


GPU Computing on Business

© 2010 Numerical Technologies Incorporated

<http://www.numtech.com/>

HPC向けに普及が進む GPU Computing





しかし...

A dramatic scene of a person in a dark suit standing in a field, holding an umbrella. The sky is dark and stormy, with several bright lightning bolts striking down. The person is silhouetted against the lighter, cloudy sky. The overall mood is one of risk and uncertainty.

ビジネスへの普及は遅い

それはなぜか...



GPU コンピューティングは速いんだぜ！

話が噛み合っていない

アカデミックのGPU Computingでは...



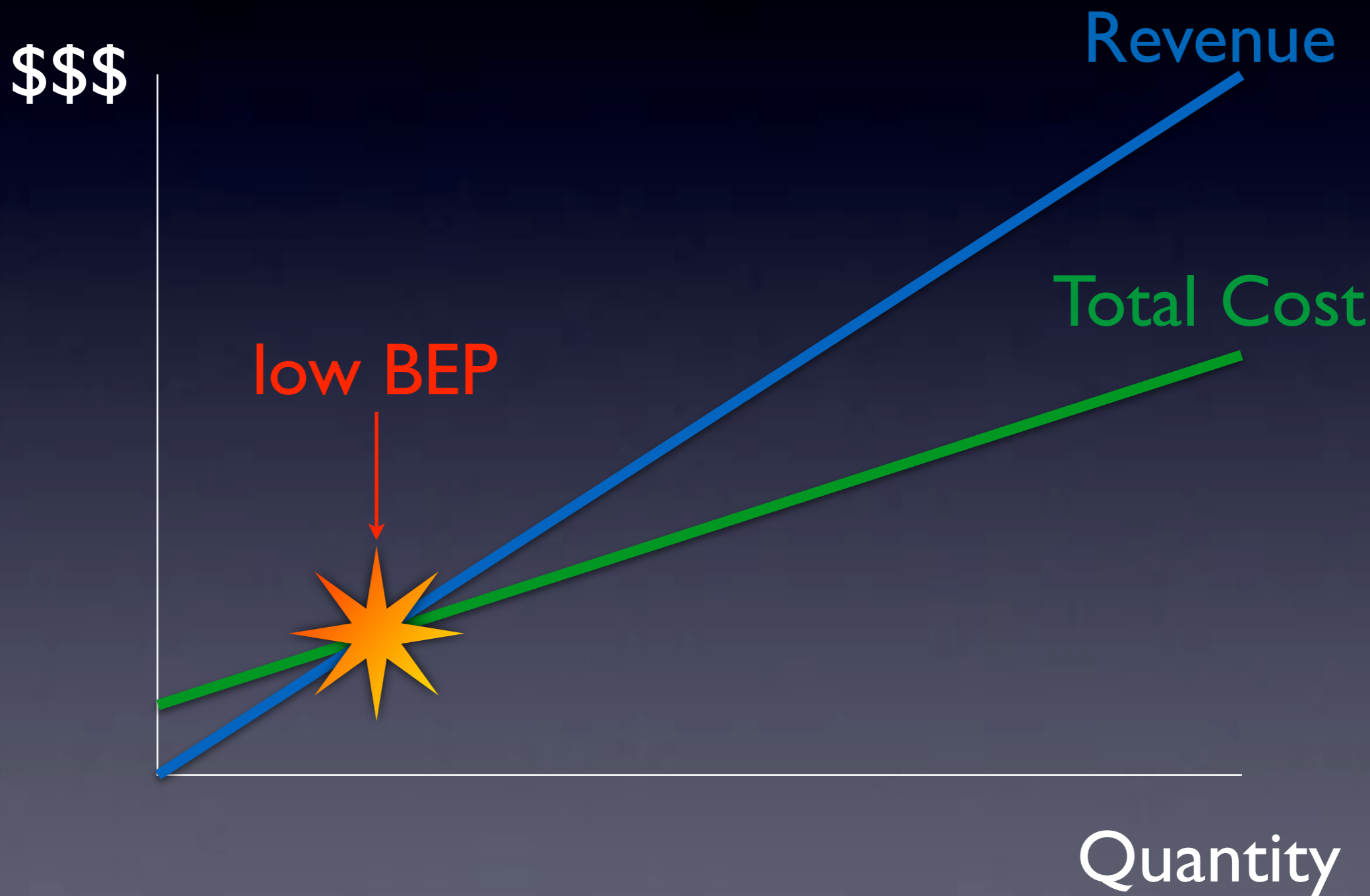
優秀な人が難しい学習の末に...



...辿り着く達人の技を駆使している

そんなのがCUDA Zoneにたくさんある。


学術系は損益分岐点が低い。投下コストを低く見積もれるからGPU Computingに投資できるのである



一方、ビジネスには...




GPU Computingの 普及を阻む壁がある



ちっとも速くならないぜ。俺のCUDAプログラムのどこが悪
いってんだ。

難しい学習

A photograph of a man with grey hair and a beard, wearing a dark suit and tie, sitting at a table and talking to a woman whose back is to the camera. The man is looking towards the woman. There are two speech bubbles overlaid on the image. The first speech bubble is in the top left, and the second is in the middle right. A large red banner with white text is at the bottom.

エンジニアの採用費、教育費、外部専門家への支払い...

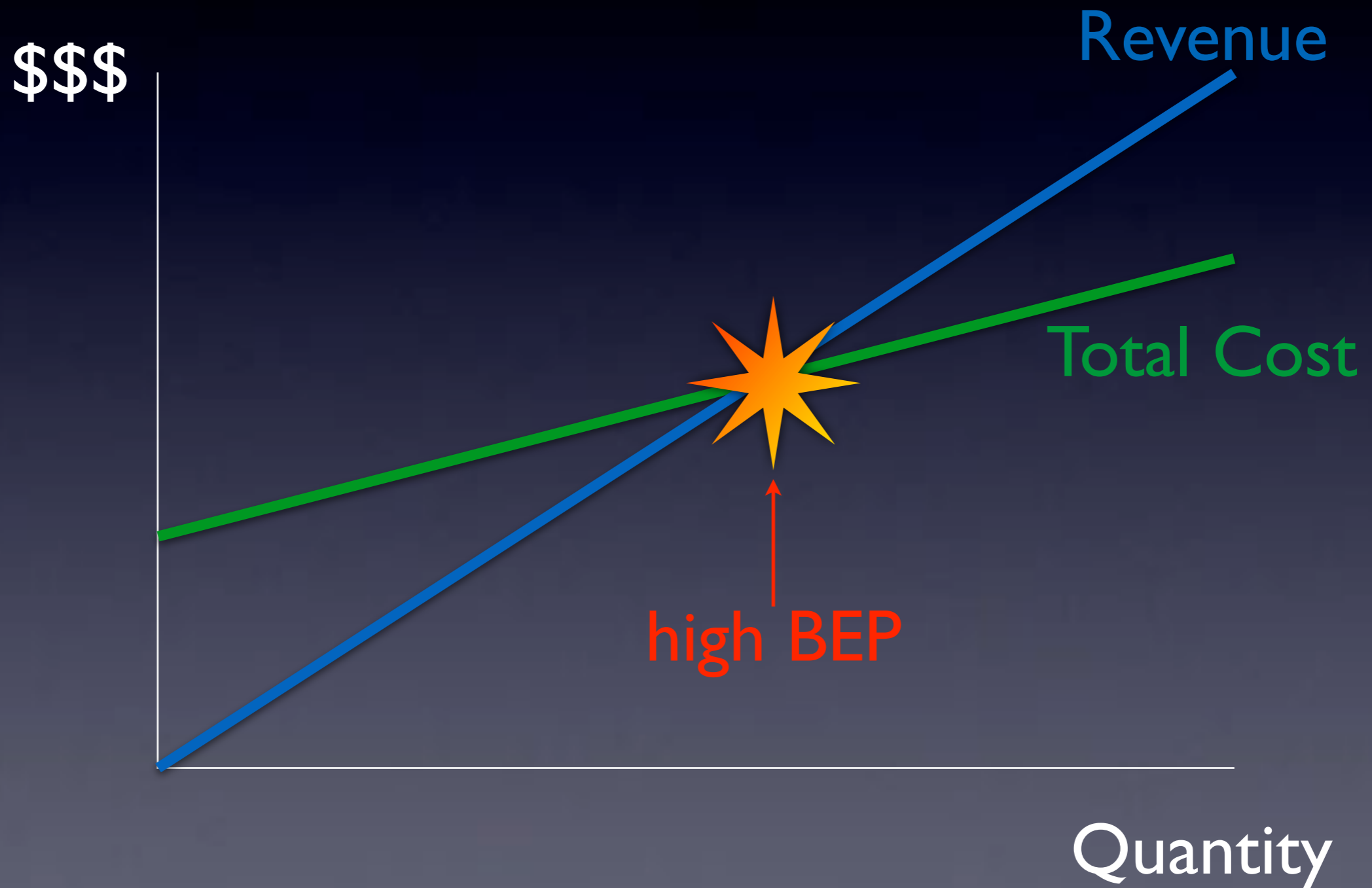
こりゃハードウェア増強した方がましだな。

技術以外のコスト負担



技術の継続性への不安

結局、多くのビジネスの視点から見ればGPU Computingは高コストすぎて敬遠してしまう





壁を破る方法

使い慣れた言語で開発

CUDA

C/C++

Java

Perl

PHP

C#

Python

もちろんnvcc、
Toolkit、SDKは
使いたくないし、
覚えたくもない




GPUを外しても
動くようにしてお
けば、Teslaがな
くなくても大丈夫



それにExcelから
でもGPUが使え
ればエンジニアも
不要じゃないか



A man with a beard, wearing a black top hat and a black tuxedo with a red bow tie, is looking intensely at the camera. He is holding a fan of playing cards in his left hand, which is wearing a white glove. The background is dark and moody.

そんなことが
できるか

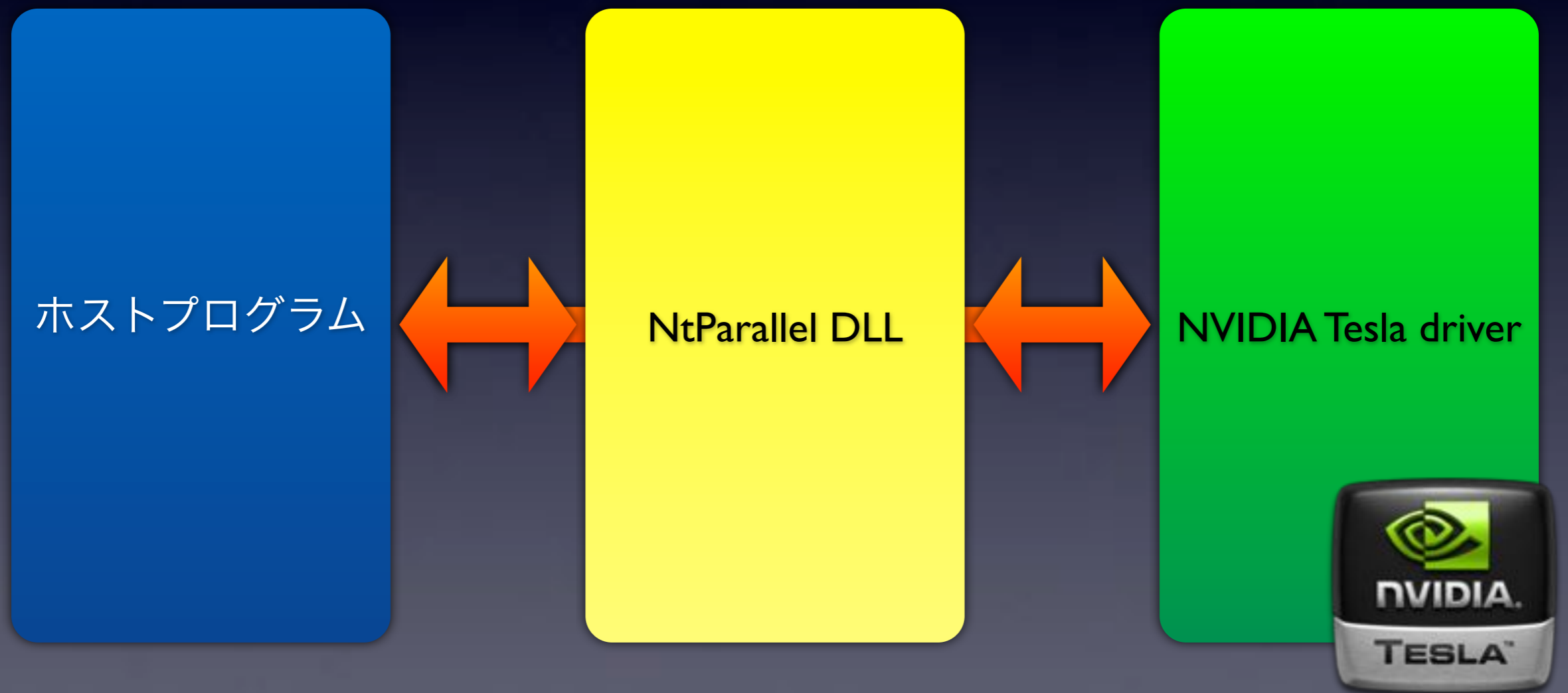


Idea



Lambda Expression

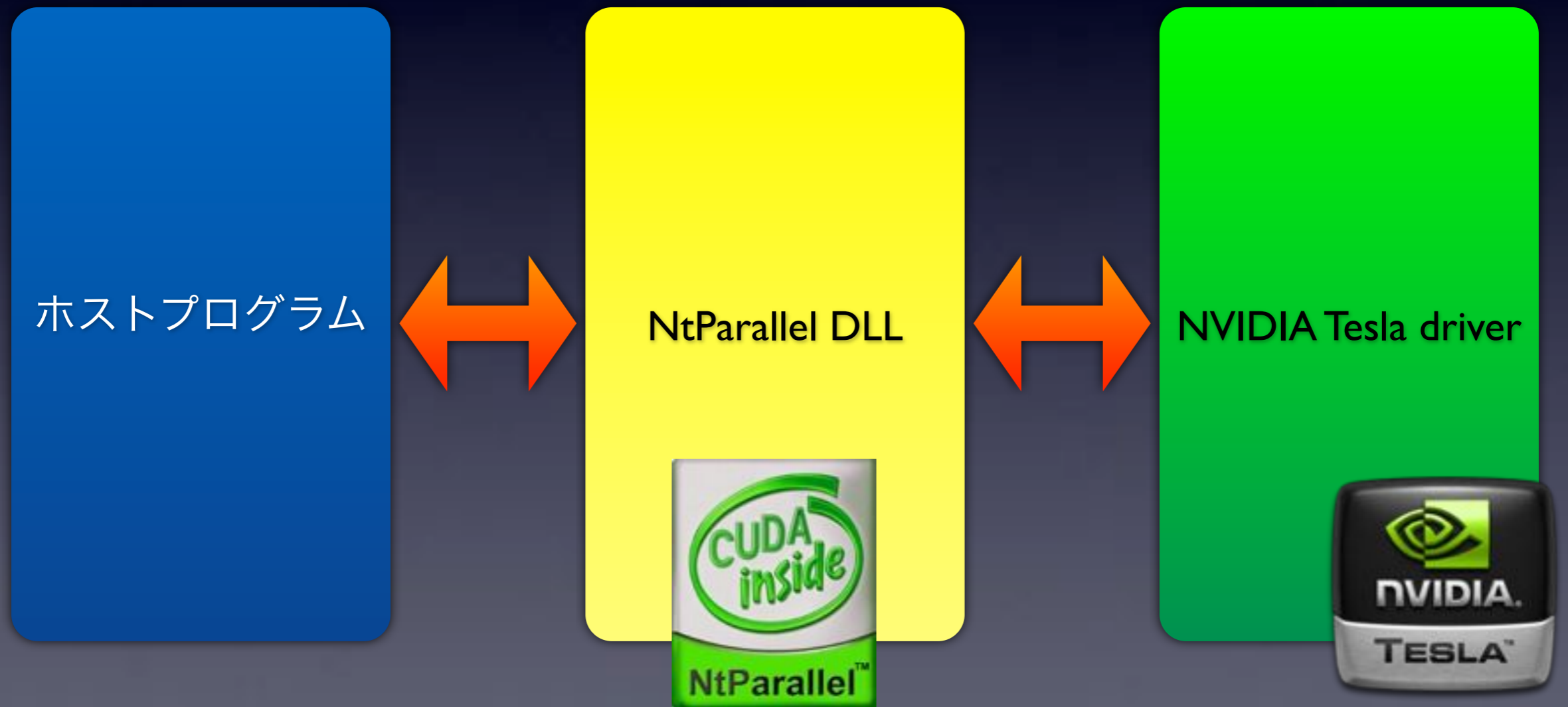
ホストプログラムから呼び出す ドライバ(NtParallel DLL)を用意



CUDAへの呼び出しをドライ
バ(NtParallel DLL)の中に隠蔽



CUDAへの呼び出しをドライバ(NtParallel DLL)の中に隠蔽



GPU呼び出し関数

nt_parallel_for

```
bool nt_parallel_for(  
    forループ内で実行するプログラム(string),  
    forループの繰り返し回数(int),  
    forループに与える引数配列1(void*),  
    forループに与える引数配列2(void*),  
    ...  
)
```


元のプログラム



```
// Black Scholes Option Formula Batch Processing Demo.
void batch_black_scholes_pricer(
    int array_size,
    double* o_data,
    double* r_data,
    double* sigma_data,
    double* s_data,
    double* k_data,
    double* t_data
)
{
    // for-loop program.
    for (int i = 0; i < array_size; ++i) {
        double R = r_data[i];
        double Sigma = sigma_data[i];
        double S = s_data[i];
        double K = k_data[i];
        double T = t_data[i];
        double rt = R * T;
        double sigmaSqrtT = Sigma * sqrt(T);
        double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;
        o_data[i] = S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);
    }
    // now you have output in o_data.
}
```

よくあるforループに注目



```
// Black Scholes Option Formula Batch Processing Demo.
void batch_black_scholes_pricer(
    int array_size,
    double* o_data,
    double* r_data,
    double* sigma_data,
    double* s_data,
    double* k_data,
    double* t_data
)
{
    // for-loop program.
    for (int i = 0; i < array_size; ++i) {
        double R = r_data[i];
        double Sigma = sigma_data[i];
        double S = s_data[i];
        double K = k_data[i];
        double T = t_data[i];
        double rt = R * T;
        double sigmaSqrtT = Sigma * sqrt(T);
        double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;
        o_data[i] = S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);
    }
    // now you have output in o_data.
}
```


forループの中を文字列(ラムダ式)に変換

```
...  
for (int i = 0; i < array_size; ++i) {  
    double R = r_data[i];  
    double Sigma = sigma_data[i];  
    double S = s_data[i];  
    double K = k_data[i];  
    double T = t_data[i];  
    double rt = R * T;  
    double sigmaSqrtT = Sigma * sqrt(T);  
    double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;  
    o_data[i] = S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);  
}  
...
```



```
...  
string code = "  
    [](int i, double R, double Sigma, double S, double K, double T) => double {  
        double rt = R * T;  
        double sigmaSqrtT = Sigma * sqrt(T);  
        double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;  
        return S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);  
    }  
";  
...
```



ホスト言語から呼び出す



```
// Black Scholes Option Formula Batch Processing Demo.
void batch_black_scholes_pricer(
    int array_size,
    double* o_data,
    double* r_data,
    double* sigma_data,
    double* s_data,
    double* k_data,
    double* t_data
)
{
    // for-loop program.
    string code = "
        [](int i, double R, double Sigma, double S, double K, double T) => double {
            double rt = R * T;
            double sigmaSqrtT = Sigma * sqrt(T);
            double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;
            return S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);
        }
";
    // call the GPU.
    nt_parallel_for(code, array_size, o_data, r_data, sigma_data, s_data, k_data, t_data);
    // now you have output in o_data.
}
```


Excelからさえ呼び出し可能

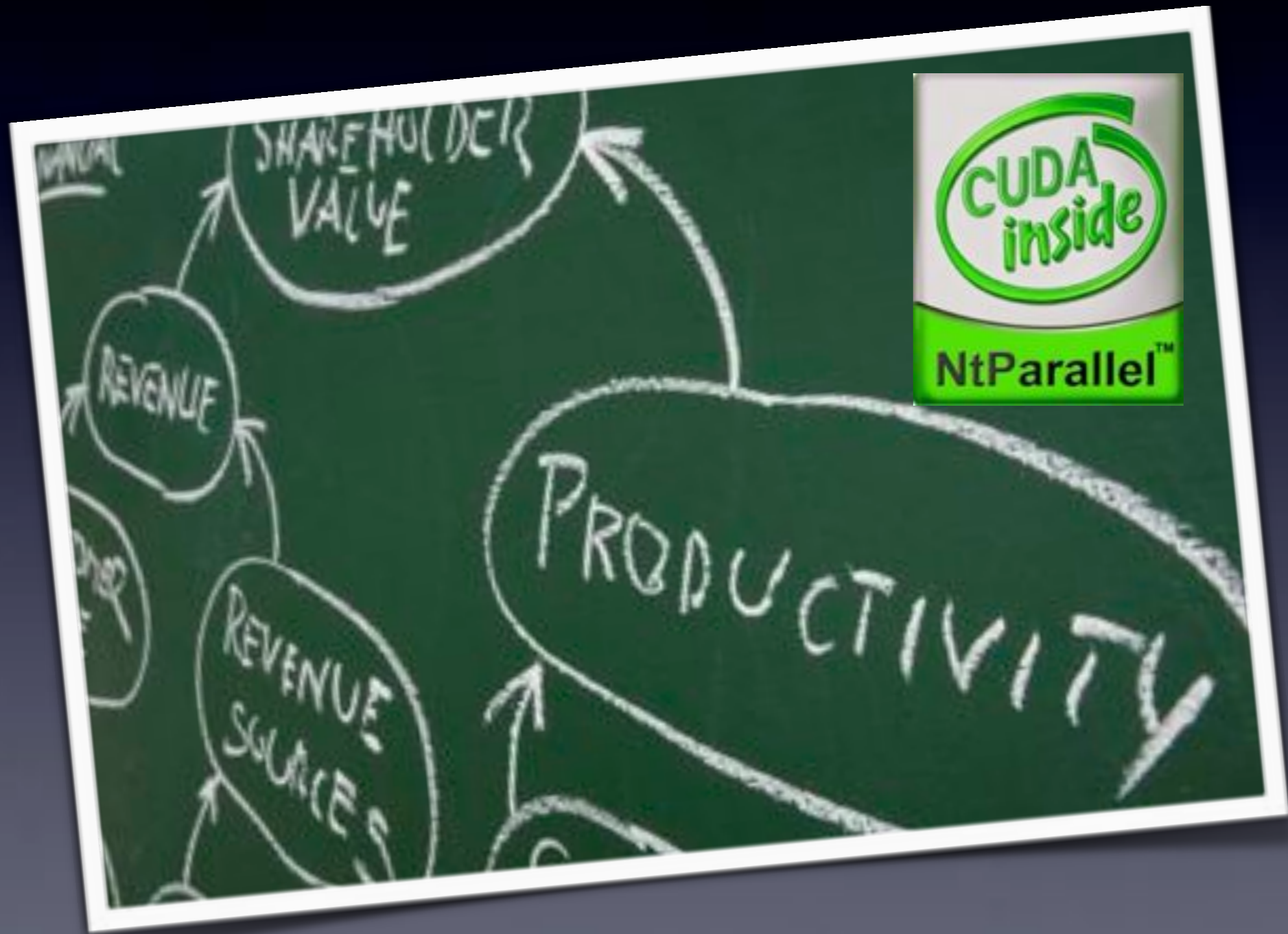
The screenshot shows the Microsoft Excel interface with a VBA function defined in the formula bar. The function is named `=NPV_PARALLEL_FOR` and takes four arguments: `int i`, `double rate`, `double pay`, and `double t`. The function returns a `double` value. The code is as follows:

```
=NPV_PARALLEL_FOR([int i, double rate, double pay, double t])=>double [  
int j;  
int term = 12* Cint([rate]);  
double cf = -pay;  
double df = 0.0;  
double r = 1.0 / (1.0 + rate/12);  
for (j=0; j<term; j=j+1) {  
df = (df + 1.0) * r;  
}  
return - cf * df;  
], A2:C101)
```

A red arrow points to the function definition in the formula bar, with the text "GPU code is here!" next to it. A "CUDA inside NtParallel" logo is also visible in the bottom right corner of the screenshot.

	A	B	C	D	F
1	Rate	monthlyPayment	term	Excel PV function	GPU (using NPV_PARALLEL_FOR)
2	0.0001	100000	35	¥41,926,411	¥41,926,411
3	0.0002	100000	35	¥41,852,995	¥41,852,995
4	0.0003	100000	35	¥41,779,750	¥41,779,750
5	0.0004	100000	35	¥41,706,677	¥41,706,677
6	0.0005	100000	35	¥41,633,775	¥41,633,775
7	0.0006	100000	35	¥41,561,043	¥41,561,043
8	0.0007	100000	35	¥41,488,481	¥41,488,481
9	0.0008	100000	35	¥41,416,089	¥41,416,089
10	0.0009	100000	35	¥41,343,865	¥41,343,865
11	0.001	100000	35	¥41,271,811	¥41,271,811
12	0.0011	100000	35	¥41,199,924	¥41,199,924

これで難解な学習は不要、
生産性は10倍だ



コピー程度の作業でGPUコードを元のCPUコードに戻せる



GPUコードに
高速化

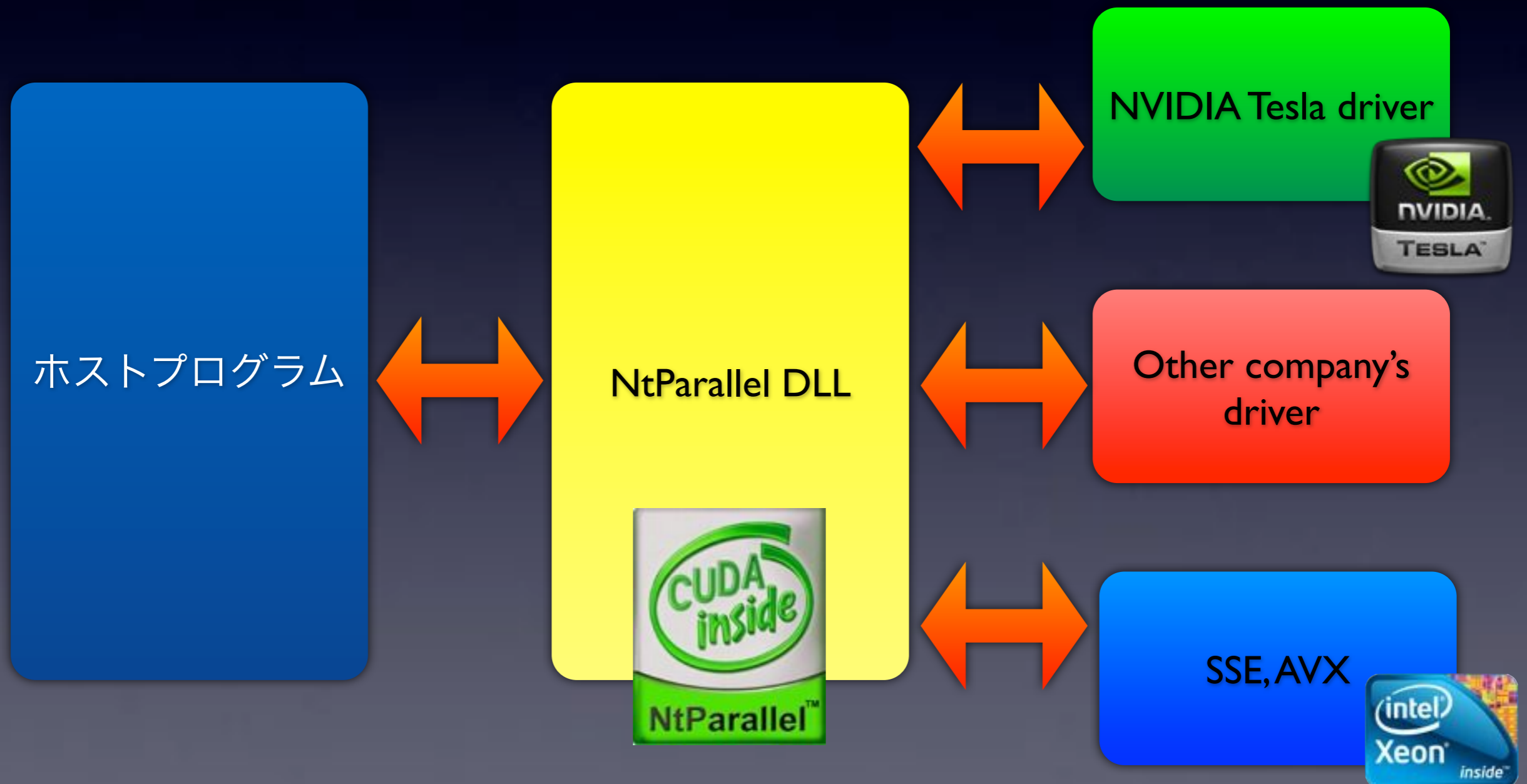


簡単に元
に戻せる



同じインターフェイスでNVIDIA以外のデバイス呼び出しが可能になる

技術的ヘッジ





Demonstration

Tesla C2050 vs. Xeon X5670

Scientific

Wall St.

1. Mandelbrot
2. Kirkwood Gaps
3. Wavelet Analysis
4. Binomial Tree Option Model
5. Black Scholes Option Model
6. Housing Loan Calculation

Boring

Accounting Stuff

Test Machine: HP Z800/CT Workstation

• Windows 7 Professional 64bit

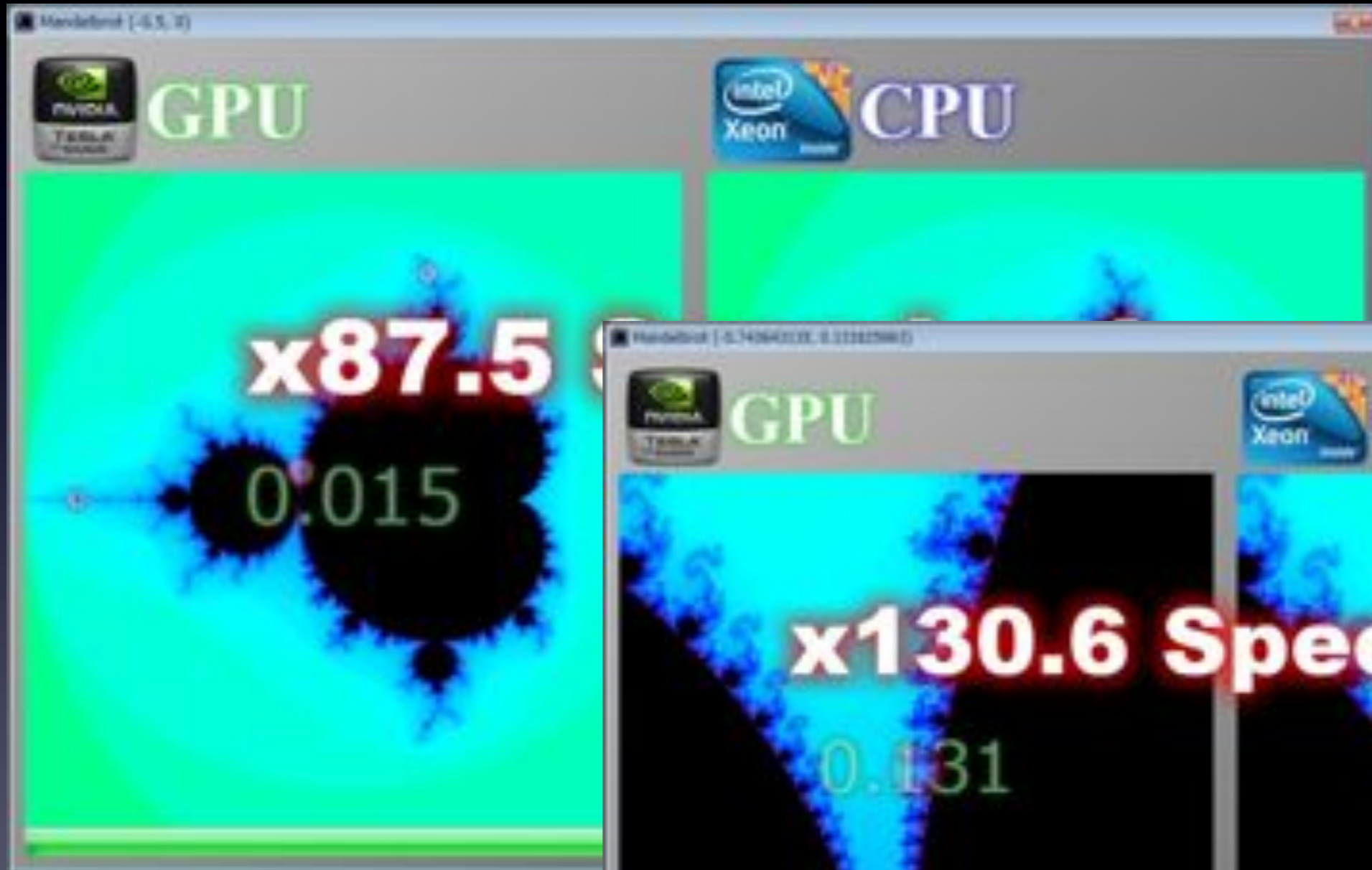
• Intel Xeon X5670/2.93GHz 6-core x2

• 24GB DDR-3 SDRAM/1333MHz, ECC

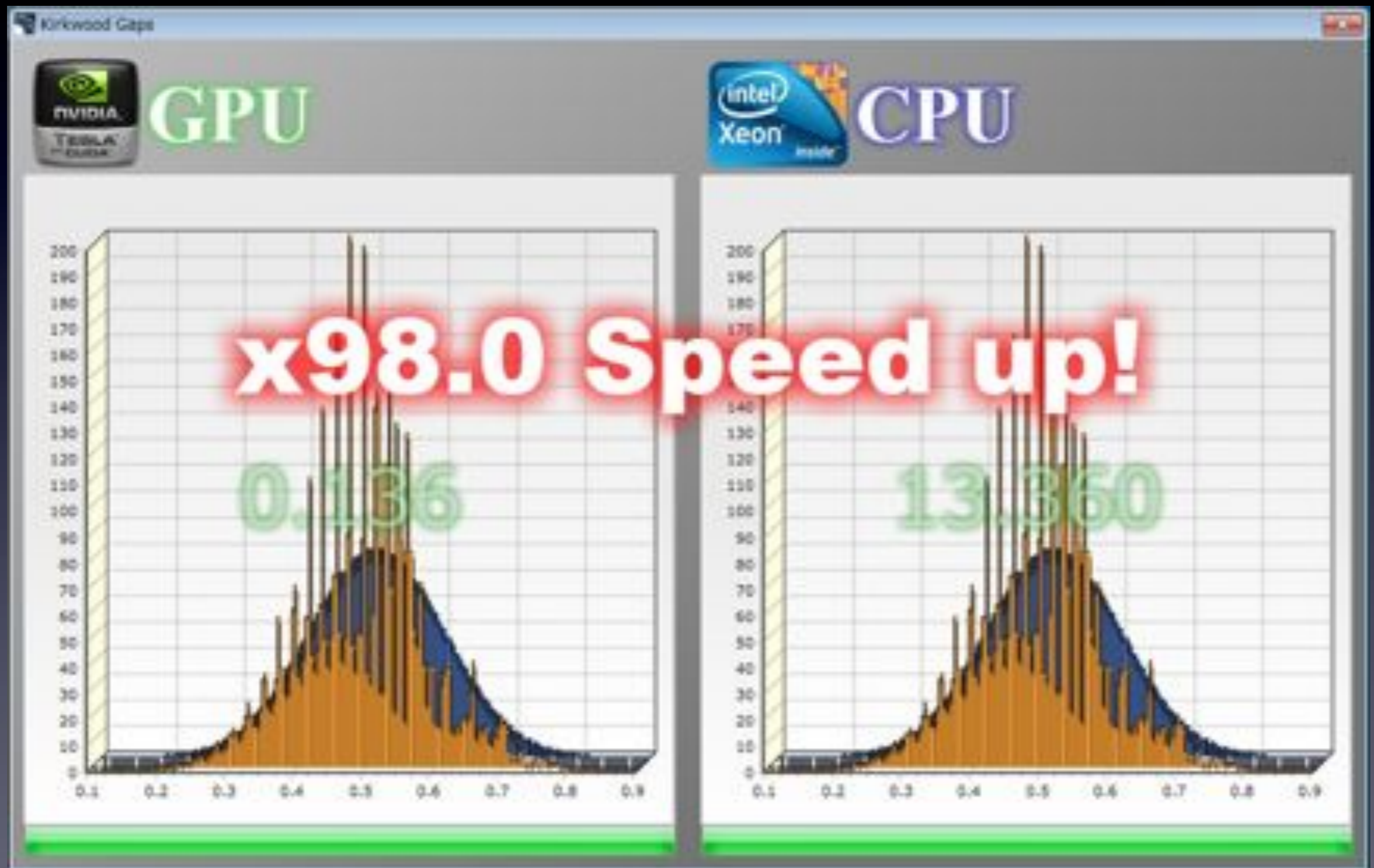
• 450GB 15Krpm SAS HDD

We compared single Tesla with single CPU thread code.

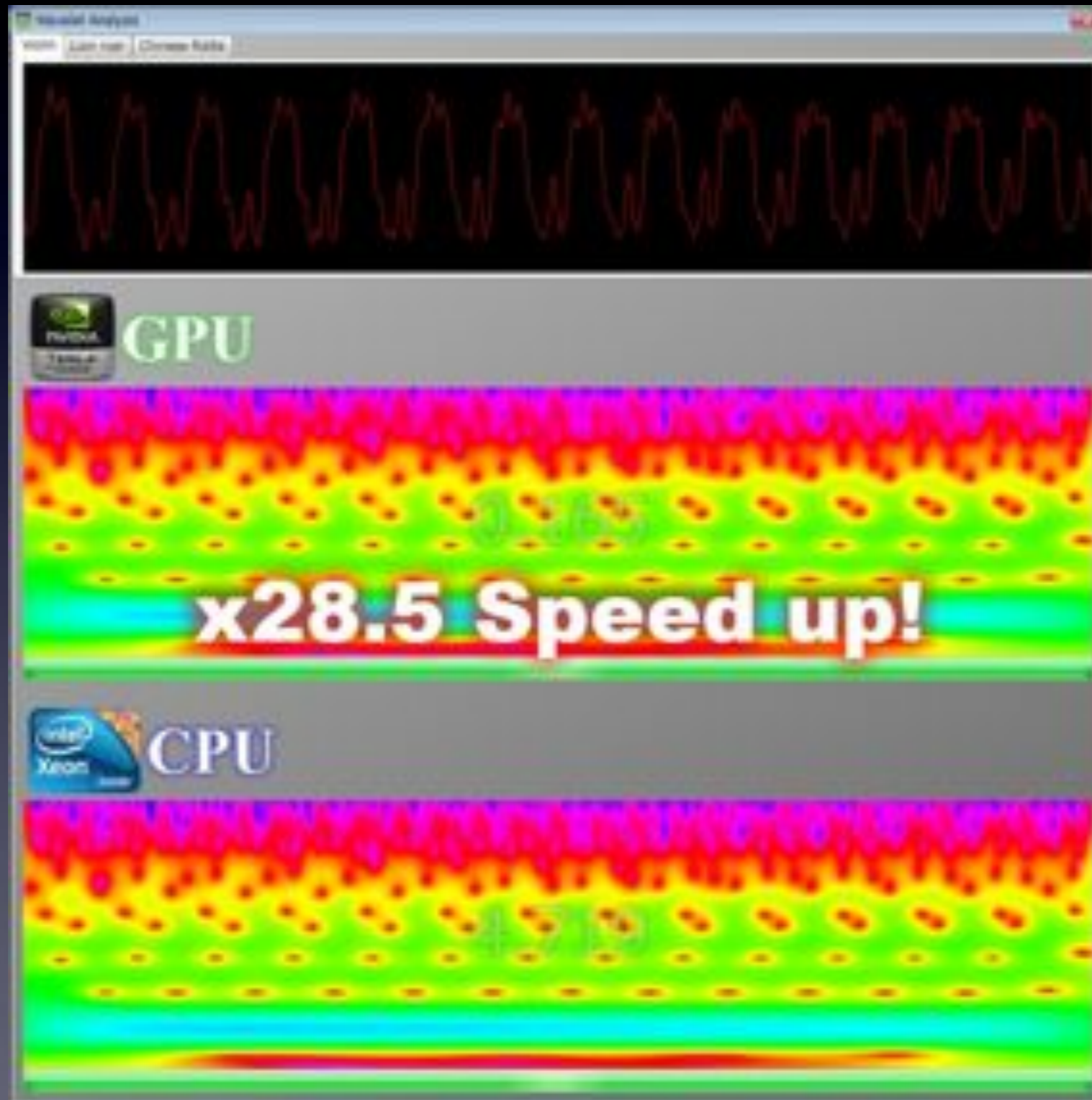
87-130倍速: Mandelbrot



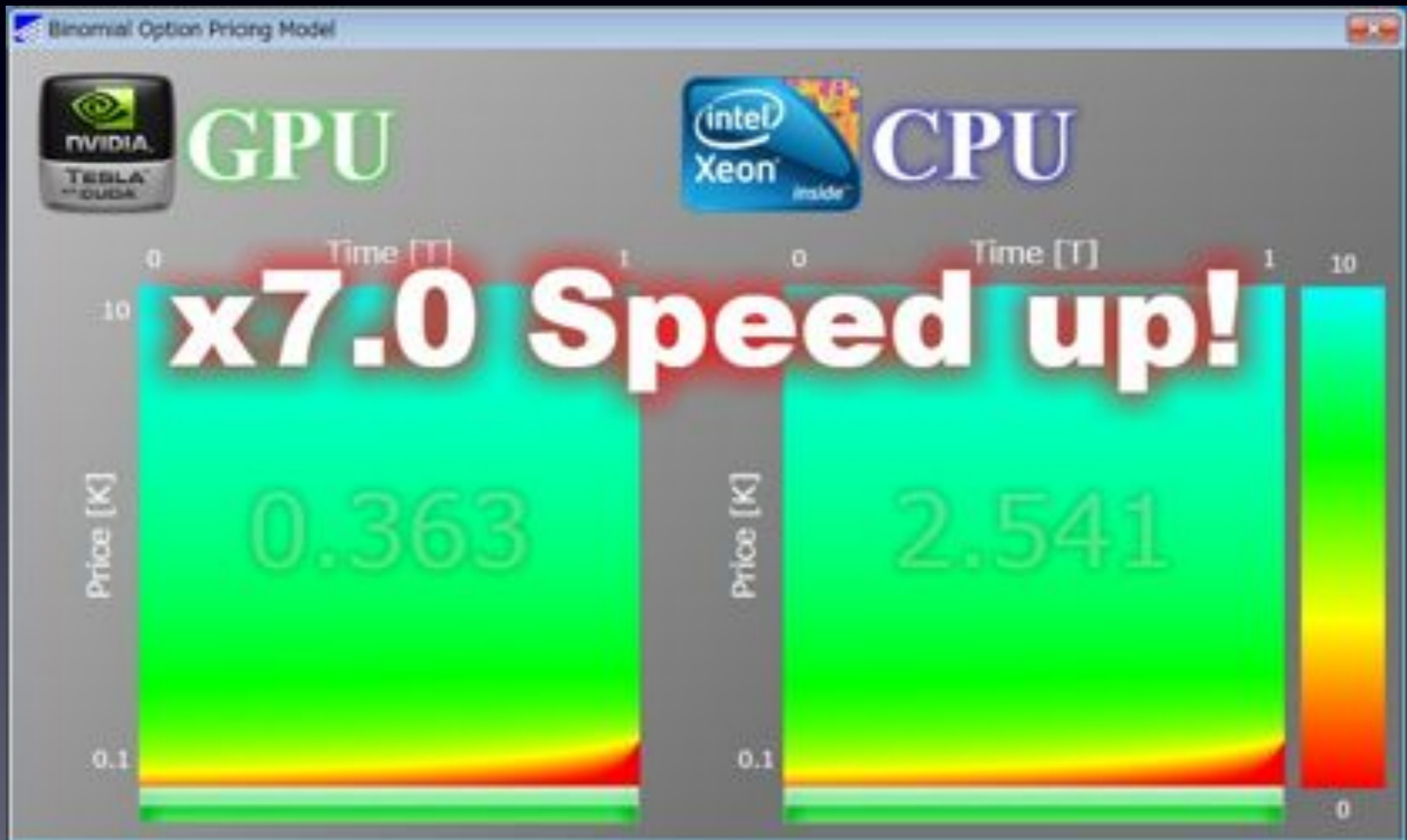
98倍速: Kirkwood Gaps



29倍速: Wavelet Analysis



7倍速: Binomial Tree Option Model



5倍速: Black Scholes Option Model



40倍速: 35年返済ローン





lambda expression方式
でもTeslaは十分に高速

速い理由



正しいスタートアップコード



GPU Computingでよくやりがちな間違いの防止

実行時コンパイルの採用



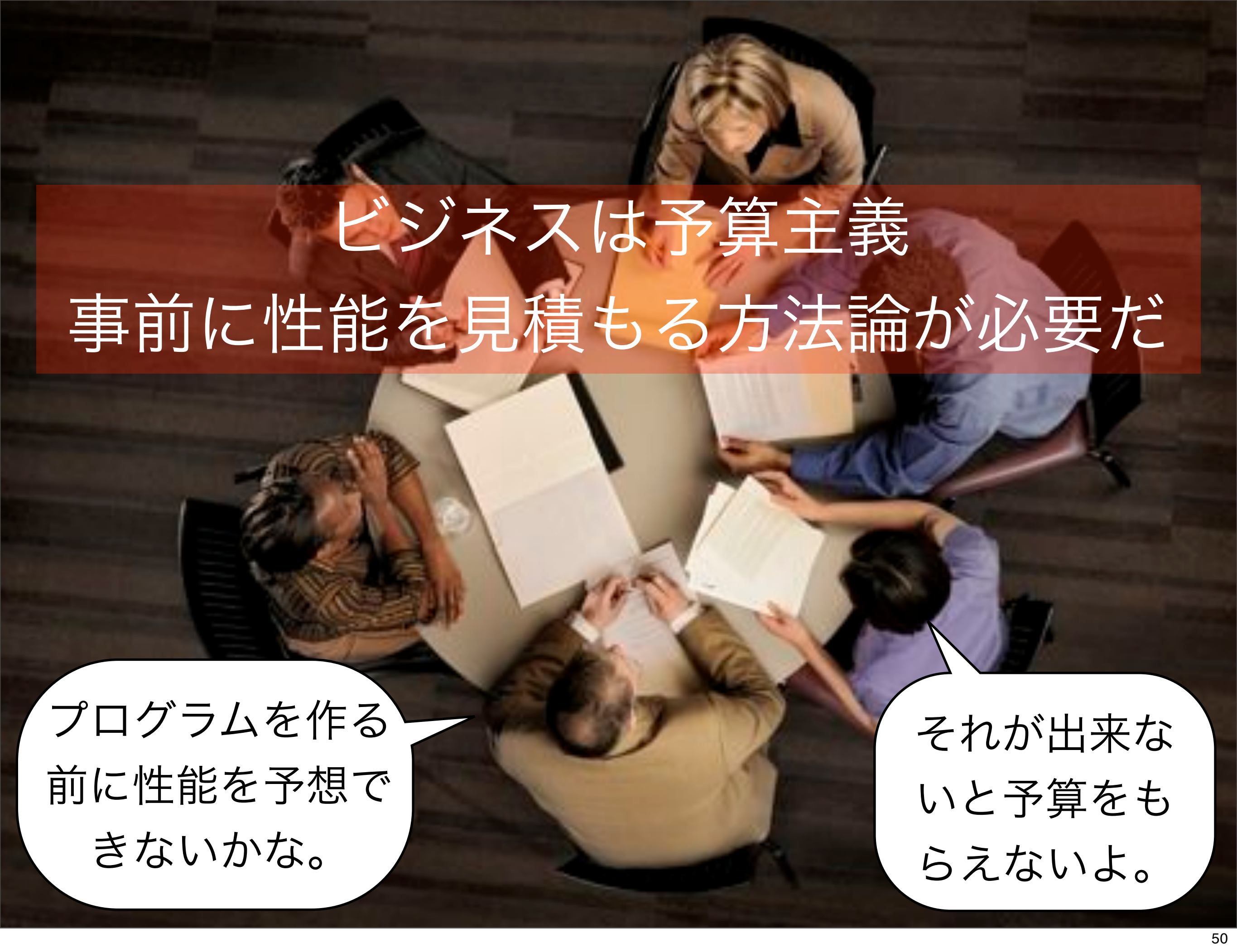
GPU側で動作させるプログラムをホスト言語からは文字列形式でAPIに渡し、nvccの代わりに実行時コンパイルしている。ここで最適化を行った。

デザインパターンの強制



nt_parallel_forの呼び出し方ではshared nothingな並列化しか書けない。逆に言えばそれ以外の用途に
使いたければCUDAで書くしかない。

GPU Computingの普及
を阻む壁はもう一つある



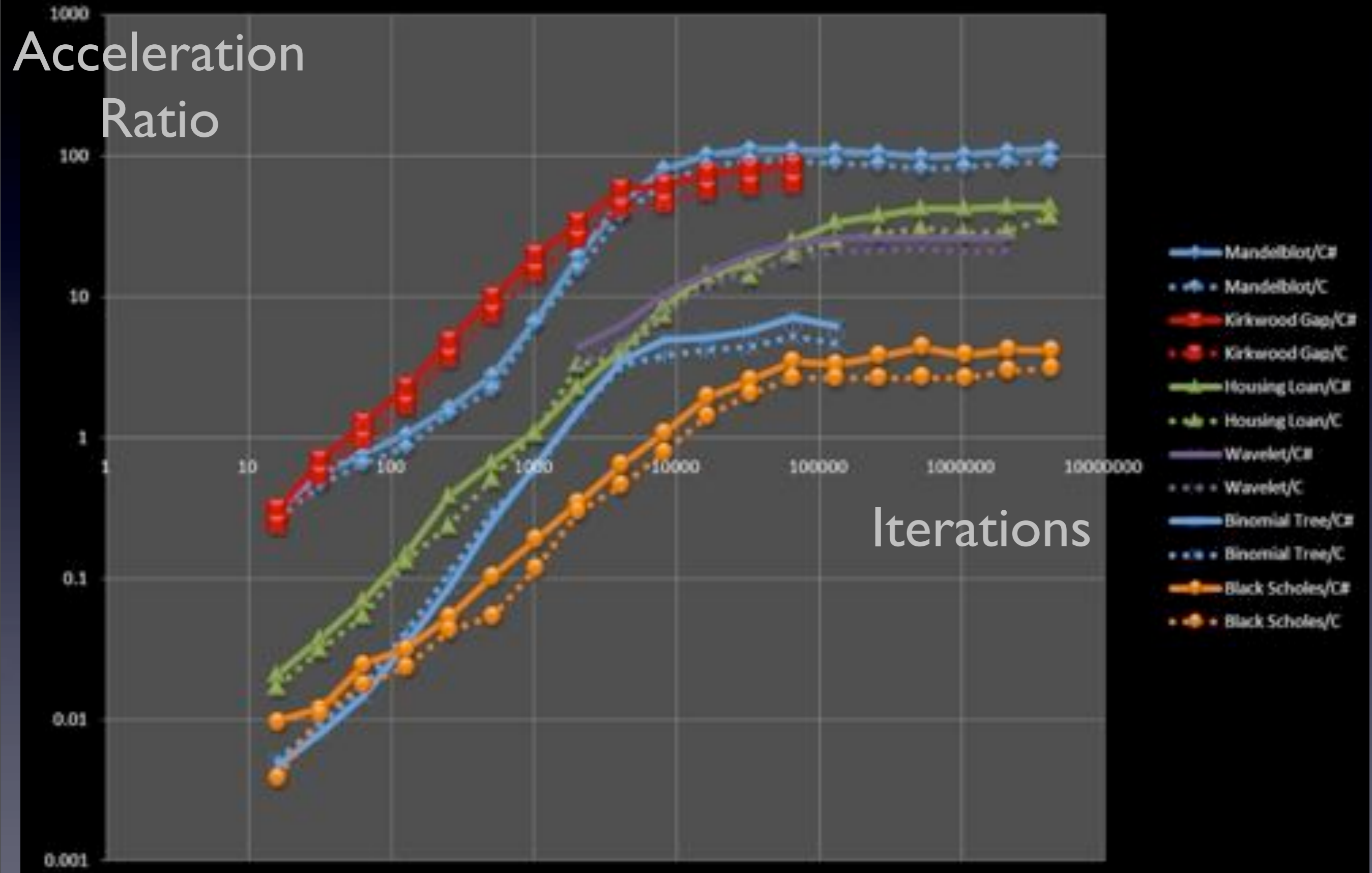
ビジネスは予算主義 事前に性能を見積もる方法論が必要だ

プログラムを作る前に性能を予想できないかな。

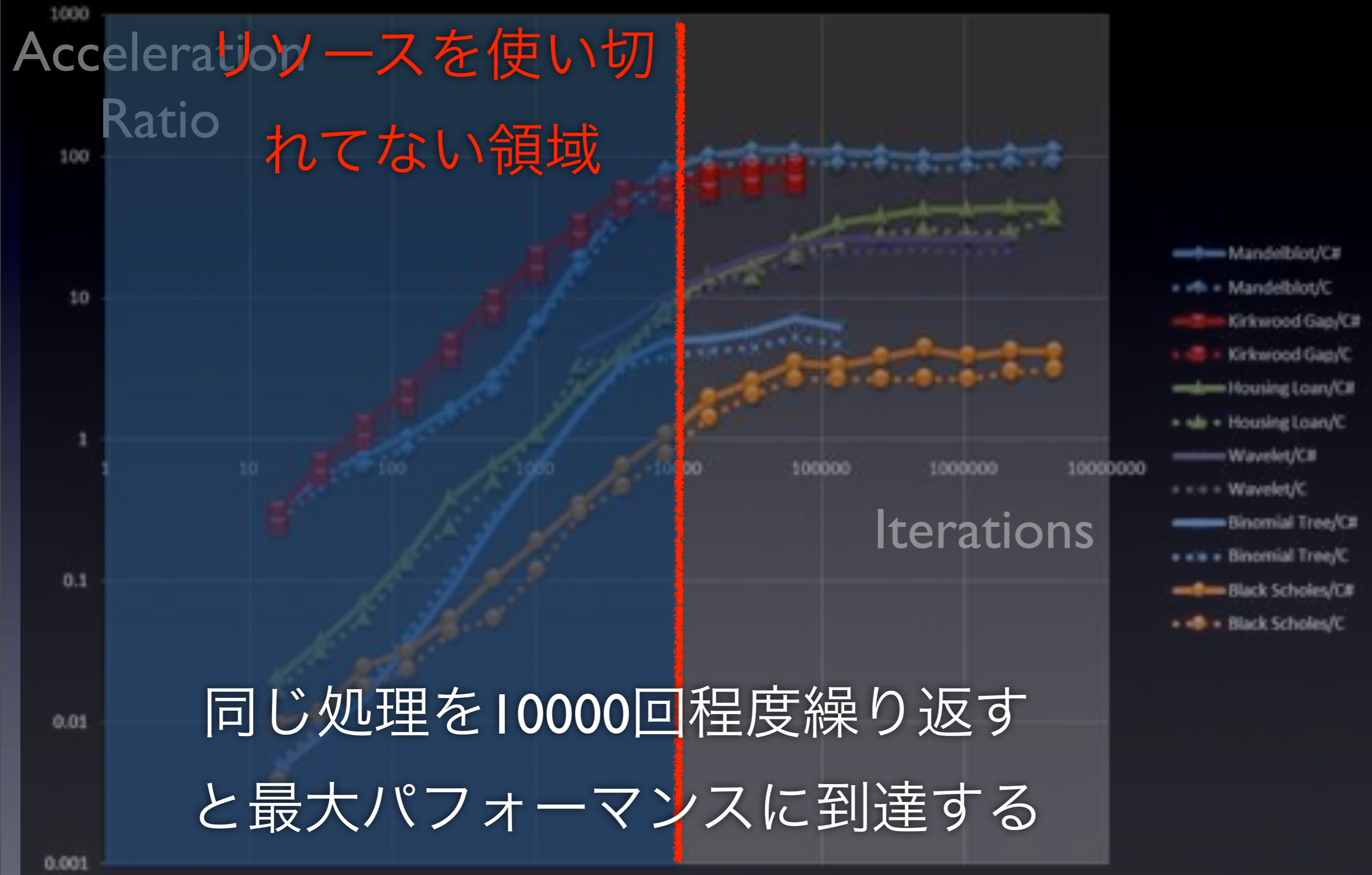
それが出来ないと予算をもらえないよ。

そこで、先程のベンチマーク
テストの結果を分析しよう

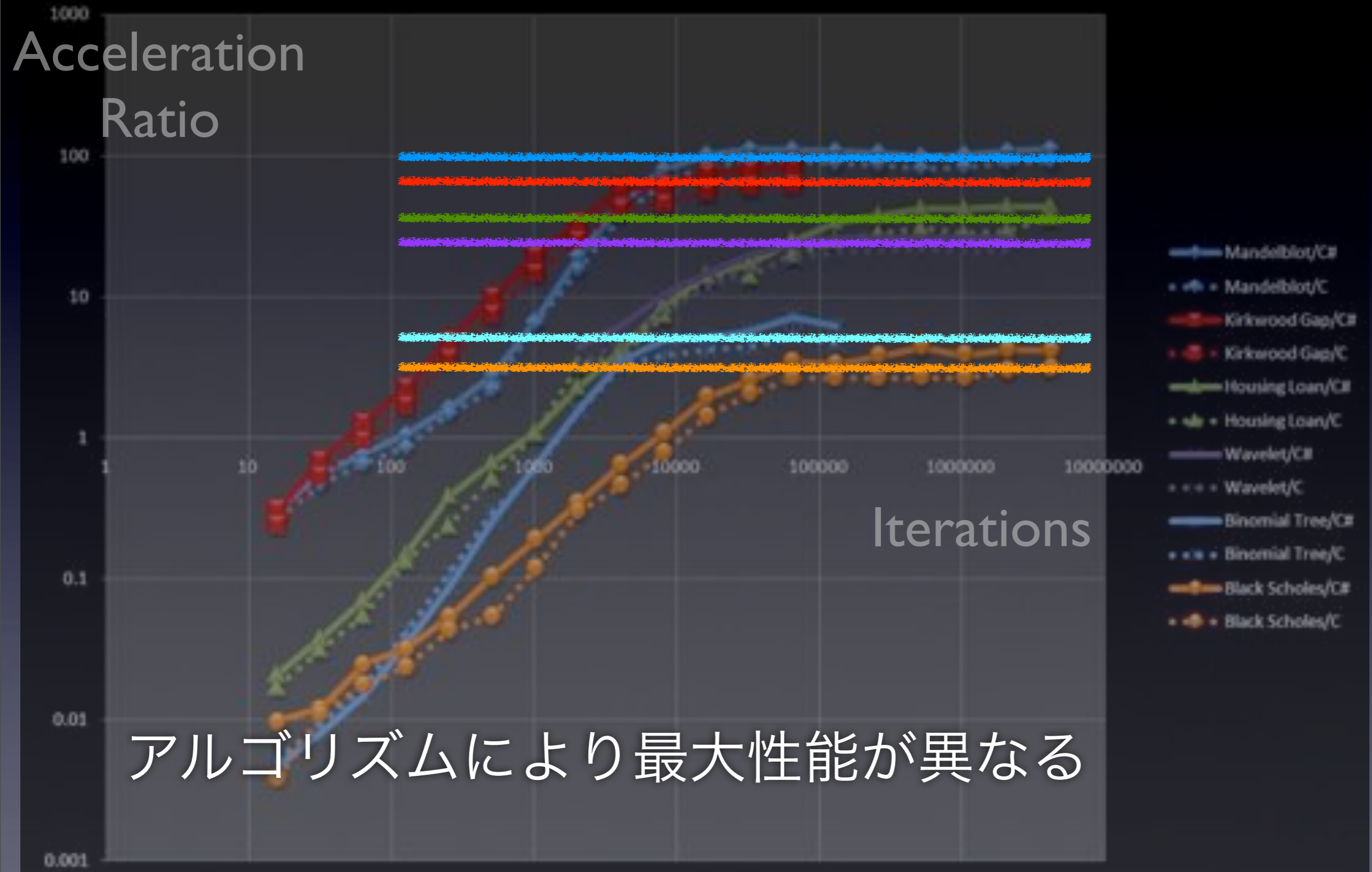
Relative Performance vs. for-loop Iterations



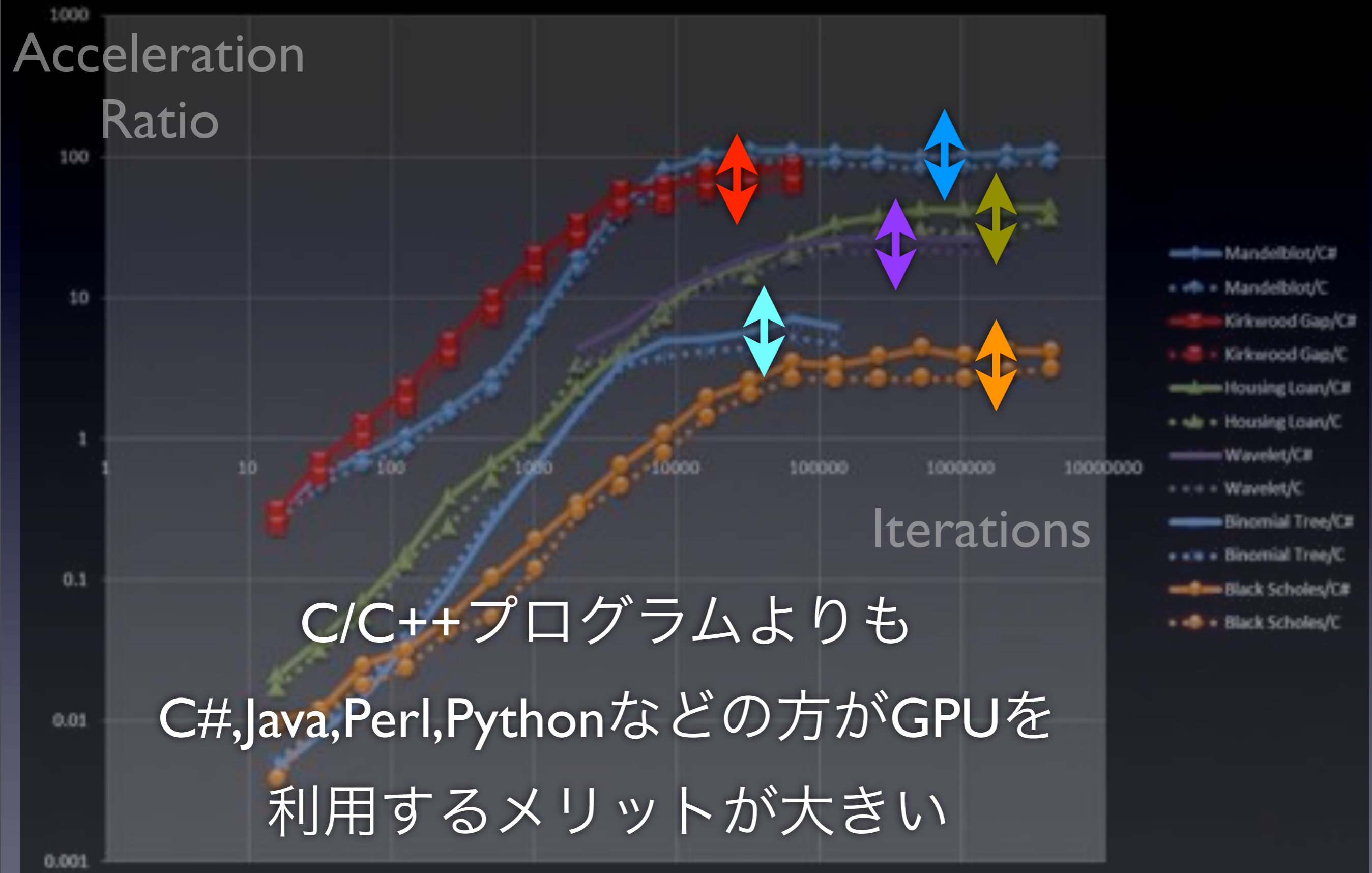
Relative Performance vs. for-loop Iterations



Relative Performance vs. for-loop Iterations



Relative Performance vs. for-loop Iterations



A photograph of two children dressed as superheroes. The child on the left is wearing a blue suit with a white star on the chest and a blue mask. The child on the right is wearing a red and yellow suit with a red star on the chest and a red mask. They are standing on a sandy beach with the ocean in the background. A semi-transparent red box is overlaid on the image, containing white text.

GPU Computing の 効率を握るカギは2つ

for-loop Iterations



同じ処理の繰り返しが多いほど性能が上がる。一般にTesla GPU 1枚あたり10000回程度の計算を行えば性能が最大化する。



Complexity

Ops/Bytes比が高いほど
性能が上がる。これは一
般にプログラムが扱う
データ量に対してコード
が長い状態を意味する。

この2つの指標を使って
チャートを作る

Complexity



100

1000

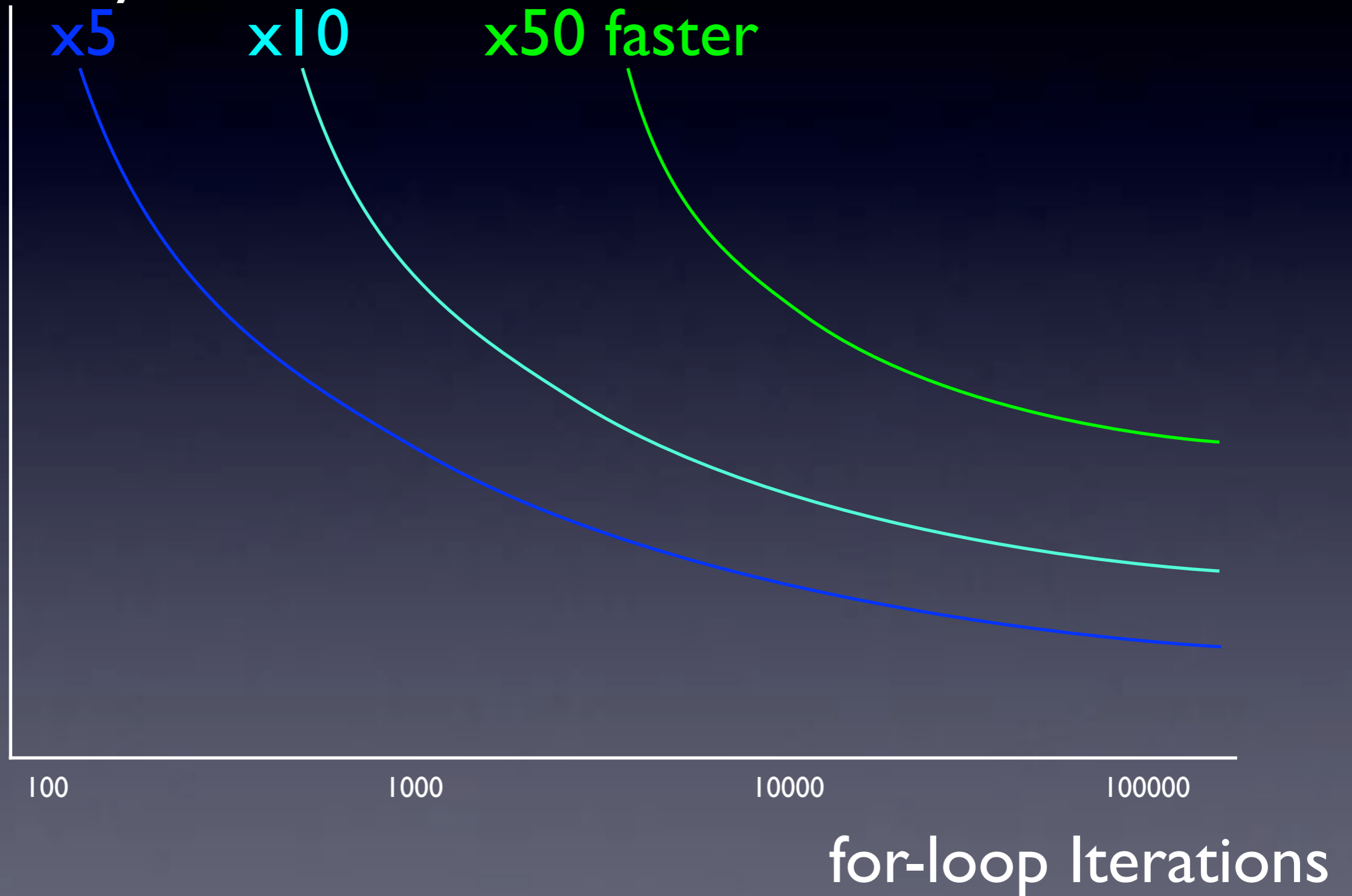
10000

100000

for-loop Iterations

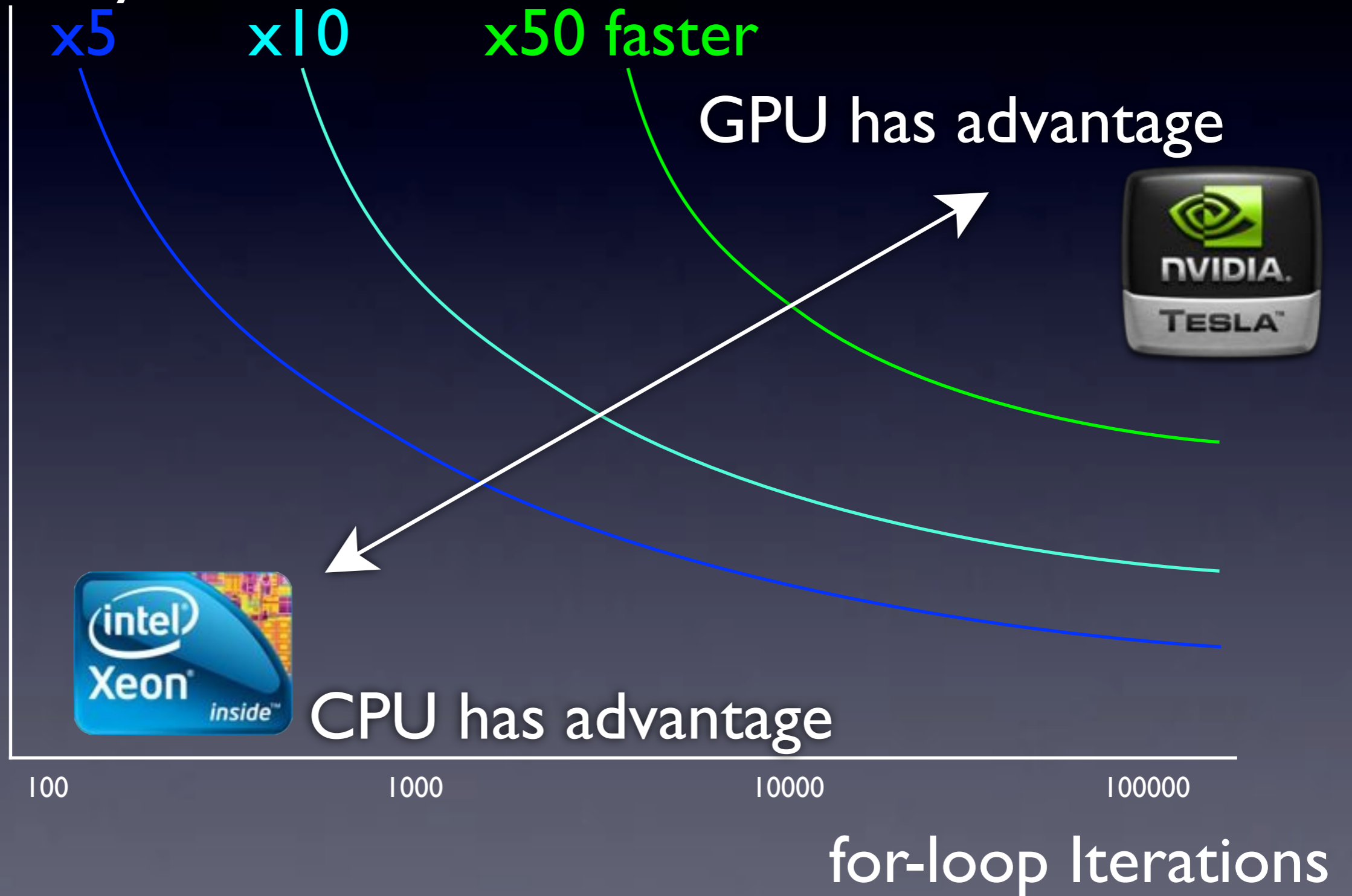
すでに見た通り、ComplexityとIterationsが大きくなるほどGPUのパフォーマンスが上がる

Complexity



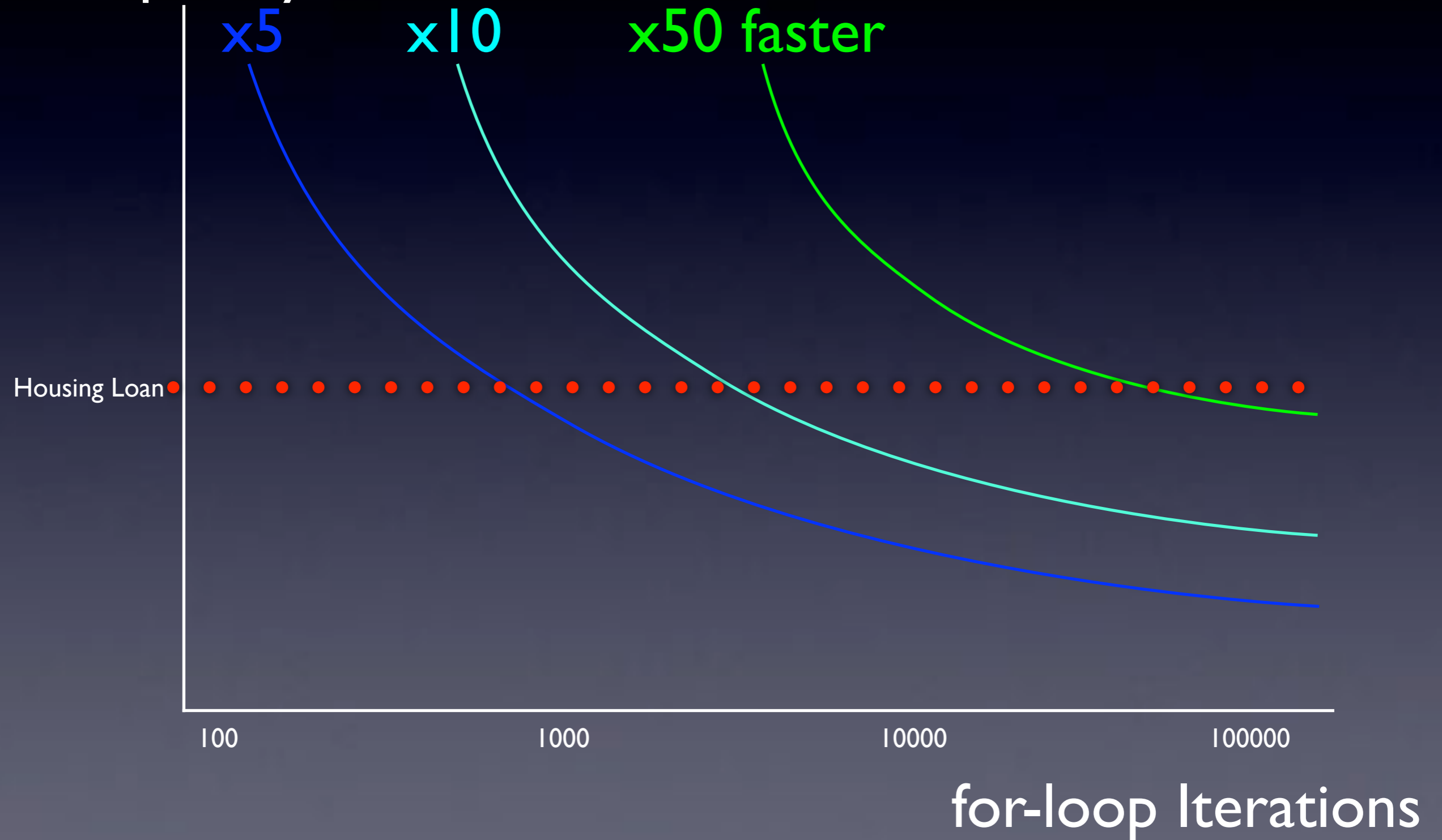
2ソケット8コアサーバーが安価に手に入る現在では、
10倍速を超えなければGPUを使う意味は乏しい

Complexity



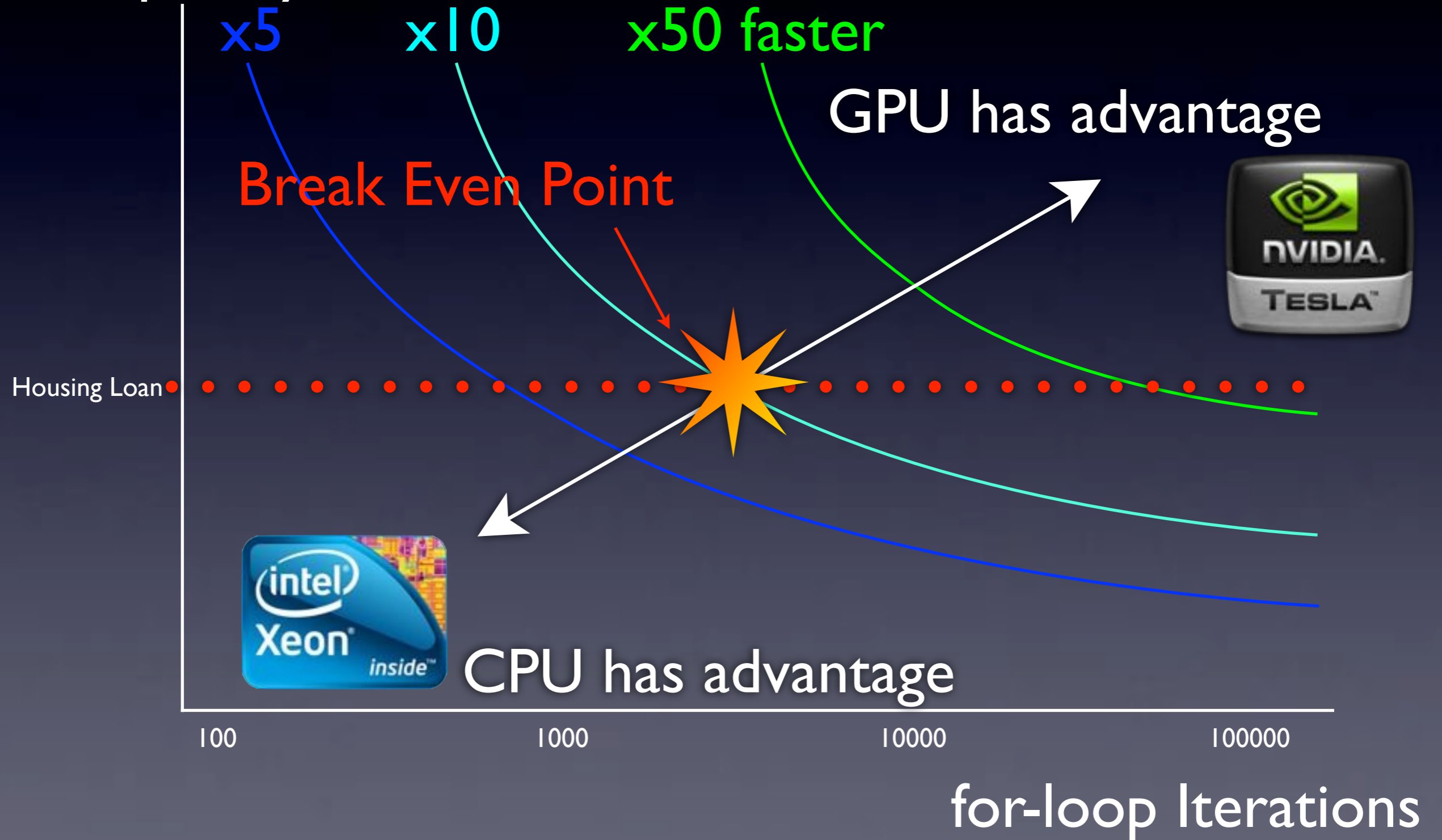
このチャートにアルゴリズム の性能線を引く

Complexity



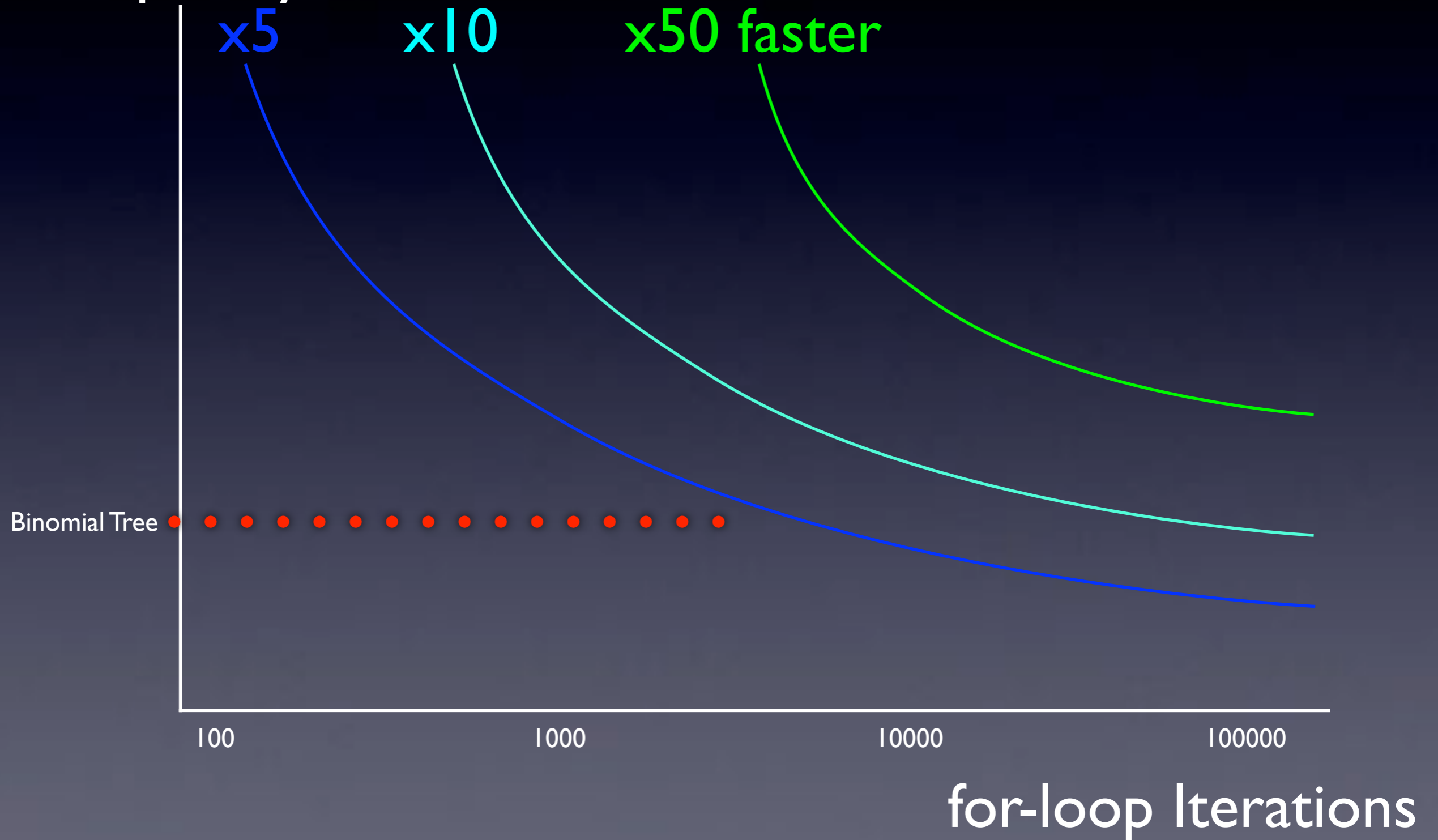
x10線との交点がこのアルゴリズムの採算点だ

Complexity



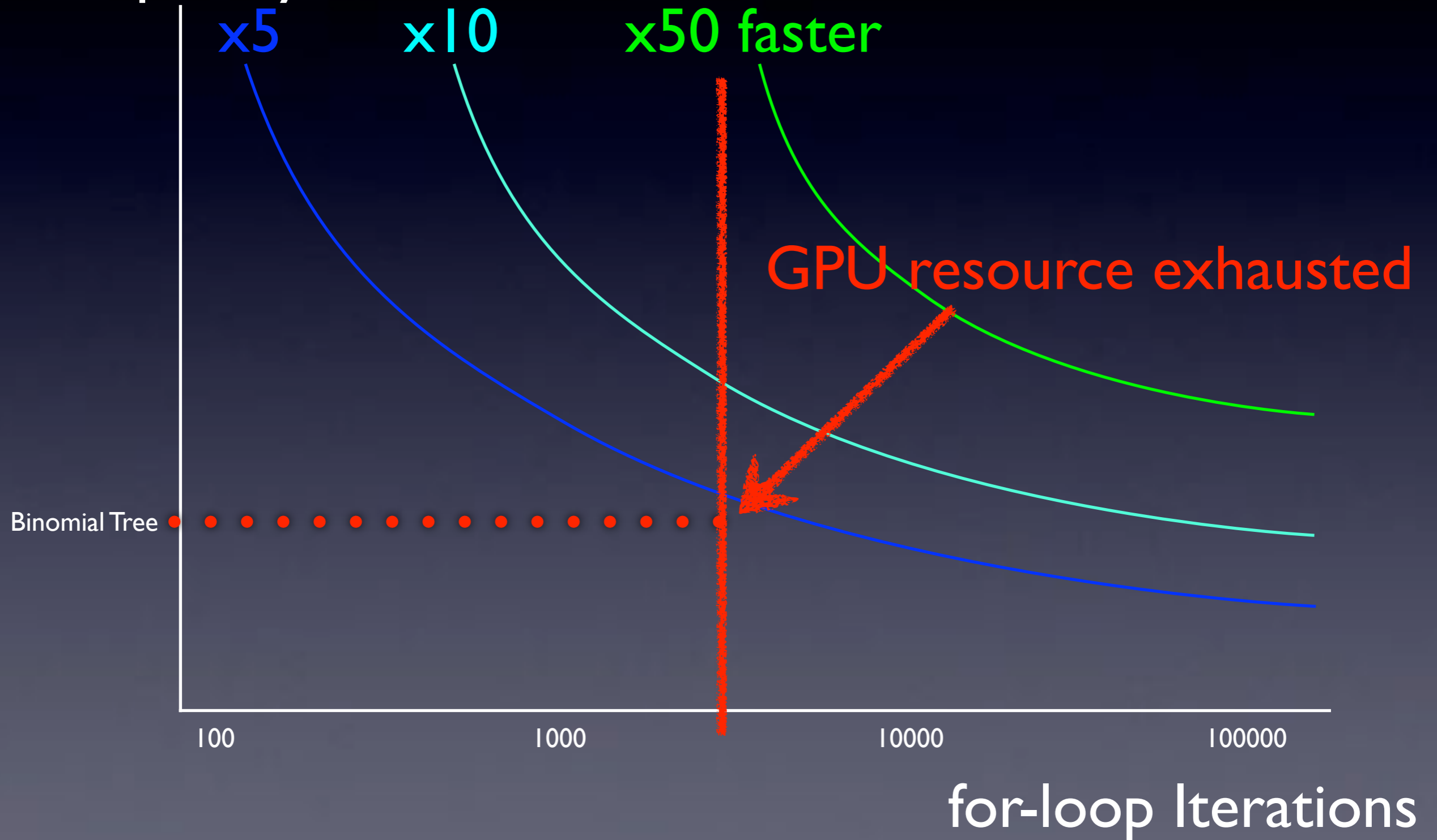
性能が伸びないアルゴリズムもある

Complexity



GPUのリソースを使い尽くしたのが原因だ

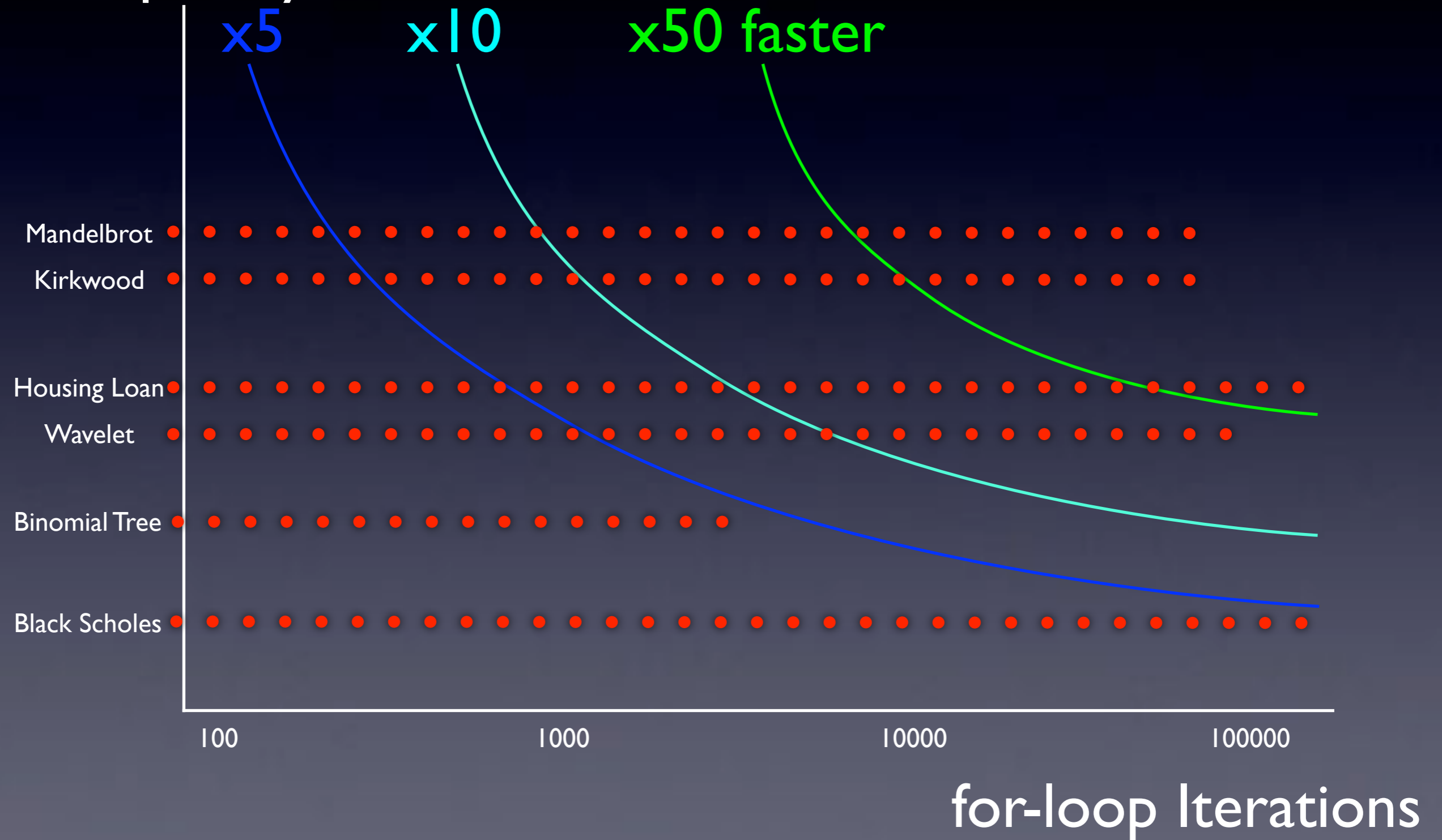
Complexity



さらにもういろいろな

アルゴリズムを並べてみよう

Complexity



意外なことに、一見複雑に思われがちな金融デリバティブの計算よりも、単純なローン計算の方がGPUのパフォーマンスを出しやすい



x5: Black Scholes

x40: Housing Loan



単なる繰り返し計算の方がComplexityが高いからだ

```
...
string code = "
  [] (int i, double annualizedRate, int term, double monthlyPayment) => double {
    const int t = term;
    const double m = monthlyPayment;
    const double monthlyDf = 1.0 / (1.0 + annualizedRate / 12.0);
    int elapsedMonth;
    double val = 0.0;
    double df = 1.0;
    for (elapsedMonth = 0; elapsedMonth < t; elapsedMonth = elapsedMonth + 1) {
      df = df * monthlyDf;
      val = val + m * df;
    }
    return val;
  }
";
...
```

x40: Housing Loan

```
...
string code = "
  [](int i, double R, double Sigma, double S, double K, double T) => double {
    double rt = R * T;
    double sigmaSqrtT = Sigma * sqrt(T);
    double d = (log(S / K) + rt) / sigmaSqrtT + sigmaSqrtT * 0.5;
    return S * NormDist(d) - K * NormDist(d - sigmaSqrtT) * exp(- rt);
  }
";
...
```

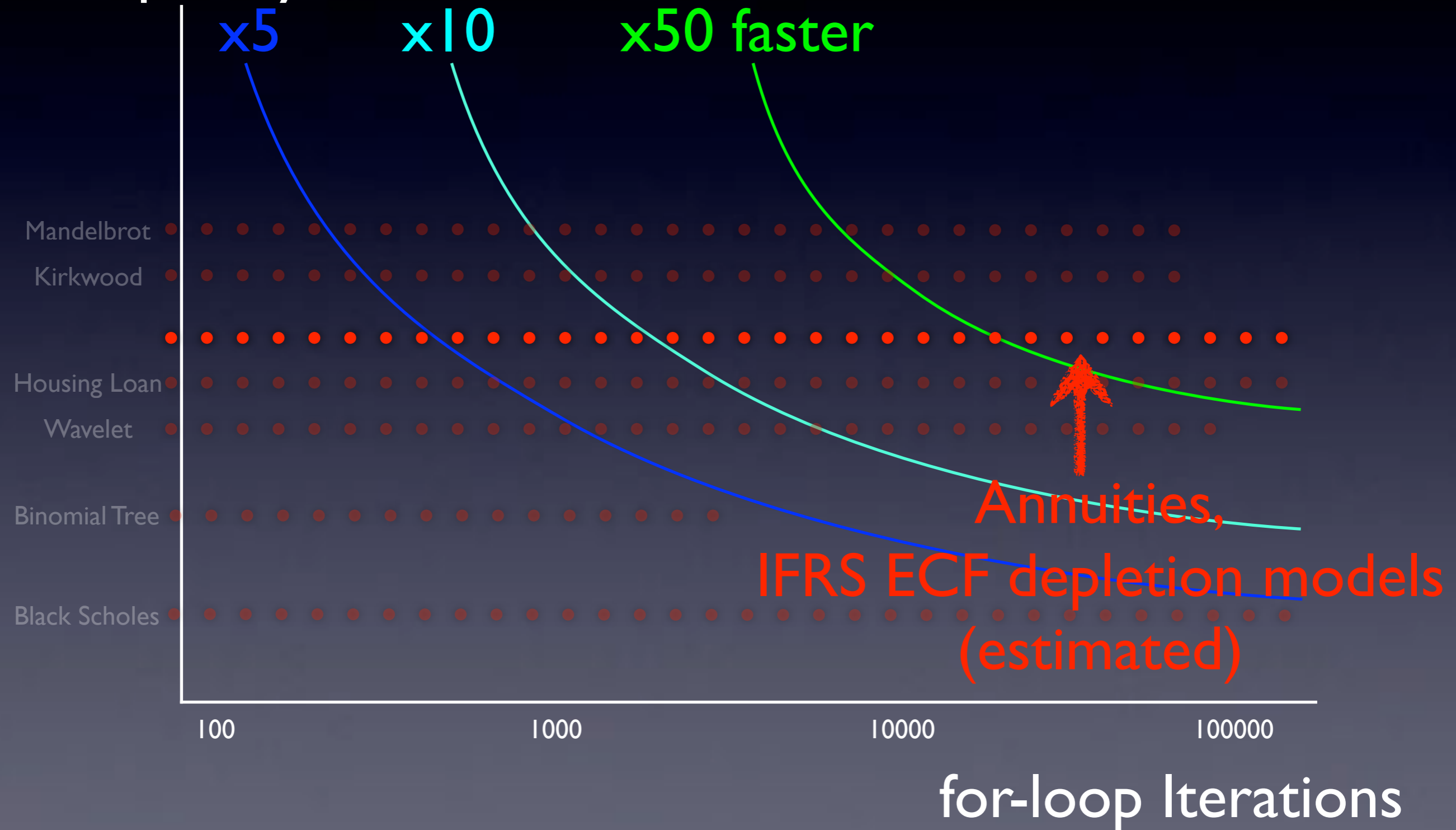
x5: Black Scholes

Q. あなたがGPUにやらせようとして
いることは、どれくらいのComplexity
とIterationsを持っているのか？

A. 類似しているアルゴリズムと比較しよう。例えばあなたが年金計算やIFRS ECF減損モデルに対応したローン評価をやりたいとする。ならば Complexityは先ほどのHousing Loanよりも大幅に高い。

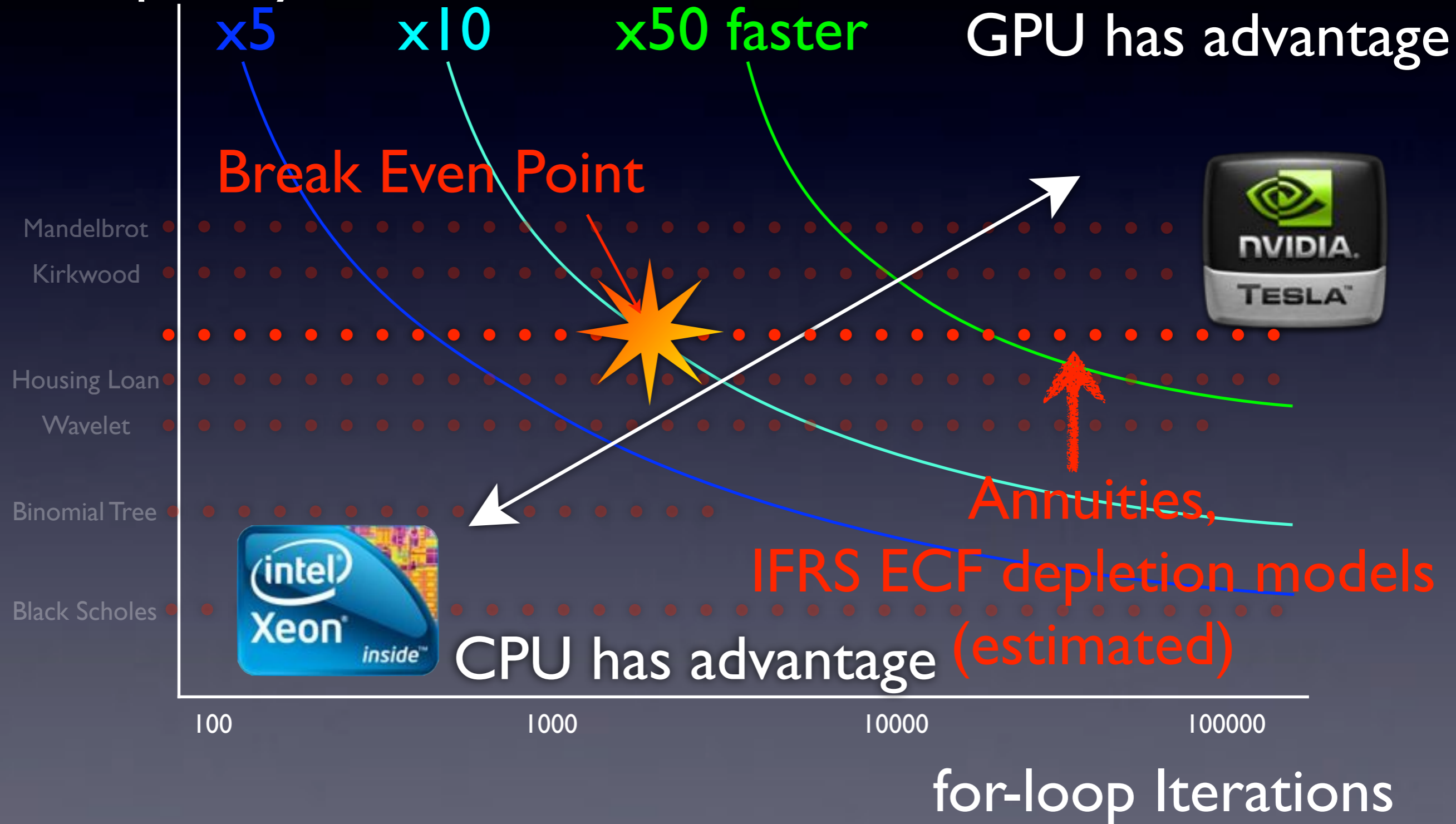
大体このあたりと予想がつく

Complexity



必要な処理件数がIterationsとx10線の交点を上回っていけばGPUへの投資価値がある

Complexity



このRule of Thumbを適用し
ようにも比較対象となるア
ルゴリズムがないならば...

nt_parallel_forを使って 実際に計測してみればよい

The screenshot shows a Microsoft Excel spreadsheet with a VBA function named `NPV` implemented using `nt_parallel_for`. The function calculates the Net Present Value (NPV) for a series of cash flows. The spreadsheet compares the results of the Excel PV function with the GPU function using `nt_parallel_for` for various interest rates and terms.

```
Function NPV(rate As Double, pay As Variant) As Double
    Dim i As Integer
    Dim term As Integer
    Dim cf As Double
    Dim df As Double
    Dim r As Double
    Dim j As Integer
    Dim sum As Double

    term = Int([rate] * 12)
    cf = -pay
    df = 1
    r = 1 / (1 + rate / 12)

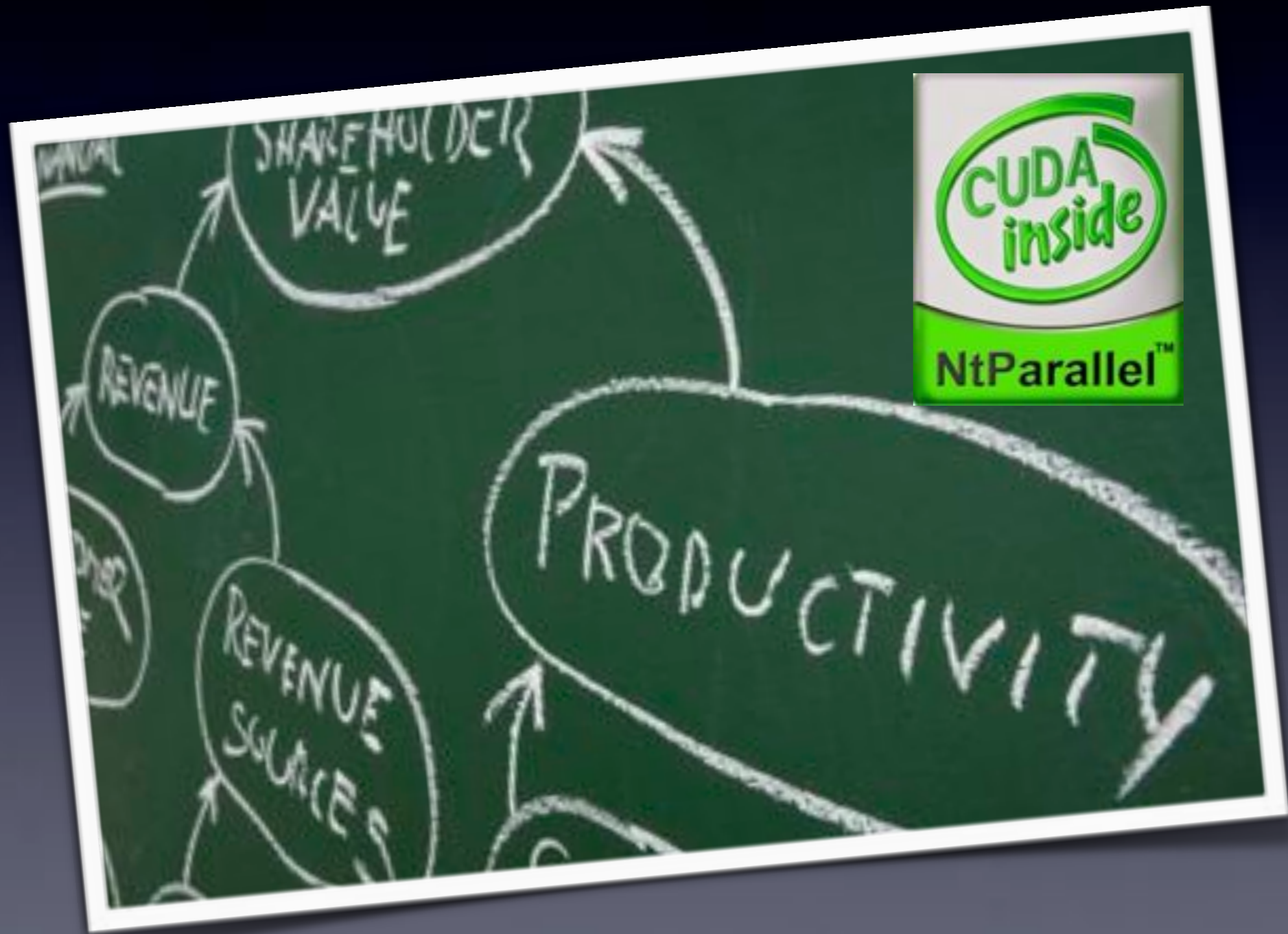
    For j = 0 To term - 1
        df = df * r
        sum = sum + cf * df
    Next j

    NPV = sum
End Function
```

Rate	monthlyPayment	term	Excel PV function	GPU (using nt_parallel_for)
0.0001	100000	35	¥41,926,411	¥41,926,411
0.0002	100000	35	¥41,852,995	¥41,852,995
0.0003	100000	35	¥41,779,750	¥41,779,750
0.0004	100000	35	¥41,706,677	¥41,706,677
0.0005	100000	35	¥41,633,775	¥41,633,775
0.0006	100000	35	¥41,561,043	¥41,561,043
0.0007	100000	35	¥41,488,481	¥41,488,481
0.0008	100000	35	¥41,416,089	¥41,416,089
0.0009	100000	35	¥41,343,865	¥41,343,865
0.001	100000	35	¥41,271,811	¥41,271,811
0.0011	100000	35	¥41,199,924	¥41,199,924




だから生産性が重要である



こうしてGPUの専門家が
いらなくなかった

COMPUTER
GENIUS

A woman with long blonde hair is sitting on a bed, wearing a red and white plaid shirt. She is looking towards a man who is lying in bed. The man is wearing a white tank top and glasses, and is holding an open book. A speech bubble is positioned above the man, containing Japanese text. The background shows a window with a view of a building and a red and white striped awning.

GPU コンピューティング使ってみようか

話が噛み合った

この技術を我々の金融アプリケーションにも使っていきたい



nt_parallel_for手法も一般化していきたい。我々とともに働きませんか？



We want you!

<http://www.numtech.com/>