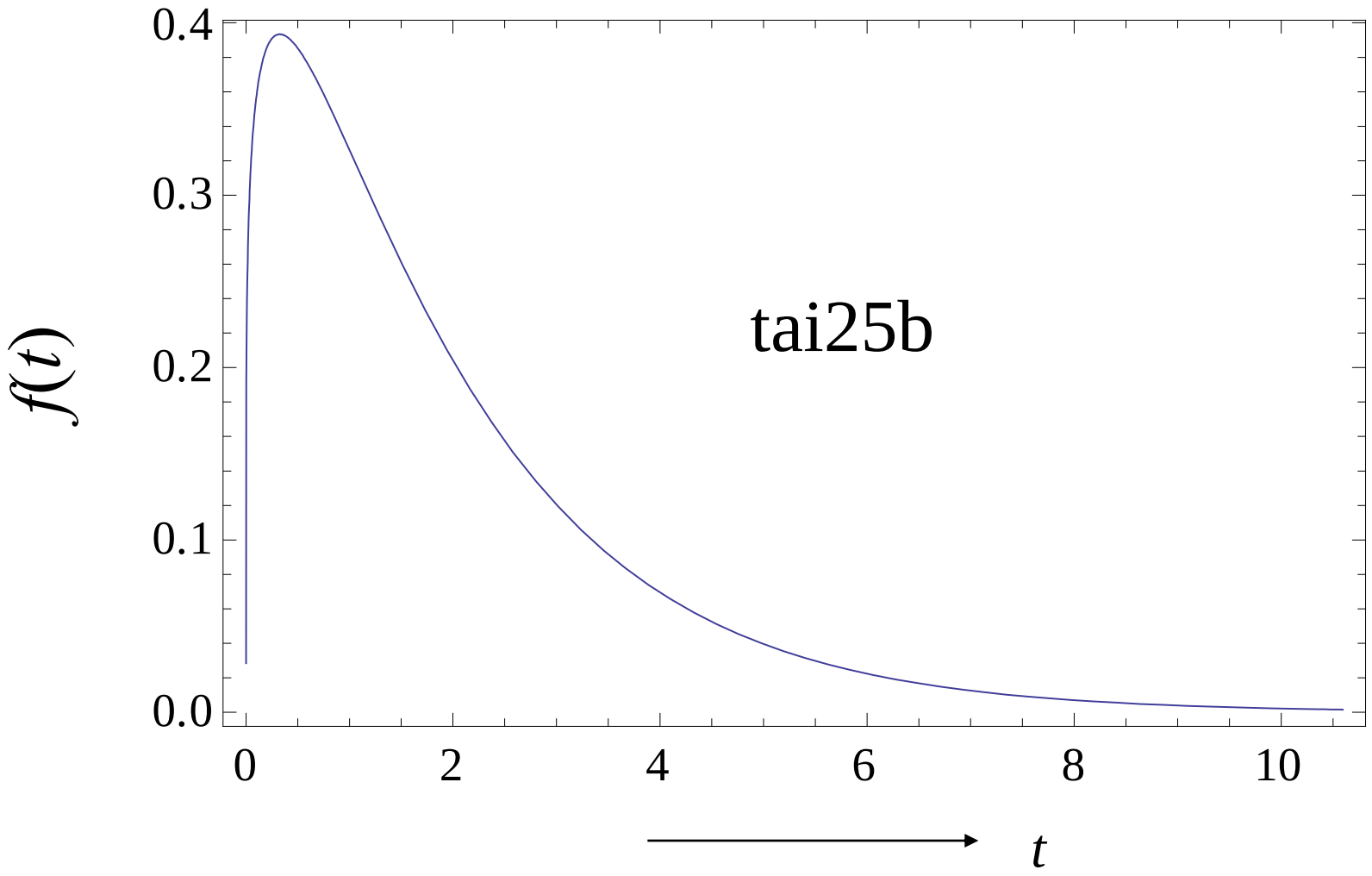
Run time distribution on a single block



Γ distribution reflects run time well



An example of Γ distribution



Comparison between Experimental and Analytical Results

GPU	Instances	tai25b			kra30a			kra30b		
	No of blocks p	GPU	Γ		GPU	Γ		GPU	Γ	
		$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p
GPU×1	1	2.02	-	-	34.25	-	-	113.69	-	-
	30	0.21	0.03	0.18	1.35	0.79	0.56	3.17	2.92	0.25
GPU×2	60	0.19	0.00	0.18	0.70	0.34	0.36	1.63	1.21	0.42
GPU	Instances	tai30b			kra32			tai35b		
	No of blocks p	GPU	Γ		GPU	Γ		GPU	Γ	
		$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p
GPU×1	1	14.31	-	-	56.18	-	-	92.08		-
	30	0.71	0.45	0.25	2.13	1.78	0.35	3.67	3.25	0.41
GPU×2	60	0.46	0.16	0.30	1.12	0.83	0.29	1.65	1.66	-0.01
GPU	Instances	ste36b			tai40b			tai50b		
	No of blocks p	GPU	Γ		GPU	Γ		GPU	Γ	
		$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p	$T_{p,avg}$	$M(P)$	Δ_p
GPU×1	1	70.82	-	-	19.07	-	-	212.55	-	-
	30	2.57	1.63	0.94	1.15	0.46	0.69	8.75	6.19	2.56
GPU×2	60	1.35	0.66	0.70	0.90	0.12	0.78	4.28	2.72	1.56

Comparison between GPU and CPU Computation

QAP instances	GPU Computaion		CPU Computation		*speedup*	
	GPU×1 ($T_{30,avg}$)	GPU×2 ($T_{60,avg}$)	CPU (T_{avg})	Population Size	GPU×1	GPU×2
tai25b	0.21	0.19	0.82	128	3.9	4.4
kra30a	1.35	0.70	6.64	1024	4.9	9.5
kra30b	3.17	1.63	25.20	128	7.9	15.4
tai30b	0.71	0.46	2.05	512	2.9	4.4
kra32	2.13	1.12	10.70	128	5.0	9.5
tai35b	3.67	1.65	12.16	512	3.3	7.4
ste36b	2.57	1.35	15.07	256	5.9	11.1
tai40b	1.15	0.90	4.44	512	3.9	5.0
tai50b	8.75	4.28	18.76	512	2.1	4.4

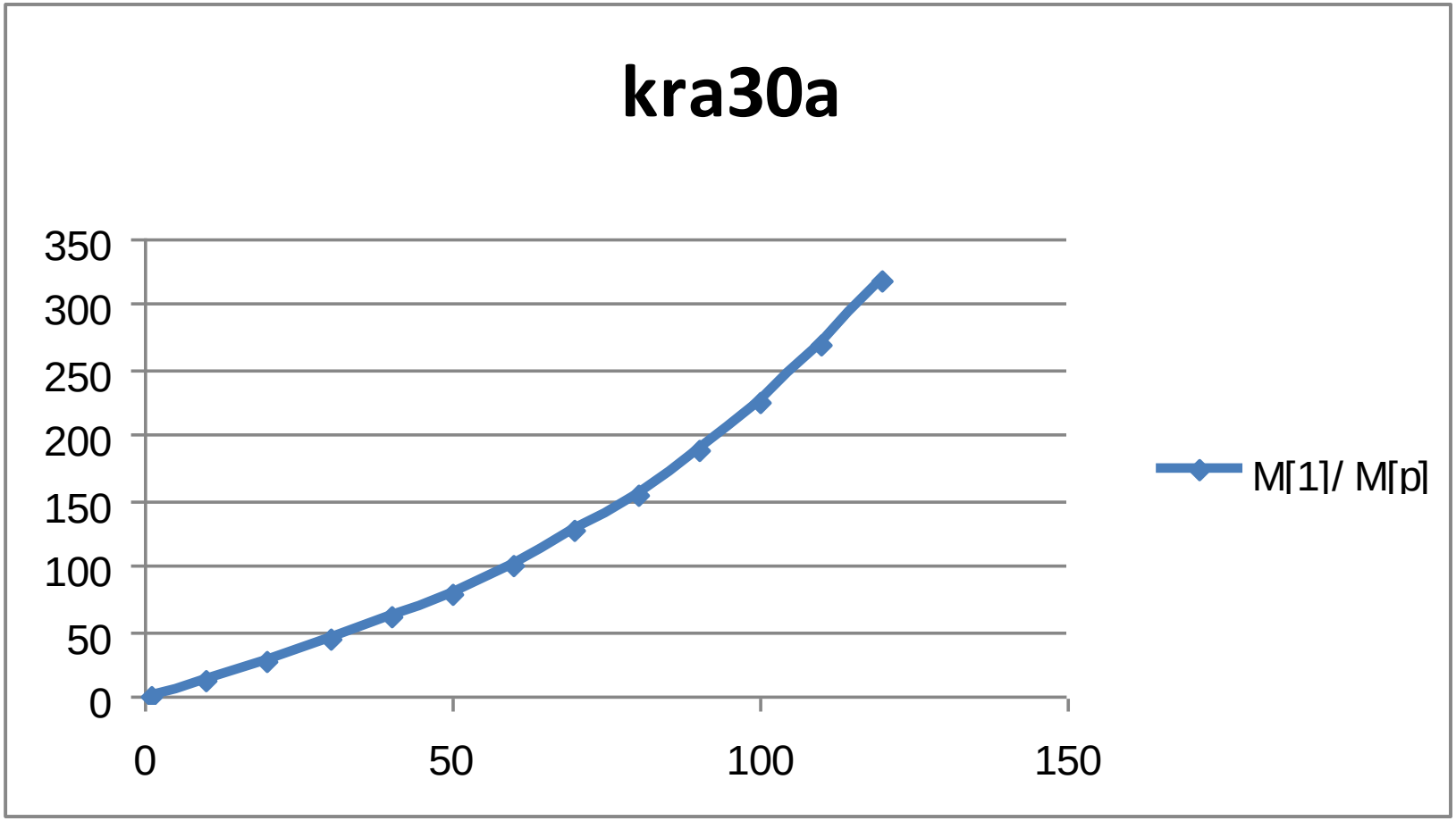
Values of $T_{30,avg}$, $T_{60,avg}$ *and* $M(P)$ are in seconds

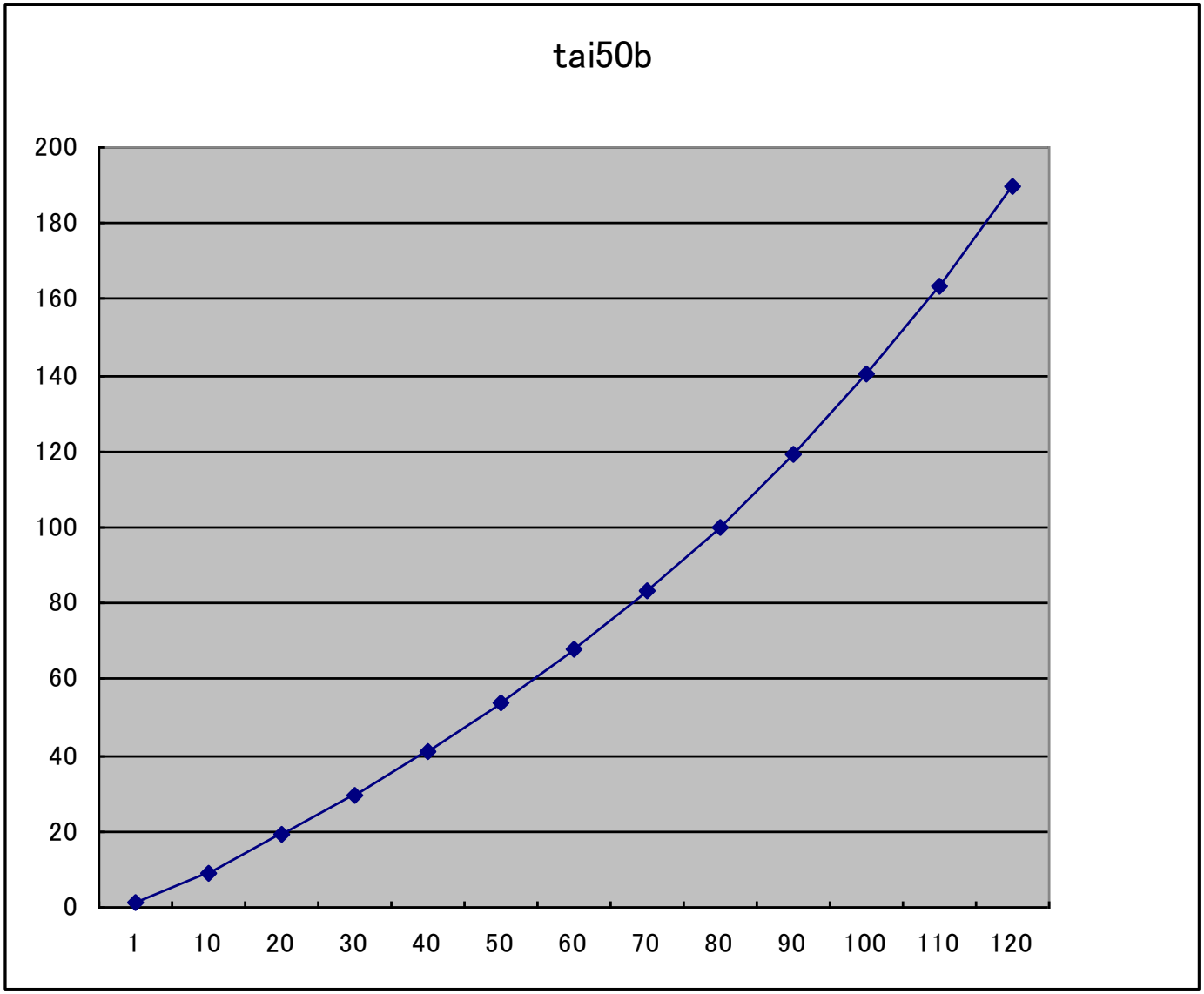
Conclusions

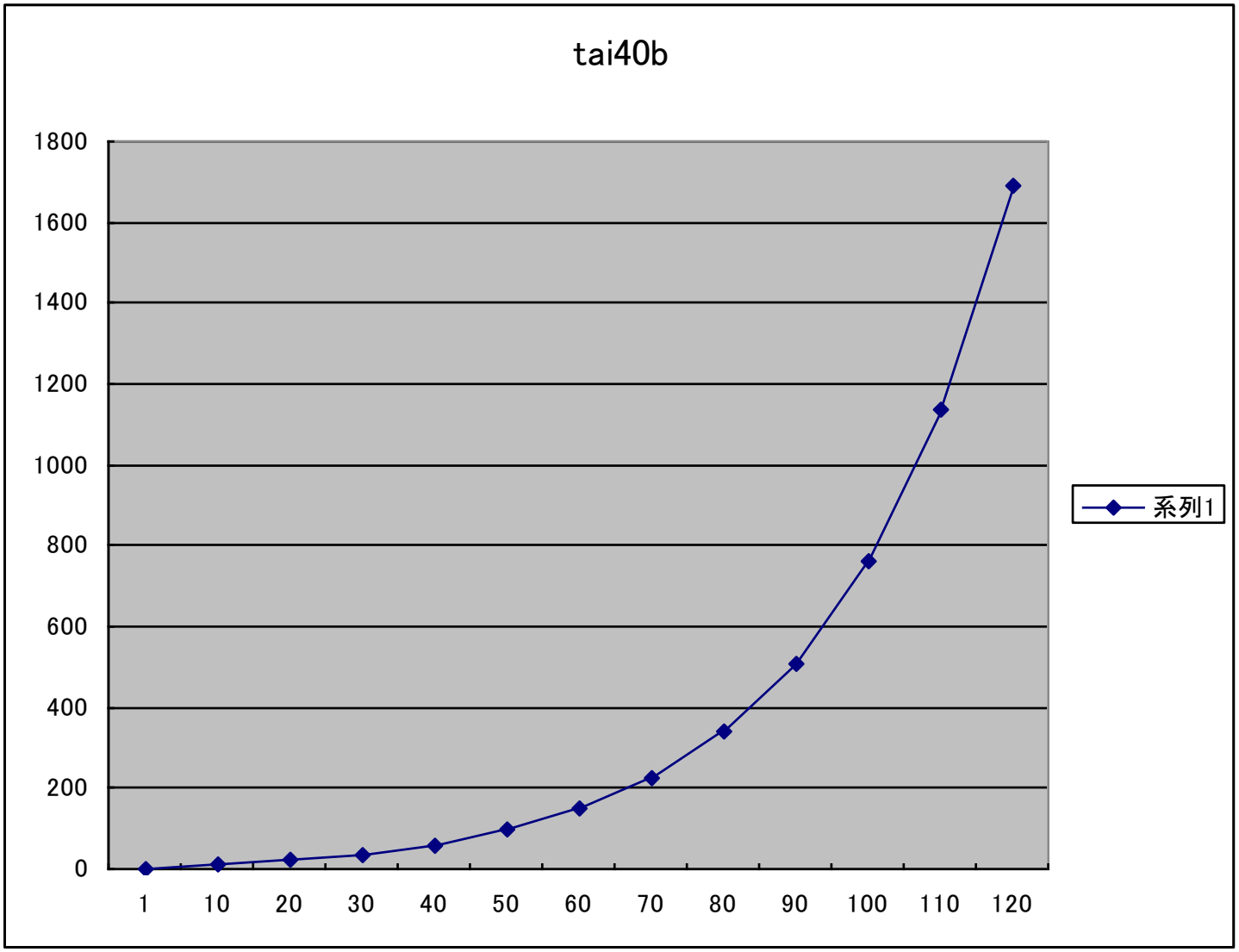
- We proposed an EA for solving QAPs with parallel independent runs using GPU computation and gave an analysis of the results
- In this parallel model, a set of small-size subpopulations was run in parallel in each block in CUDA independently
- With this scheme, we got a performance of GPU computation that is almost proportional to the number of equipped multiprocessors (MPs) in the GPUs
- We explained these computational results by performing statistical analysis
- Regarding performance comparison to CPU computations, GPU computation showed a speedup of x4.4 and x7.9 on average using a single GPU and two GPUs, respectively

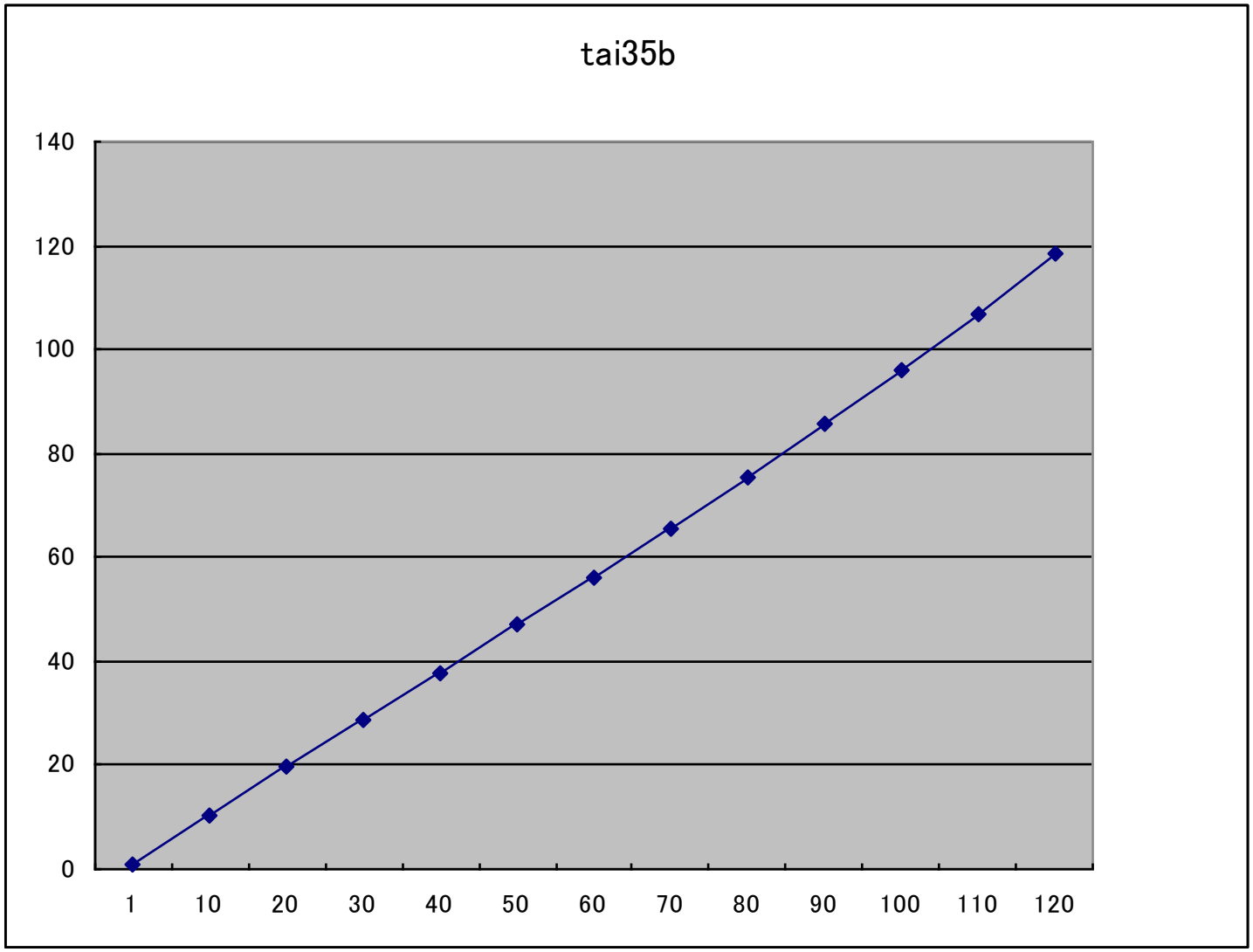
Future Work

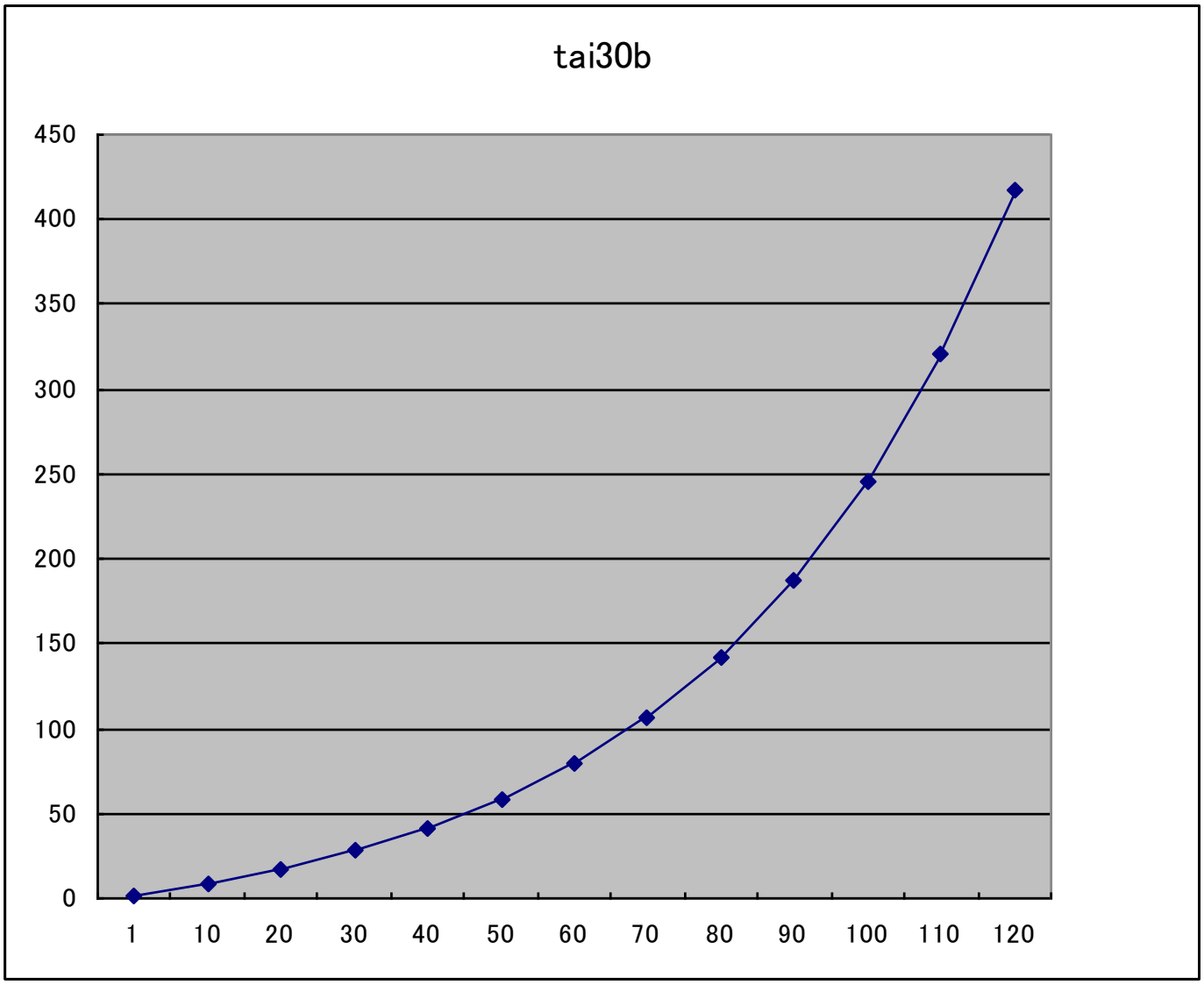
- To obtain higher speedup values, we need to improve the implementation of variation operator used in each thread in the blocks
- Each warp of 32 threads is essentially run in a SIMD fashion in a MP; high performance can only be achieved if all of a warp's threads execute the same instruction
- We can consider many parallel evolutionary models for GPU computation. To implement these models and analyze them remain for future work

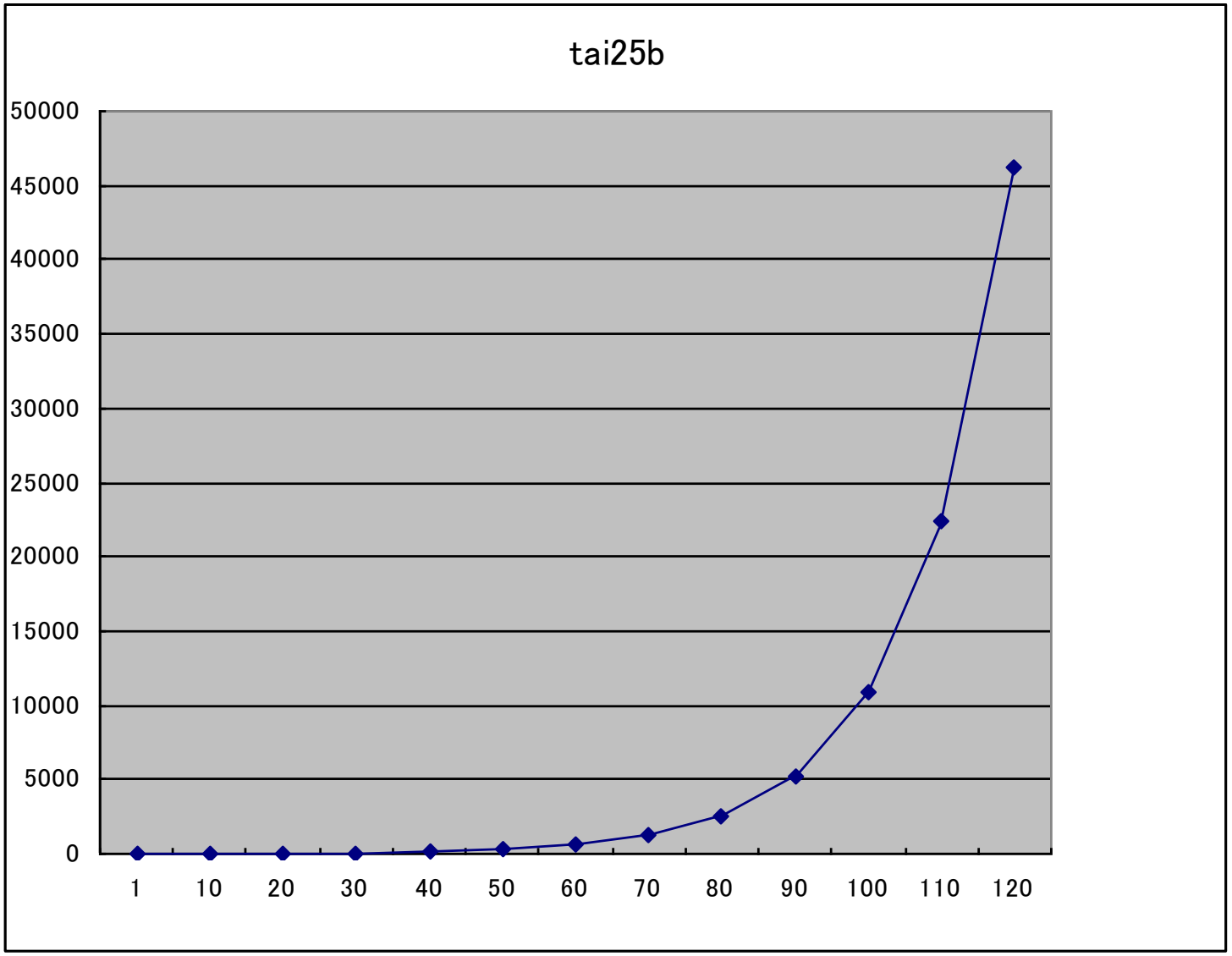

kra30a
350
300
250
200
150
100
50
0
0
50
100
150
M[1]/ M[p]

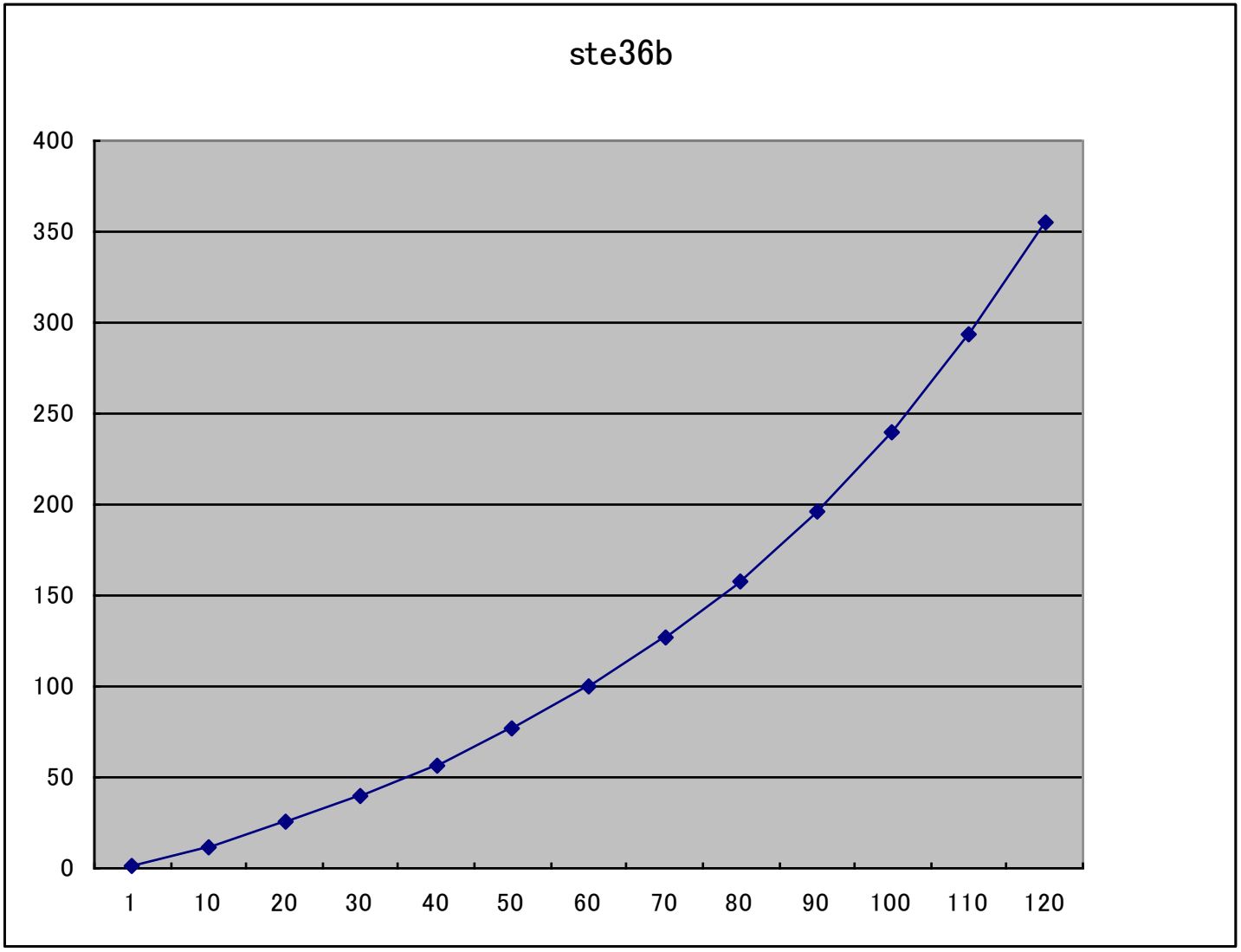

tai50b
200
180
160
140
120
100
80
60
40
20
0
1
10
20
30
40
50
60
70
80
90
100
110
120

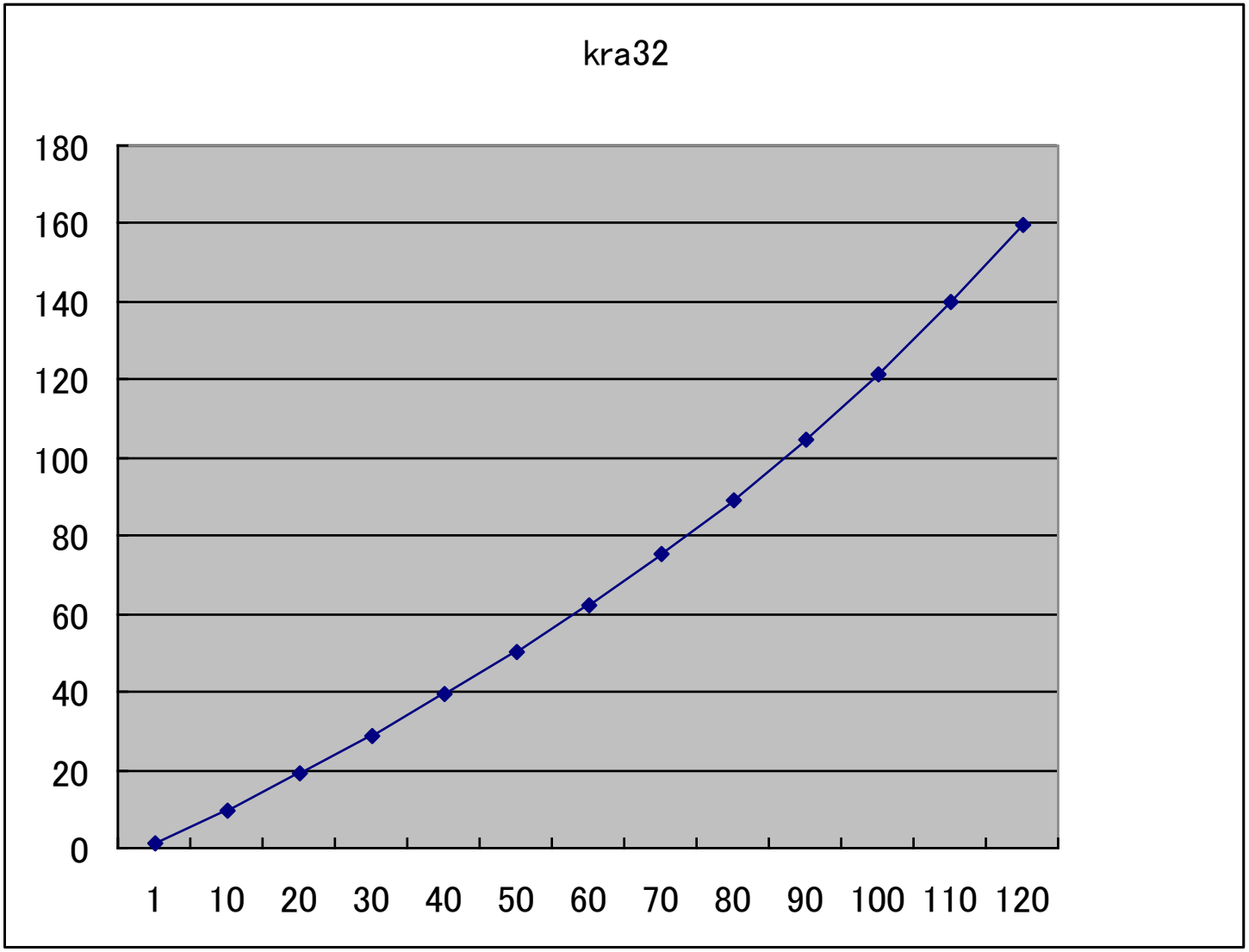

tai40b
1800
1600
1400
1200
1000
800
600
400
200
0
1
10
20
30
40
50
60
70
80
90
100
110
120
系列1

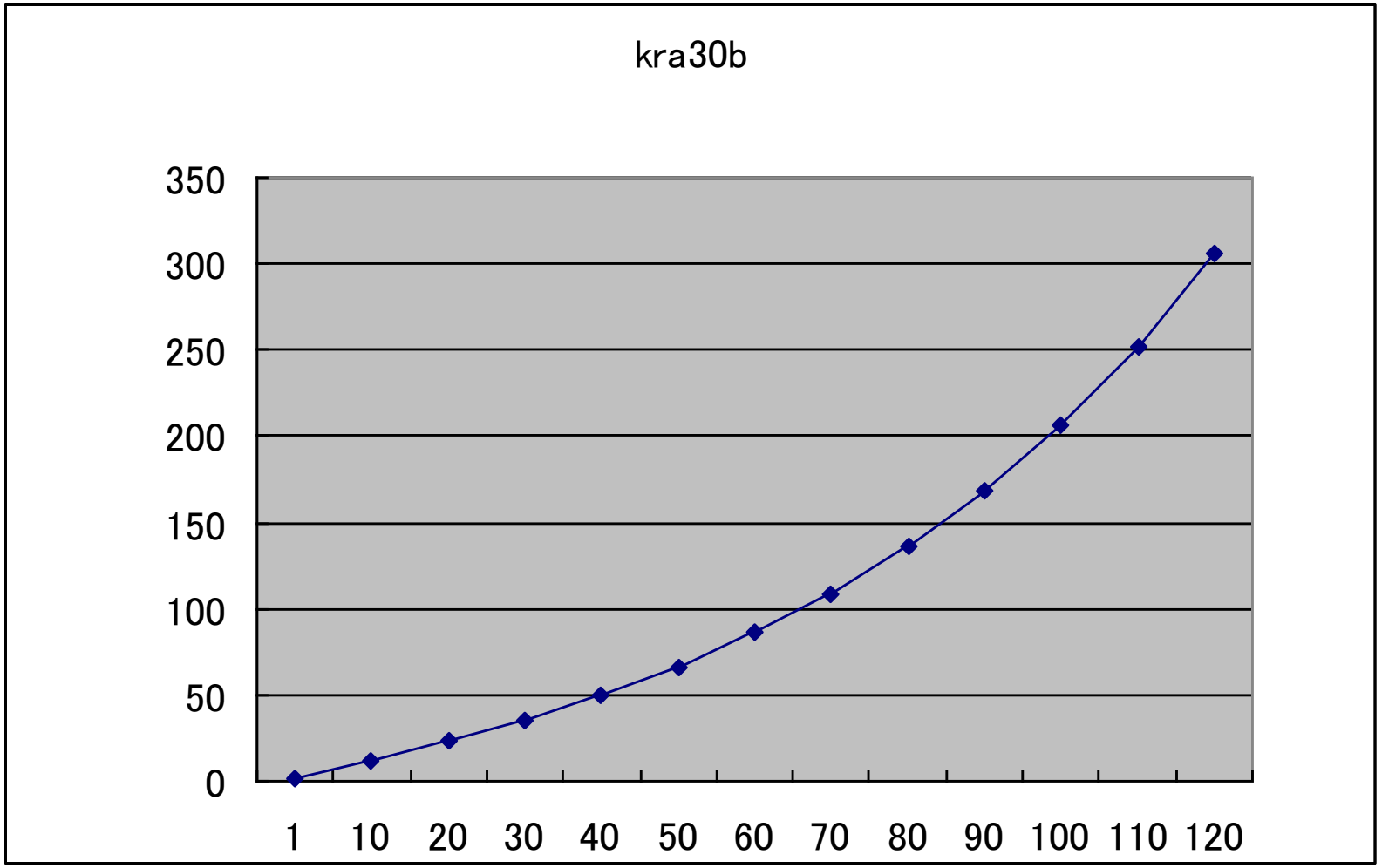

tai35b
140
120
100
80
60
40
20
0
1
10
20
30
40
50
60
70
80
90
100
110
120


tai30b
450
400
350
300
250
200
150
100
50
0
1
10
20
30
40
50
60
70
80
90
100
110
120


tai25b
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
1
10
20
30
40
50
60
70
80
90
100
110
120


ste36b
400
350
300
250
200
150
100
50
0
1
10
20
30
40
50
60
70
80
90
100
110
120


kra32
180
160
140
120
100
80
60
40
20
0
1
10
20
30
40
50
60
70
80
90
100
110
120


kra30b
350
300
250
200
150
100
50
0
1
10
20
30
40
50
60
70
80
90
100
110
120