

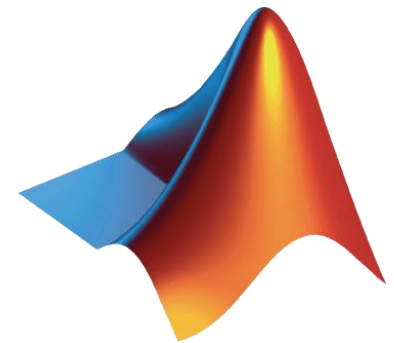
MATLABを高速化しよう！

Speeding Up MATLAB

テジャ ムピララ
MathWorks Japan

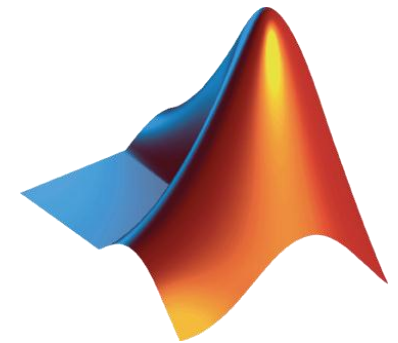
Speeding Up MATLAB

- Q: MATLABは遅いですか？
- A: コードの書き方によります。



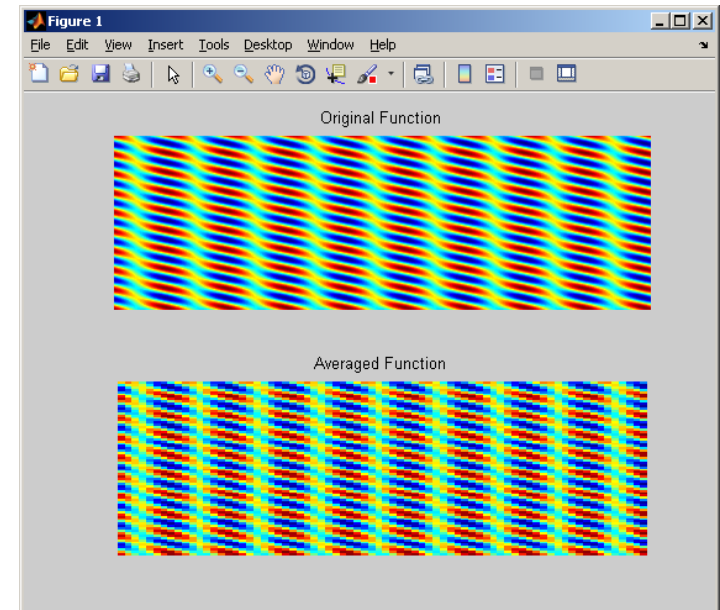
実施内容

- パフォーマンスを向上させる技法
 - 事前割り当て
 - ベクトル化
 - 特別関数
- 3つの例題を通して、コードを分析しながら、説明する
 - 例題 1: 画像のブロックプロセッシング
 - 例題 2: ランダムウォーク
 - 例題 3: 実験データベースのルックアップ
- Parallel Computing Toolbox™
並列処理によるさらなるSpeed-Up



例題1: 画像のブロックプロセッシング

- 関数を2000x2000グリッドで評価
- 25x25 ピクセルの平均をとる
- 結果を比較
- コードのパフォーマンスを比較



例題1: 画像のブロックプロセッシング (reshapeとsum)



2000x2000




25 x 160000



1 x 160000



25 x 6400



1 x 6400



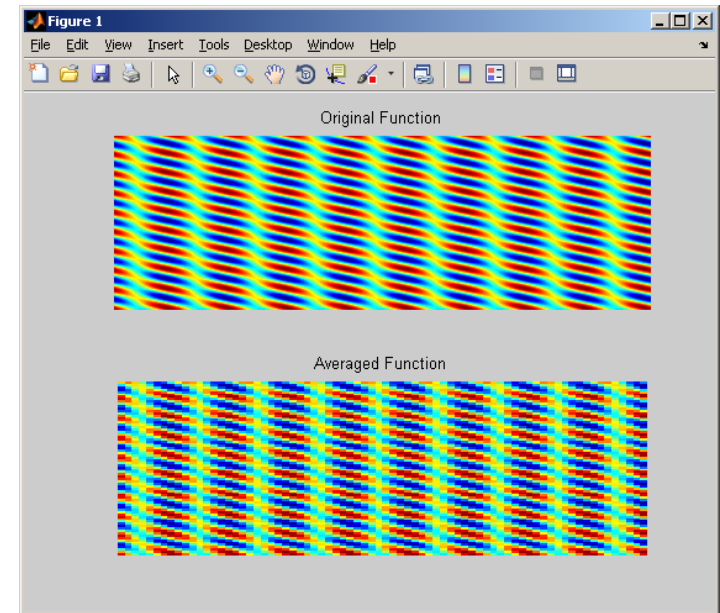
80x 80

例題1: 画像のブロックプロセッシング — まとめ

- 時間を計るために

```
>> tic
```

```
>> toc
```
- コードアナライザを利用
- 事前割り当て
- コードのベクトル化



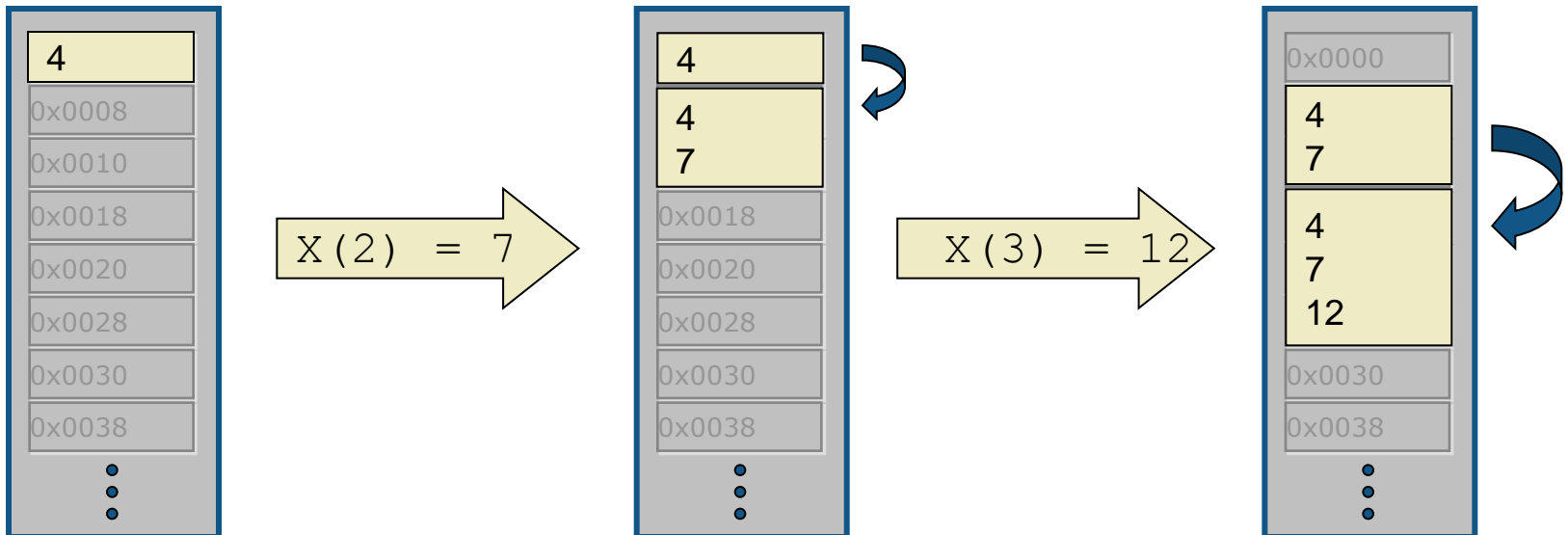
メモリの事前割り当てをしなければ...

```
>> x = 4
```

```
>> x(2) = 7
```

```
>> x(3) = 12
```

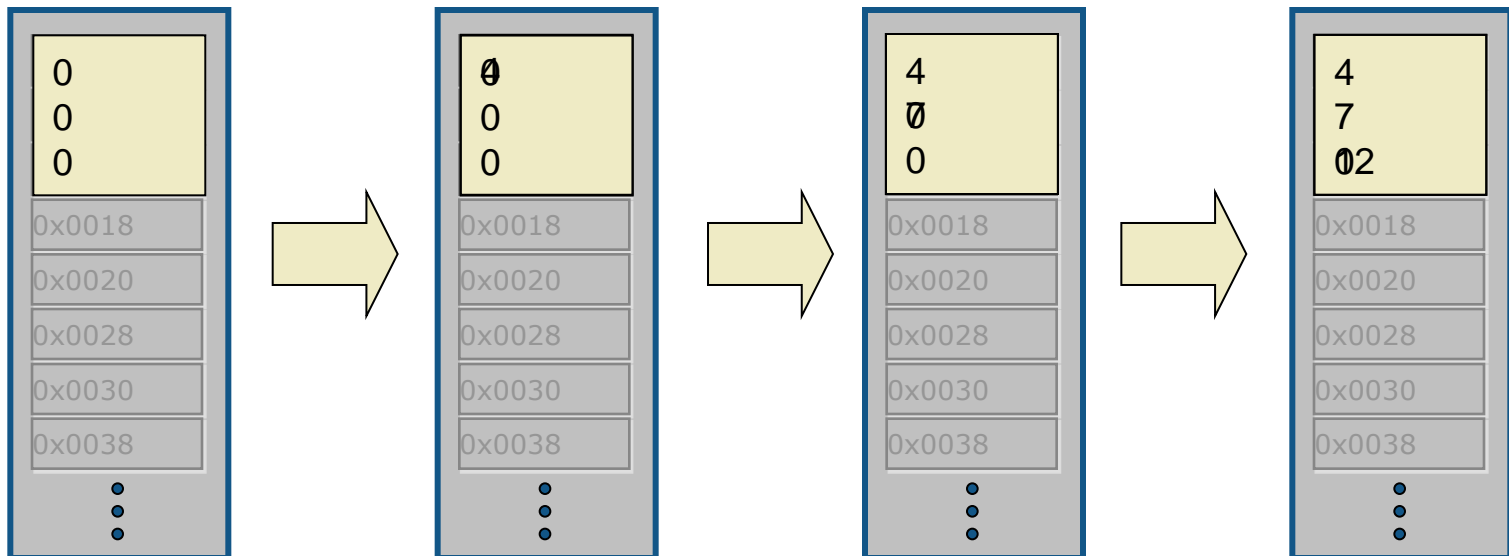
配列のリサイズは
時間がかかる



事前割り当ての効果

```
>> x = zeros(3,1)
>> x(1) = 4
>> x(2) = 7
>> x(3) = 12
```

無駄なデータ
コピーは不要

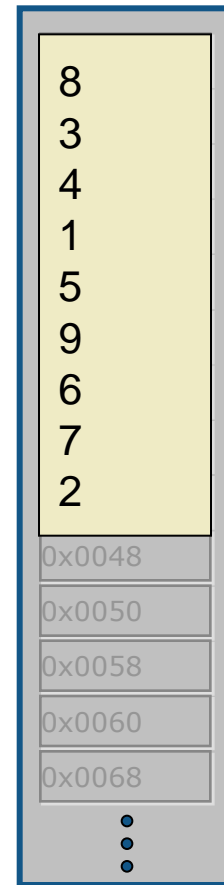


MATLAB配列のデータ格納

```
>> x = magic(3)
```

```
x =
```

8	1	6
3	5	7
4	9	2



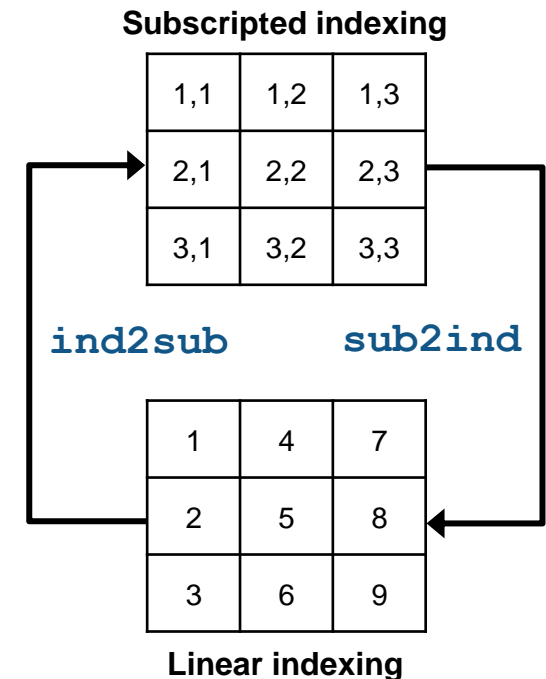
列優先

See the June 2007 article in “The MathWorks News and Notes”:

http://www.mathworks.com/company/newsletters/news_notes/june07/patterns.html

MATLAB 配列にアクセスする、さまざまな方法

- サブスクリプト・インデックス
 - 行番号、列番号を指定
- 線形インデックス
 - 単一の数値を指定
- 論理的インデックス
 - 論理演算を利用して、条件で選択する



MATLAB の計算ライブラリ

- 基本線形代数
 - BLAS: Basic Linear Algebra Subroutines (マルチスレッド)
 - LAPACK: Linear Algebra Package
 - etc.
- JIT/Accelerator
 - ループ(for、while)を高速化させる
 - MATLABコードをその場でCompileする
 - 常に改良されている

BLAS、LAPACK
は隣接するメモリが
必要

JIT の効果を阻害する要因

- ループ内の変数サイズの変更

```
>> x = 1;  
>> x = [1 2; 3 4];
```

- ループ内のデータ型の変更

```
>> x = 1;  
>> x = 'hello';
```

- 等間隔でない ループインデックス または [] を利用したインデックス

```
for n = (1:1000).^2  
    ...  
end
```

JIT の注意点

- IF 文では、評価しやすいものから記述すれば速い

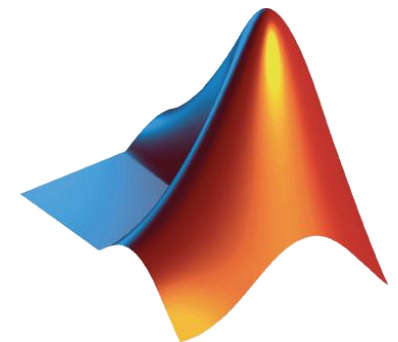
```
if A || B || C
```

```
    ...
```

```
end
```

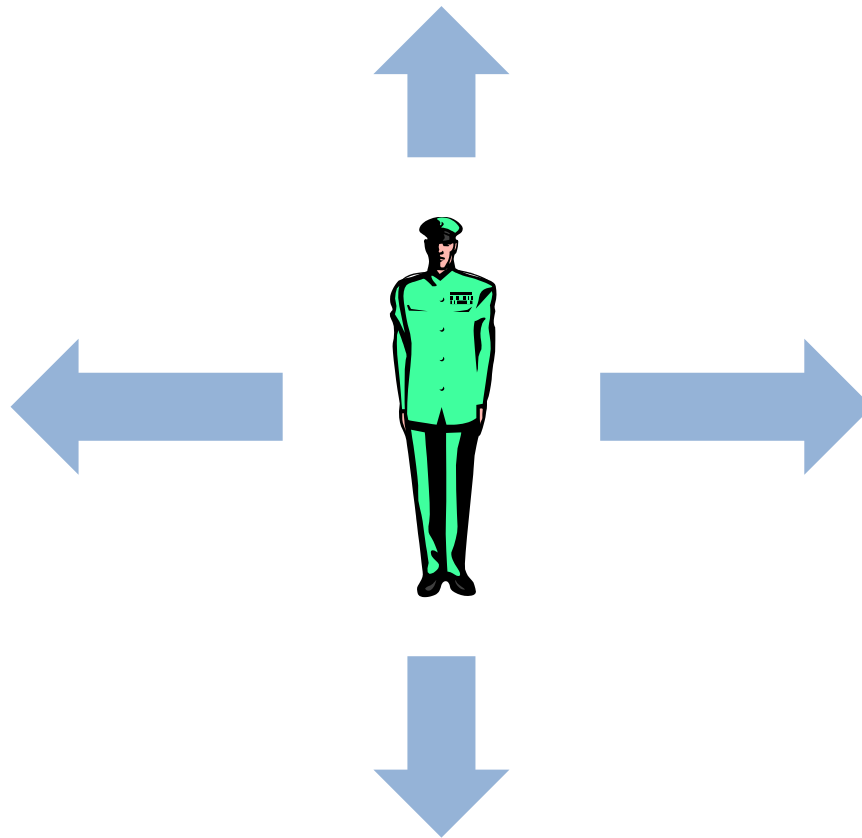
実施内容

- パフォーマンスを向上させる技法
 - 事前割り当て
 - ベクトル化
 - 特別関数
- 3つの例題を通して、コードを分析しながら、説明する
 - 例題 1: 画像のブロックプロセッシング
 - 例題 2: ランダムウォーク
 - 例題 3: 実験データベースのルックアップ
- Parallel Computing Toolbox™
並列処理によるさらなるSpeed-Up



例題2: ランダム ウォーク

- 東西南北、ランダムに方向を決めて、一歩進む



例題2: ランダム ウォーク

- 最初:
- 事前割り当て:
- ベクトル化:
- 組み込み関数を利用して、アルゴリズムの再検討:

例題2: ランダム ウォーク

- 事前割り当て & ベクトル化
- Scriptより、Function（特にループ内では）
- グラフィックスは、hardware rendererが速い
 - `set(gcf,'renderer','opengl');`
- 特別な演算のために特別な関数
 - ベクトルの累積 → `cumsum`
 - 高速化における便利な関数:
`bsxfun`, `reshape`, `accumarray`, `histc`,
`diff`, `repmat`, `permute`, `sparse`

bsxfun

- 「列数が同じ、行数が違う」のをループなしで処理
 - Ex.1 行列Aのそれぞれの列から、それぞれの平均値を引く

$$A = \begin{bmatrix} 2 & 5 & -2 \\ 4 & 1 & 3 \\ 0 & 6 & 2 \end{bmatrix}$$

$$\text{mean}(A) = [2 \quad 4 \quad 1]$$

```
bsxfun(@minus, A, mean(A))
```

```
ans =
```

```

      0      1     -3
      2     -3      2
     -2      2      1
```

bsxfun

- 「列数が同じ、行数が違う」のをループなしで処理
 - Ex.2 行列Aのそれぞれの行が、それぞれのPの値より大きい要素を探す

$$A = \begin{bmatrix} -1 & 0 & 3 \\ 6 & 8 & 7 \\ 11 & 7 & 13 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 \\ 5 \\ 10 \end{bmatrix}$$

```
bsxfun(@gt, A, P)
```

```
ans =
```

0	0	1
1	1	1
1	0	1

sparse

- データが大きいけれど、ほとんどの要素がゼロ
 - 例えば: 熱伝導方程式 の 有限差分 Simulation

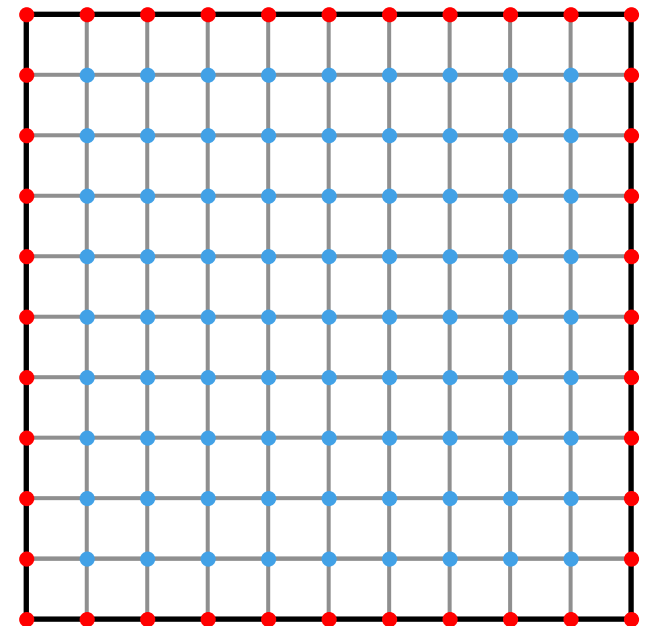
$$\frac{\partial U}{\partial t} = k \nabla^2 U$$

周囲の温度 = 1
内部の温度 = 0

計算に必要な行列サイズ

$$= N^2 \times N^2$$

$$\begin{bmatrix} \dots & & & & \\ & \dots & & & \\ & & \dots & & \\ & & & \dots & \\ & & & & \dots \end{bmatrix}$$

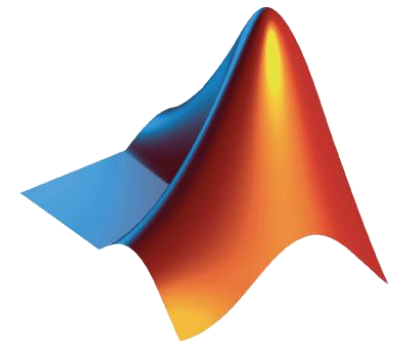


高速化に便利な関数: MATLAB→Help→ユーザーガイド→ プログラミングの基礎→パフォーマンス

```
bsxfun, reshape, accumarray, histc,  
diff, repmat, permute, sparse, meshgrid,  
...
```

実施内容

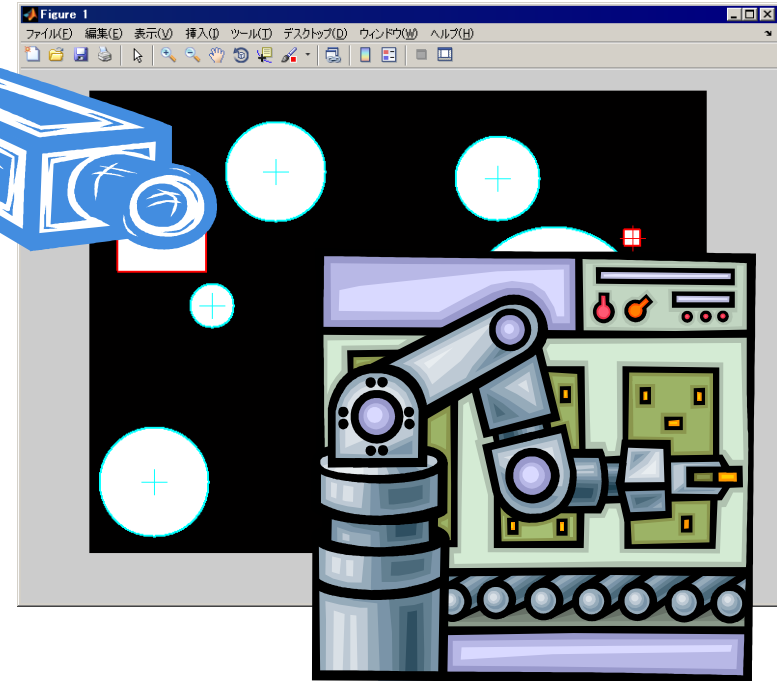
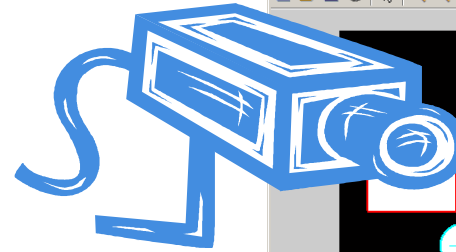
- パフォーマンスを向上させる技法
 - 事前割り当て
 - ベクトル化
 - 特別関数
- 3つの例題を通して、コードを分析しながら、説明する
 - 例題 1: 画像のブロックプロセッシング
 - 例題 2: ランダムウォーク
 - 例題 3: 実験データベースのルックアップ
- Parallel Computing Toolbox™
並列処理によるさらなるSpeed-Up



例題3: 実験データベースの ルックアップ

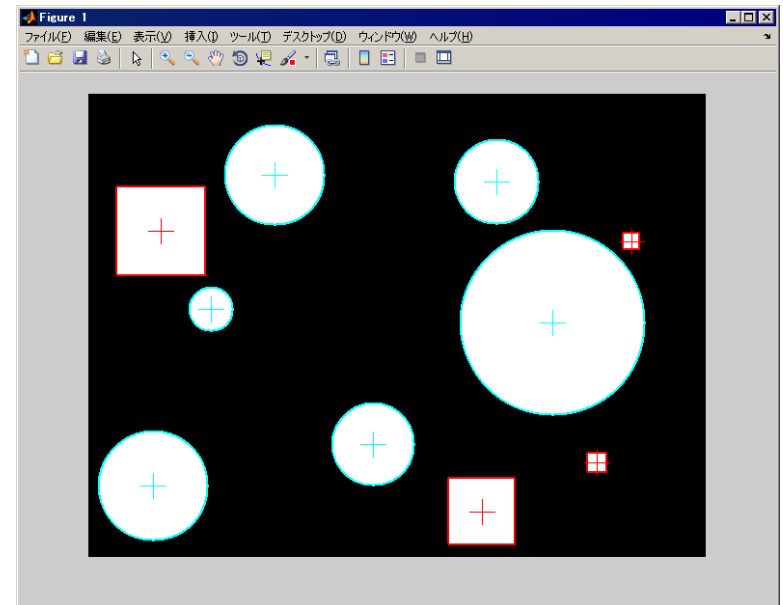
- 複数のテスト画像を読み込む
- 形状の情報を抽出(○と□の位置とサイズ)
- そのデータをデータベースで検索
- 画像のシリアル番号をExcelに書き込む

画像のシリアル番号:
8E3CA649



例題3: 画像のデータベース検索

- Profilerを利用して、コードを分析
- ネックとなる部分を狙って、修正した
- File I/Oを削減
- Figureを再利用



Profiler 結果の解釈

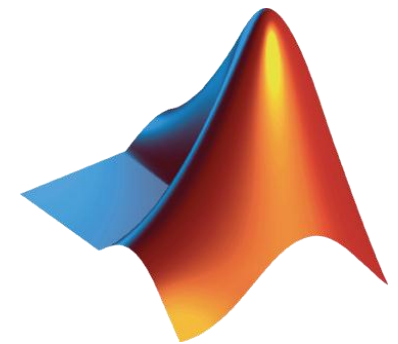
- Profilerで一番ネックとなっている部分を狙う
 - 関数呼び出し回数
 - 実時間
- Functionの呼び出し
 - 適切な関数を見つけるのに、試行錯誤と経験が必要
 - 似ている関数を比較する
 - (例えば `fgetl` vs. `fread`、`fscanf`、`textread`、`textscan` . . .)
 - カスタム関数を書く
 - MATLABの関数の大半は、ソースコードが見られる

よくあるBottlenecks

- File I/O
 - ディスクへのアクセスが遅い
 - できれば、`load` と `save` を利用する
- 結果の表示
 - Figureの作成は、時間がかかる
 - Command Windowへの出力は時間がかかる（‘;’ で抑える）
- 単純に、演算が多い

演算の重いアルゴリズムの高速化プロセス

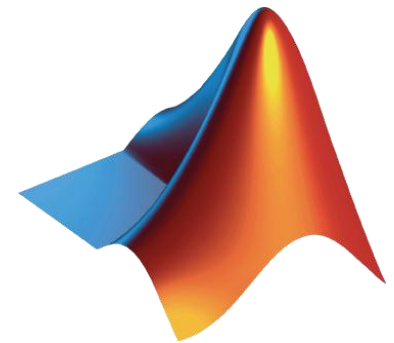
- とりあえず、エラーのないコードを書く（遅くてもいい）
- 今日習ったテクニックを利用する
- 可読性、保全性なども考慮する
- 他の言語やハードウェアを導入する
 - MEX、GP-GPUs、FPGAs、CPUクラスター、など



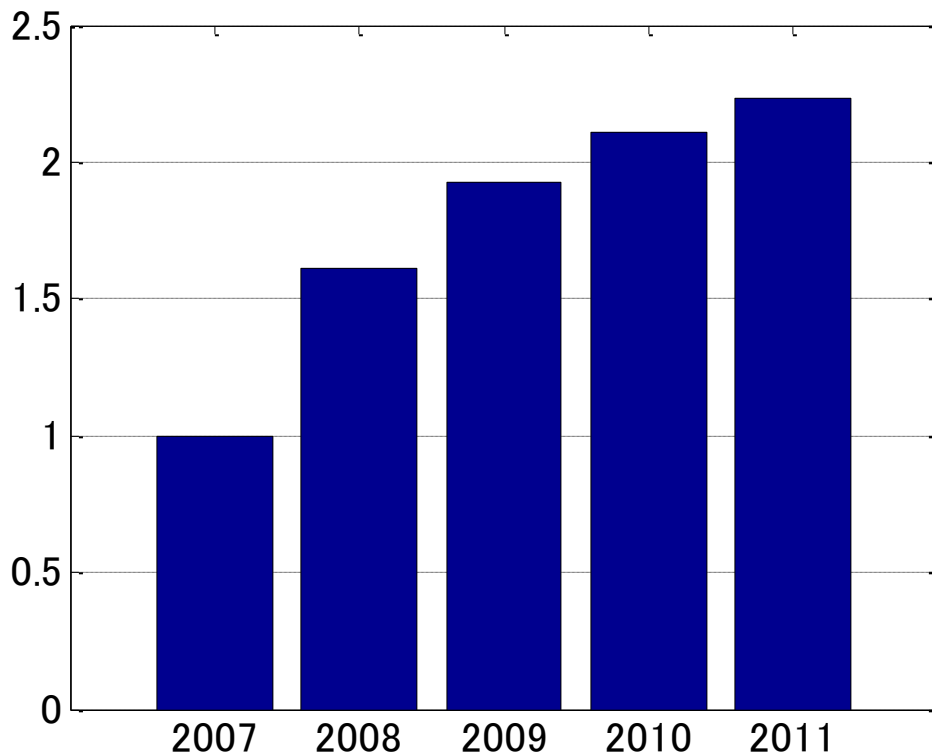
最後にもう一つの裏ワザ...

最新版に アップグレード

他に何もしなくても、ただアップグレードするだけで、
早くなることが多いです。



基本線形代数：線形システムの解



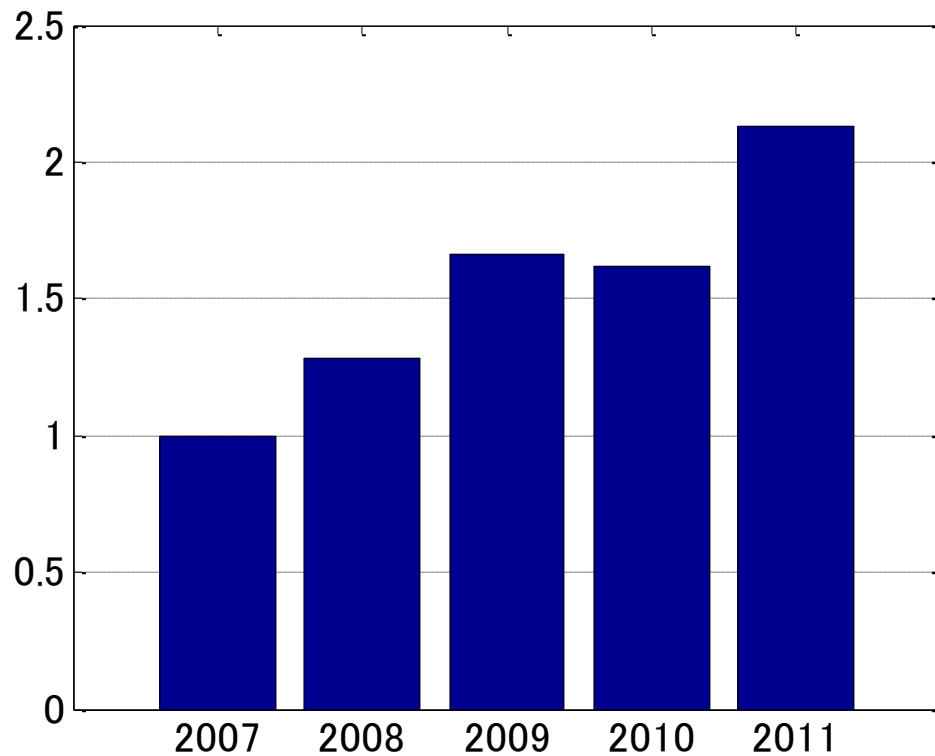
$$\begin{aligned}x_1 + 2x_2 &= 5 \\ 4x_1 + 3x_2 &= 10\end{aligned}$$

$$\begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 10 \end{bmatrix}$$

$$A \vec{x} = \vec{b}$$

```
>> A = rand(1000);  
>> B = rand(1000);  
>> X = A\B;
```

基本的な数学： 大規模データの足し算、割り算

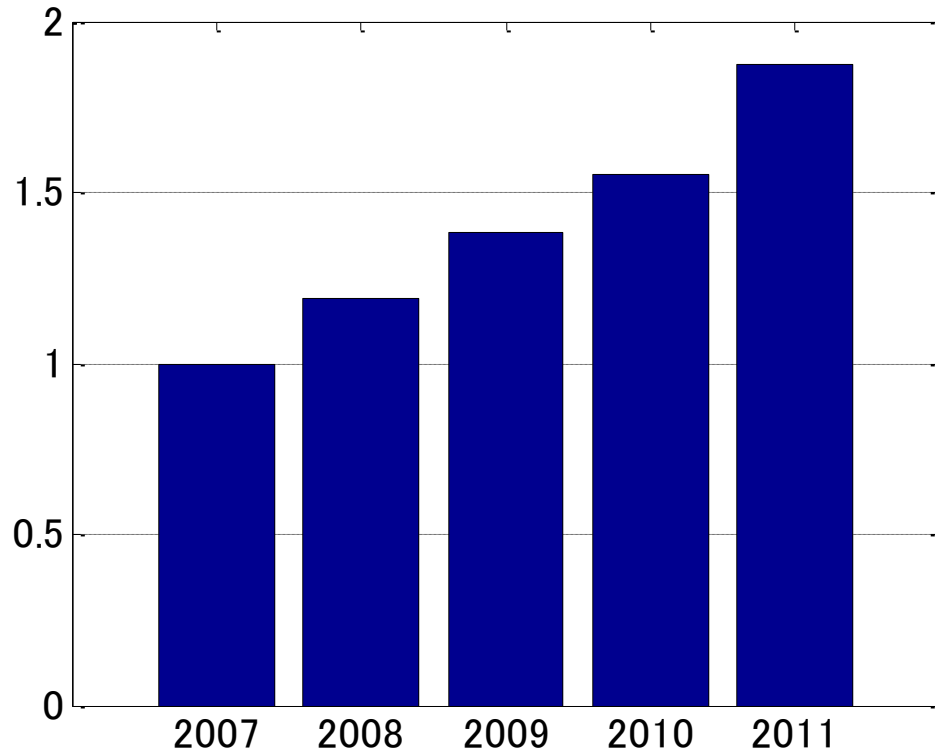


```
A = rand(2000);
```

```
A = A+A';
```

```
A = A/max(A(:));
```

FFT

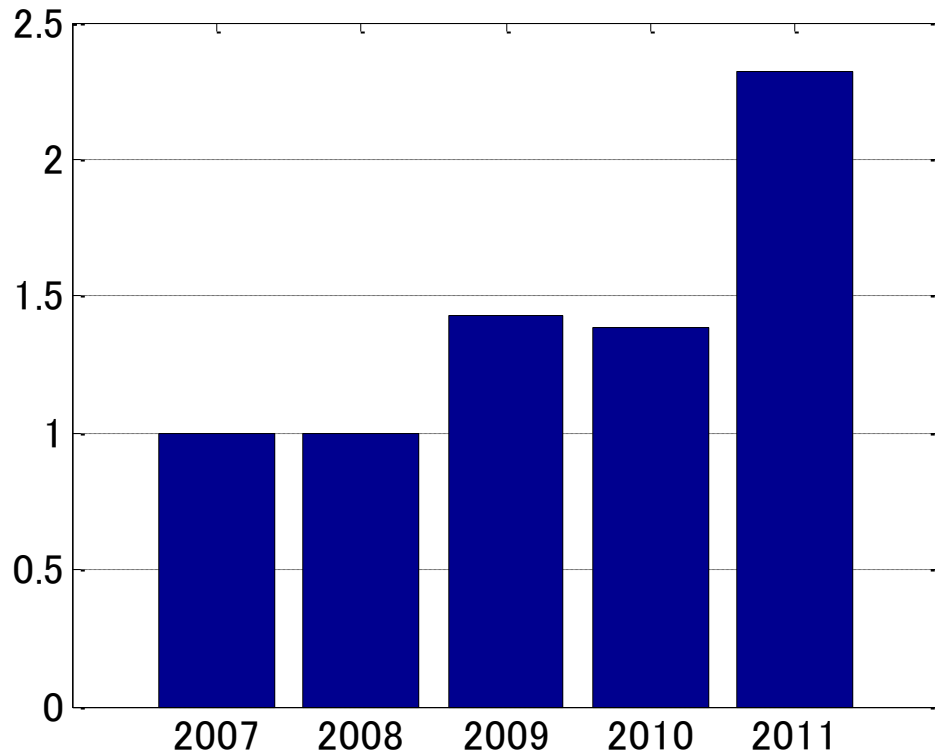


```
>> y = rand(1e6,1);  
>> Y = fft(y);
```

データの検索

$S2 = [1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1]$

$S1 = [1\ 1\ 0\ 0]$

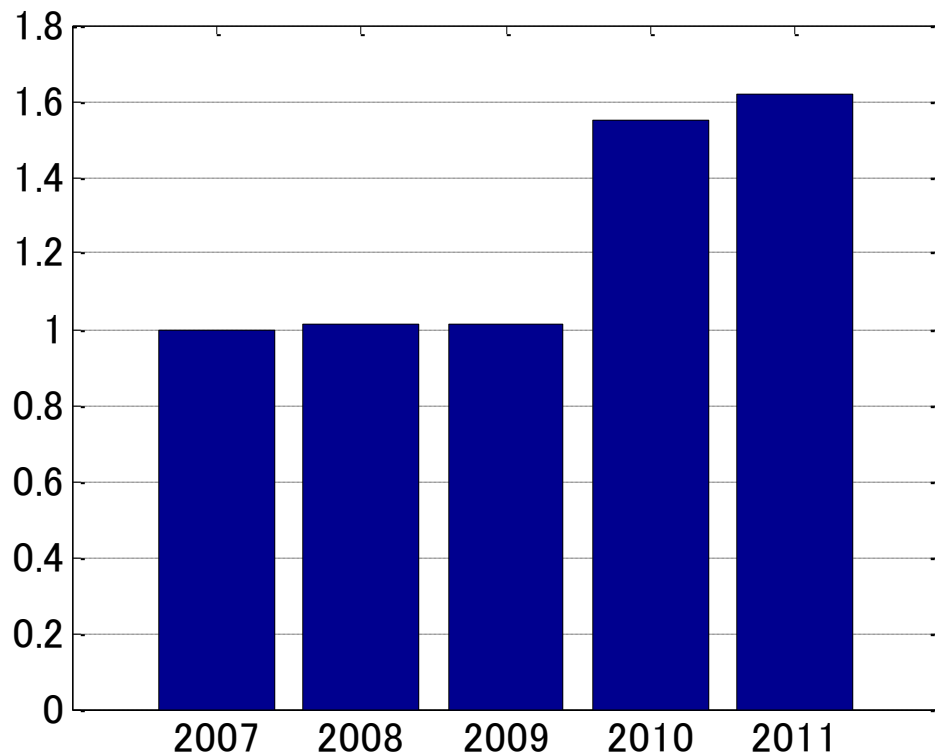


```
S1 = round(rand(1,1e6));
S2 = ones(1,10);

tic; strfind(S1,S2); toc;
```

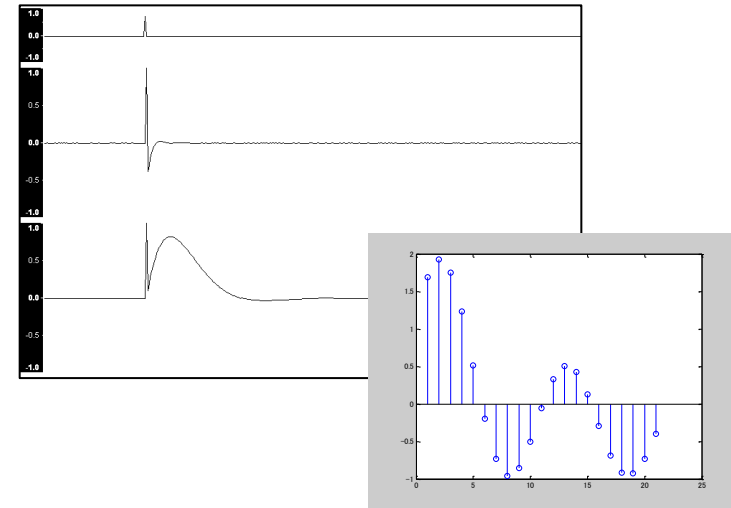
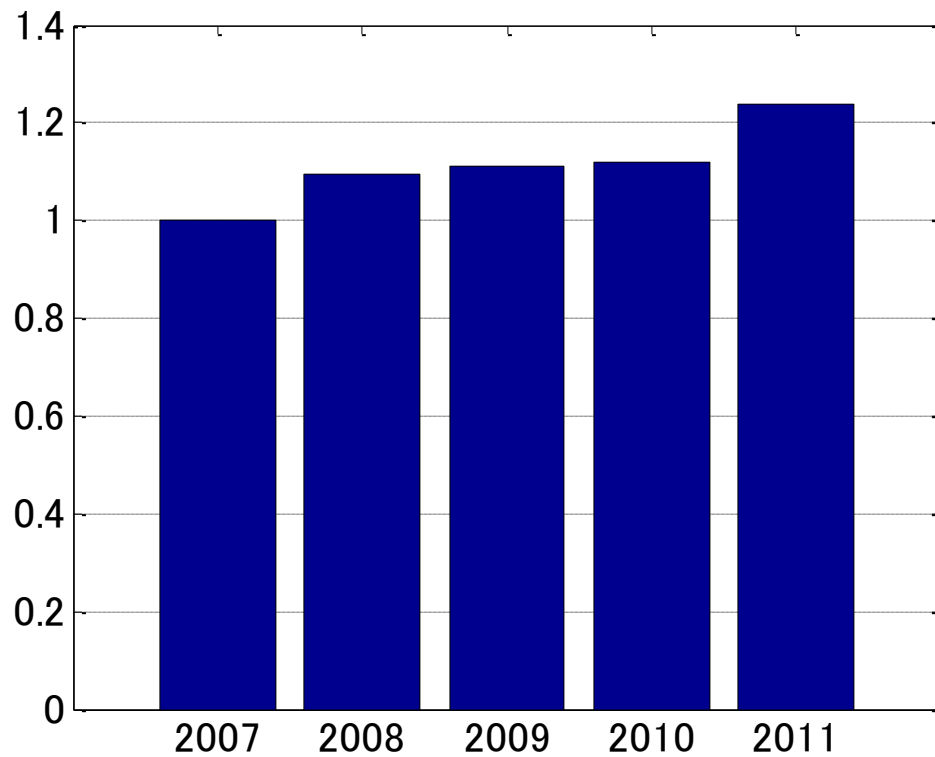

データの並べ換え

$[1.4 \ 5.6 \ 2.5 \ 4.9] \rightarrow [1.4 \ 2.5 \ 4.9 \ 5.6]$



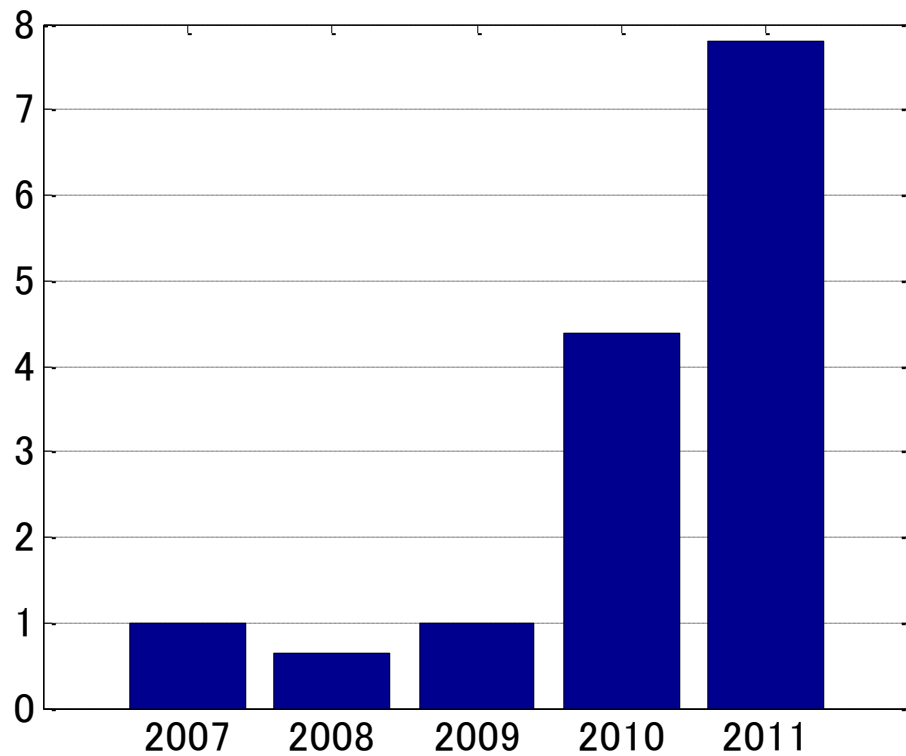
```
>> DATA = rand(1,1e6);  
>> DATA = sort(DATA);
```

動的システムのインパルス応答



```
>> R = rss(20);  
>> Y = impulse(R, 0:0.01:10);
```

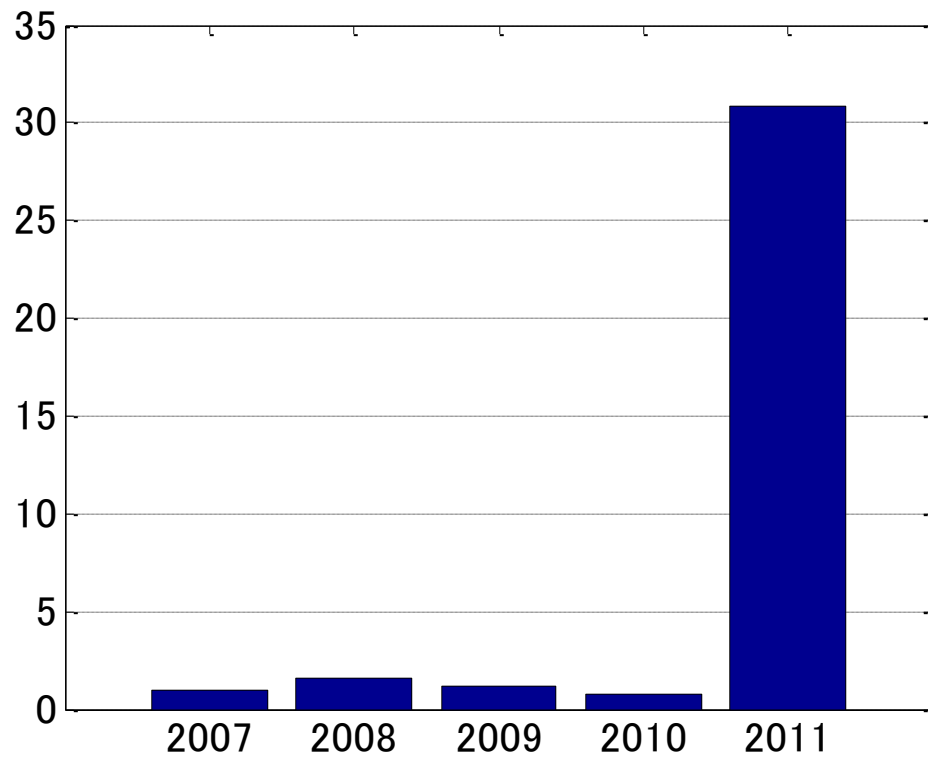
画像処理： リサイズと回転



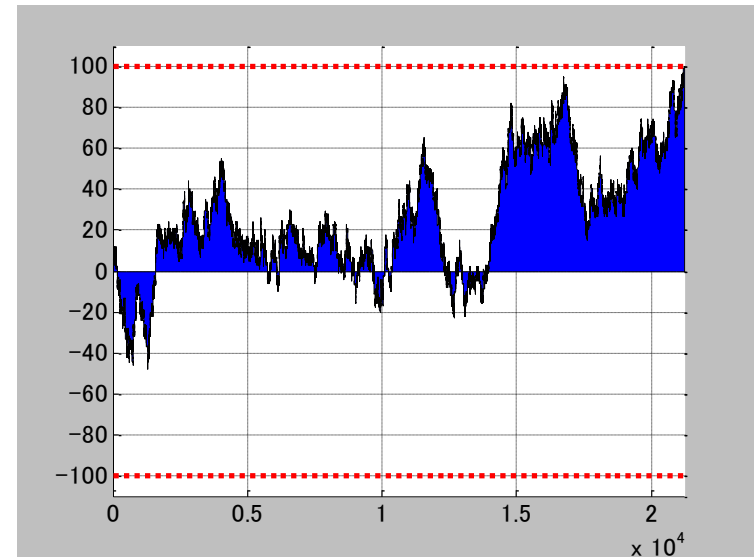
```
>> I = imread('peppers.png');  
>> I2 =  
    imresize(imrotate(I,45),2);
```

行列の自動拡大

例題: 事前割り当てが難しい、統計的シミュレーション



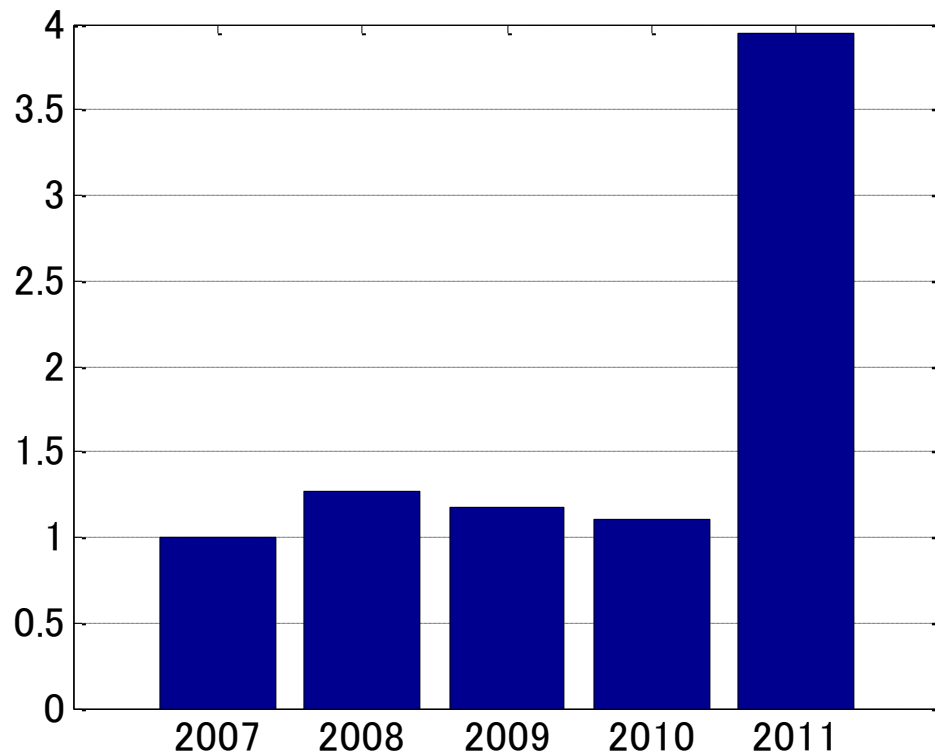
```
x = 0;  
n = 1;  
  
while abs(x(n)) < 100  
    x(n+1) = x(n) + sign(randn);  
    n = n+1;  
end
```



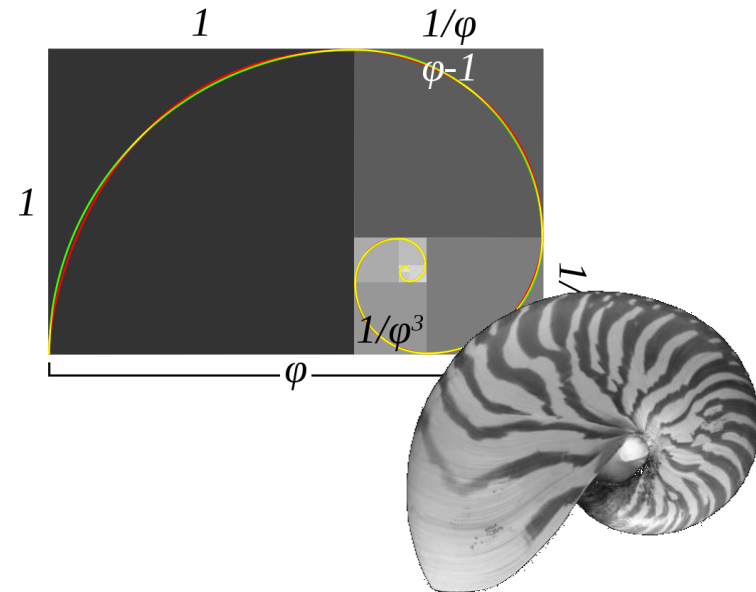
フィボナッチ数列の作成

$$x_n = x_{n-1} + x_{n-2}$$

1, 1, 2, 3, 5, 8, 13, 21, 34...

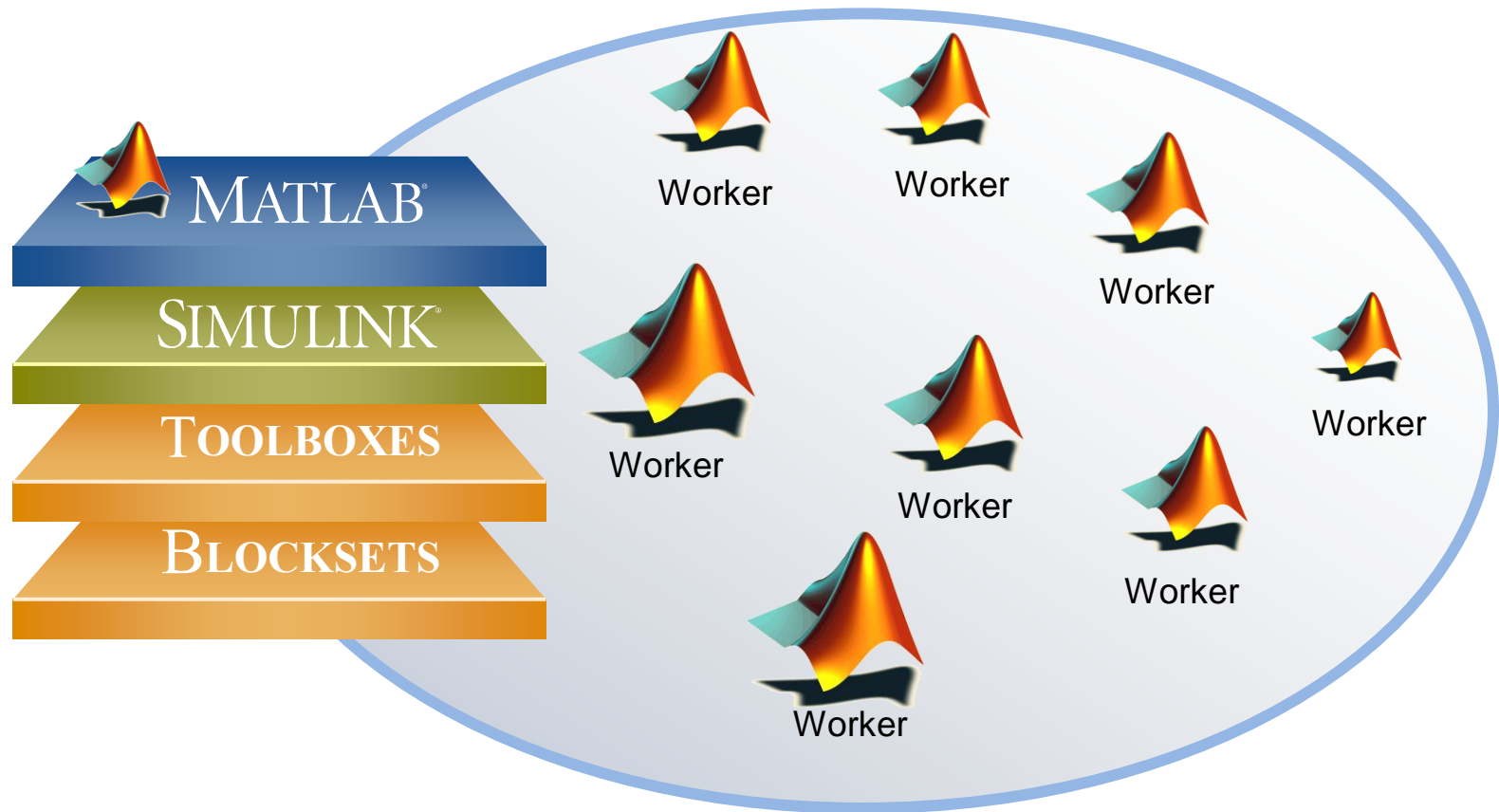


```
for n = 3:1000
    x(n) = x(n-2) + x(n-1);
end
```



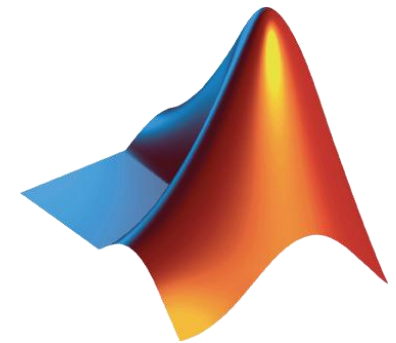
休憩

MATLABにおける並列処理 (Parallel Computing)

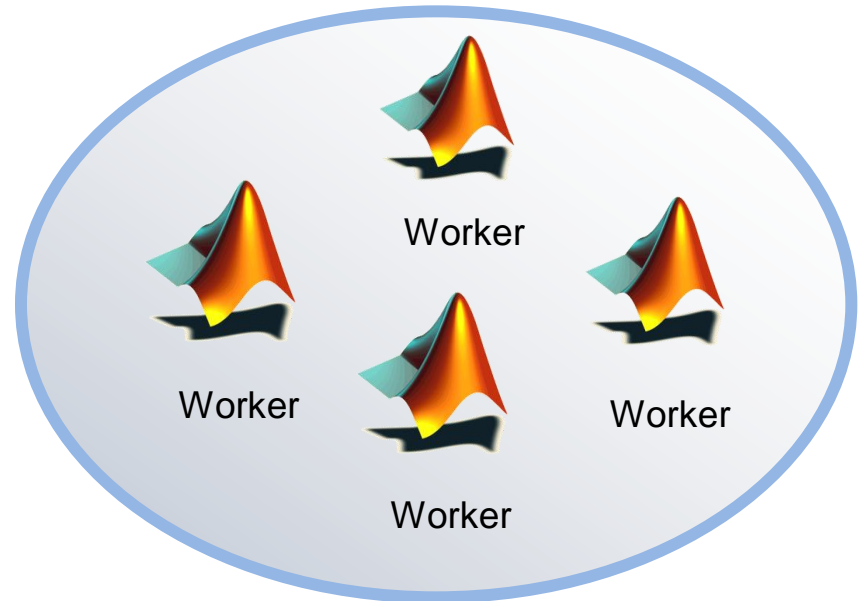
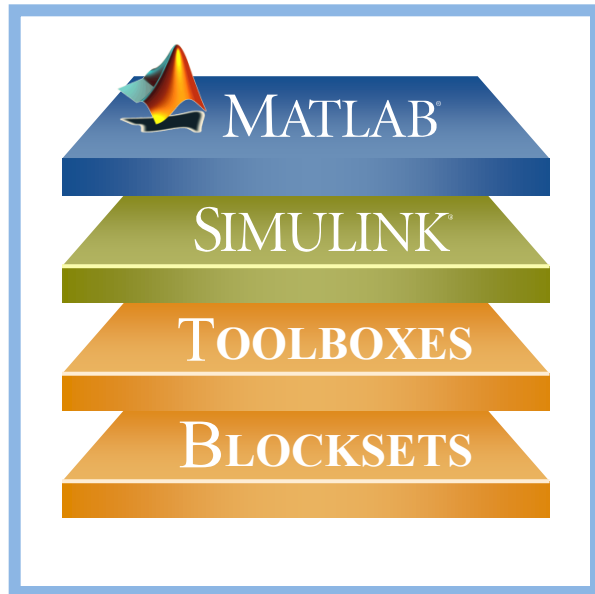


実施内容

- 並列FOR文（PARFOR）
 - 例題1：100個のファイルから最大を探す
 - 例題2：Simulink Modelのパラメータスイープ
- GPUのデモ
 - 例題3：画像処理 エッジ強調フィルターの適用
- 並列処理ソリューション全体の概要・まとめ

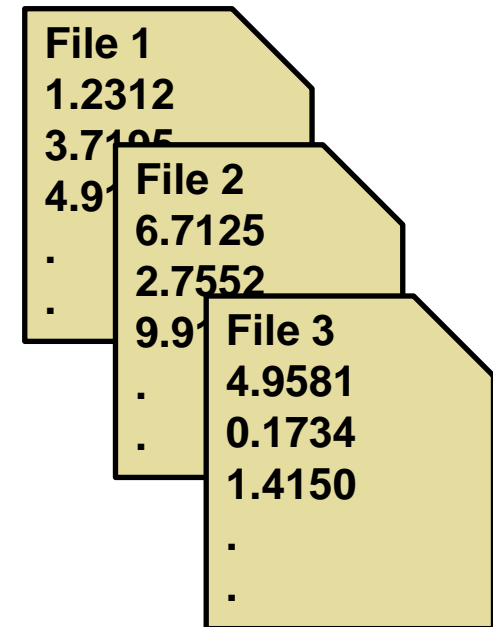


タスク・パラレル アプリケーション



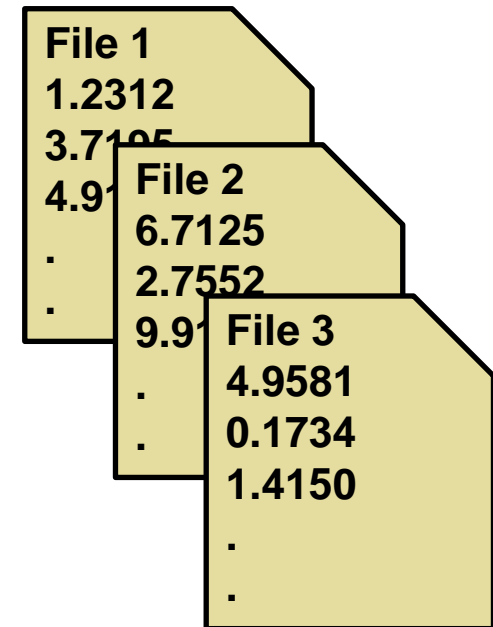
Example 1: 複数ファイルのバッチ処理

- 100個のファイル
- 各ファイルに 1万データ点がある
- 各ファイルの最大の値を抽出

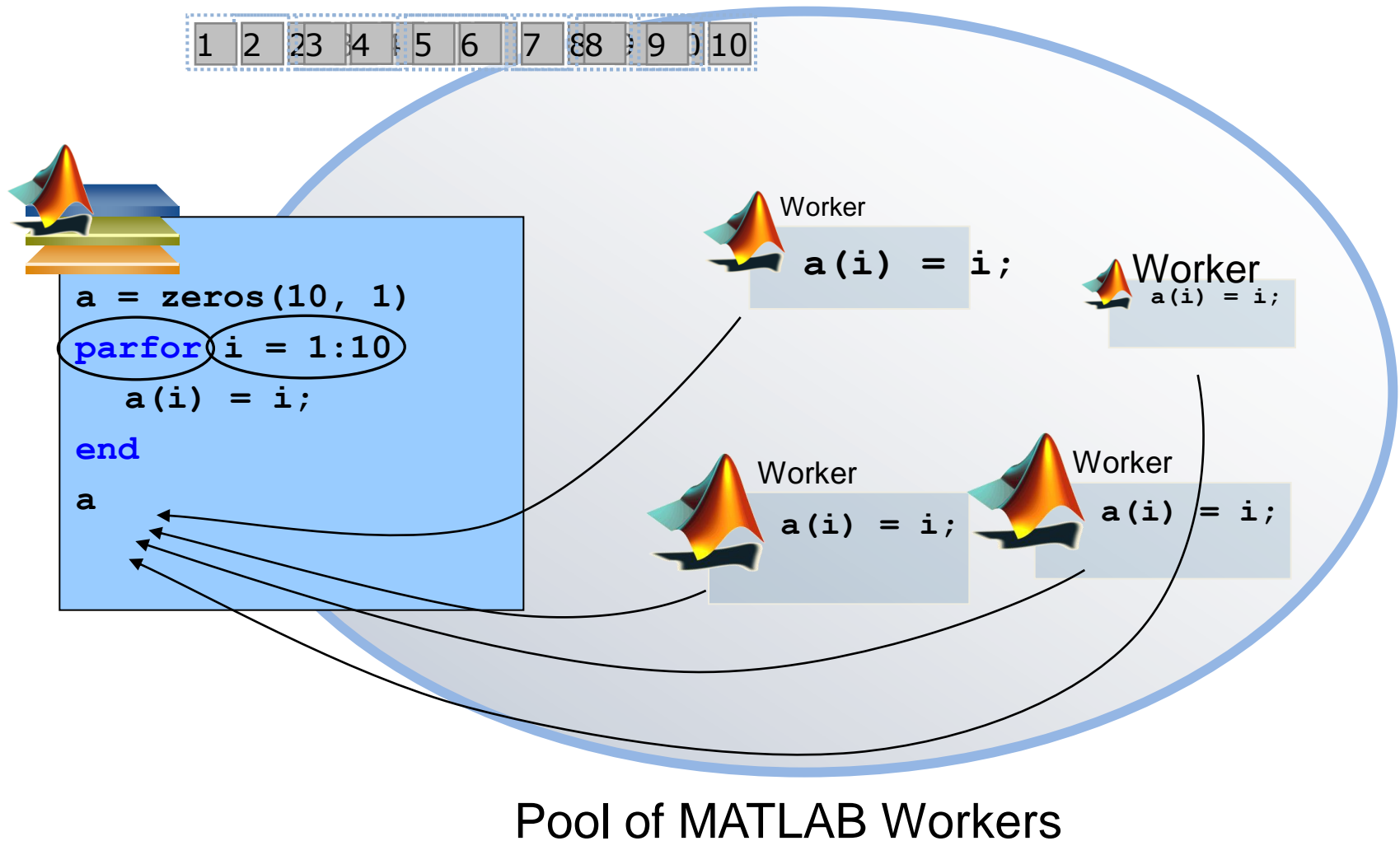


Example 1: 複数ファイルのバッチ処理

- `for` 文を`parfor`に変更
- 3x ぐらいのSpeed Up
- 並列とシリアルコードの等価性



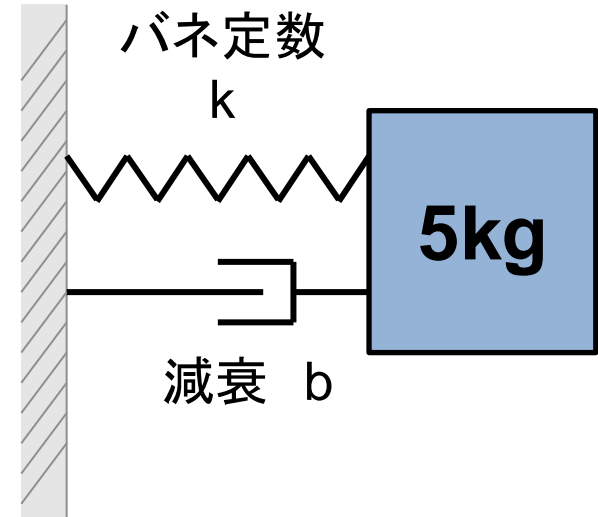
parfor とは...



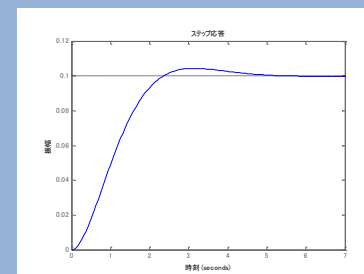
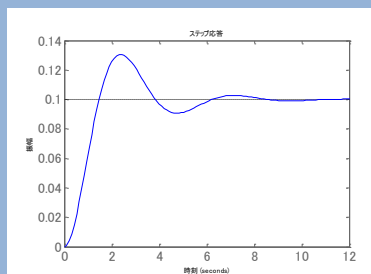
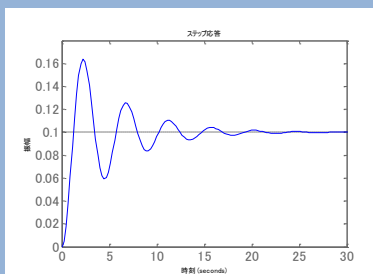
Example 2: Simulink Parameter Sweep

- 単純なバネ・マス・ダンパー系のSimulinkモデル:

$$\underbrace{m}_{5} \ddot{x} + \underbrace{b}_{1,2,\dots} \dot{x} + \underbrace{k}_{10,20,30,\dots} x = 0$$

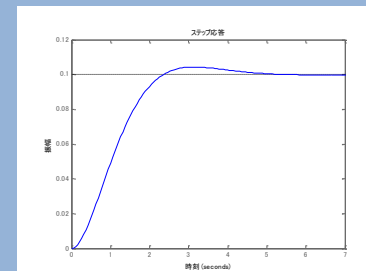
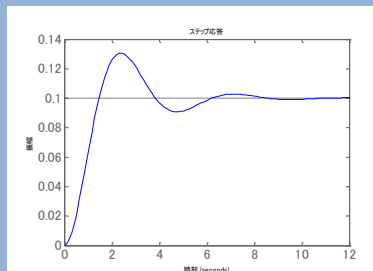
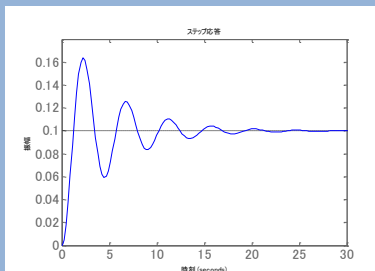


- M = 5として、“b” と “k” を変更
- 各実行の結果のデータを取る

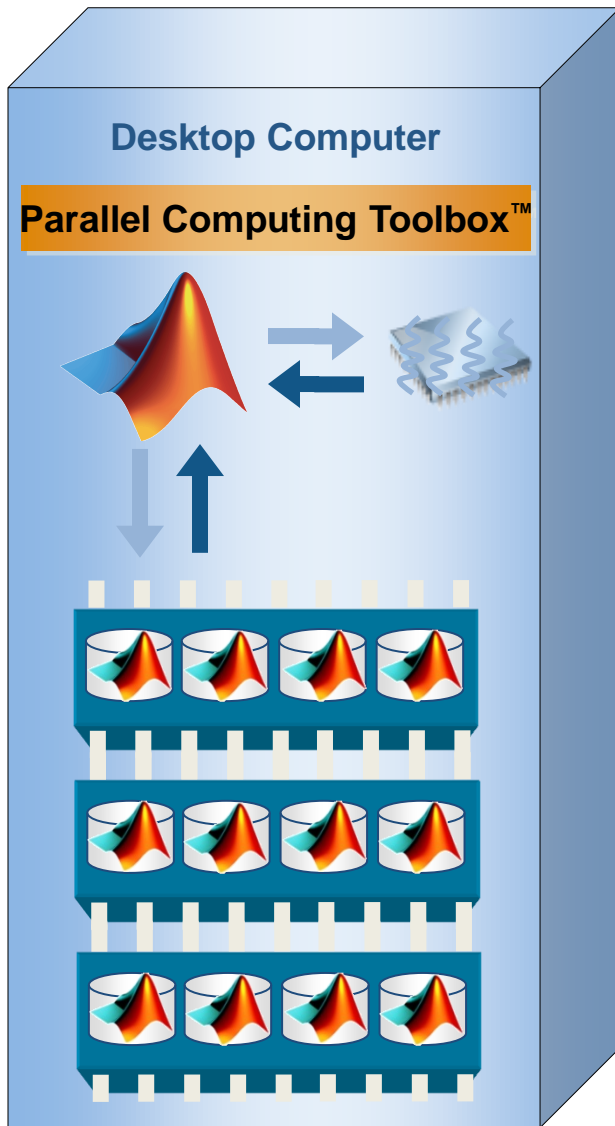


Example 2: Simulink Parameter Sweep

- `for` 文を`parfor`に変更
- 3x以上のSpeed-up
- Simulinkのモデルを開かずに実行



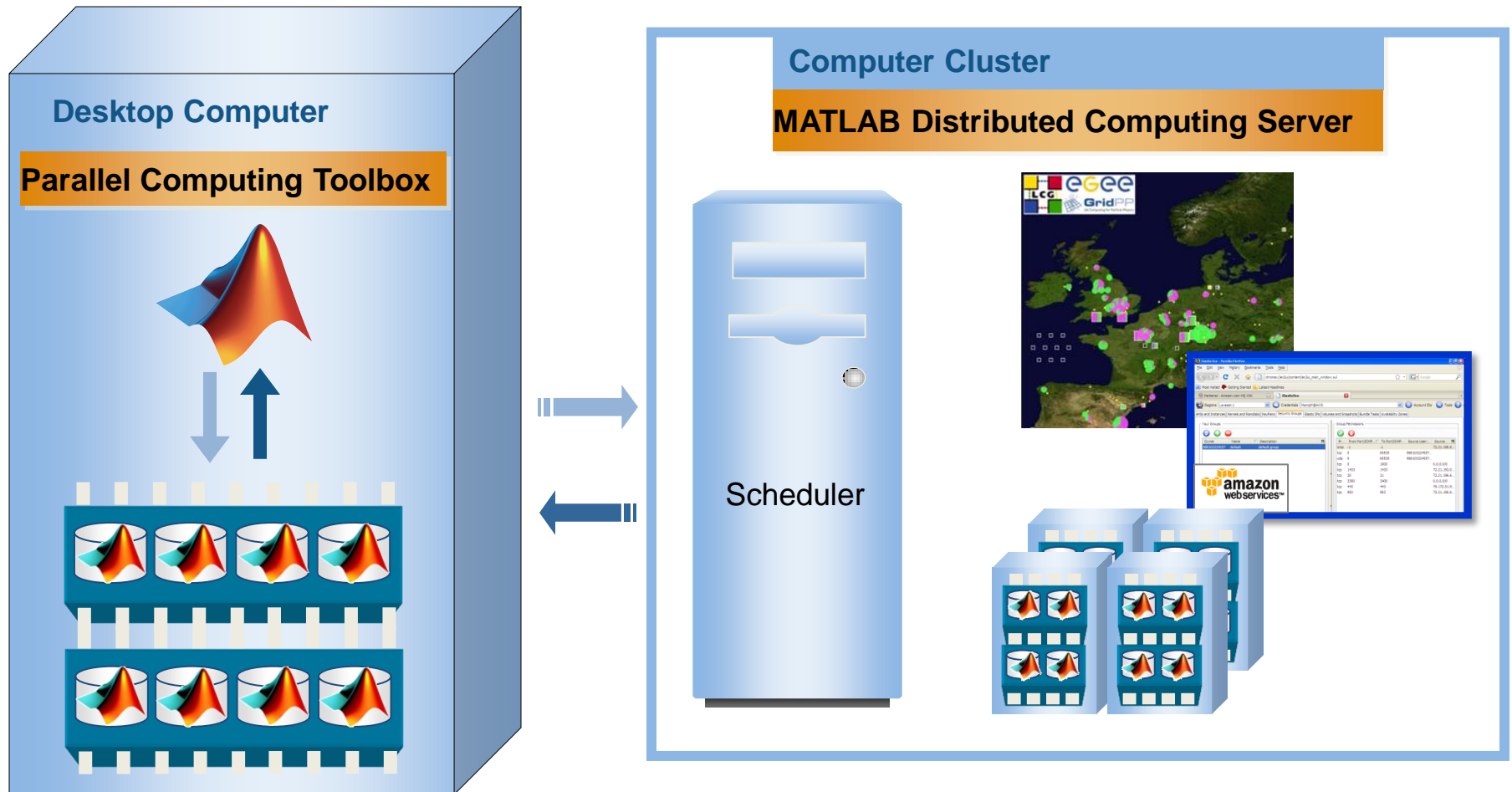
Parallel Computing Toolbox デスクトップ環境での並列計算



- Parallel Computing Toolbox™ の使用により、
- ローカルPC上でアプリケーションを並列化し
- マルチコアCPUやGP-GPU上で実行することによって高速化を行ないます
- R2011b : 最大ワーカー数を12に増加

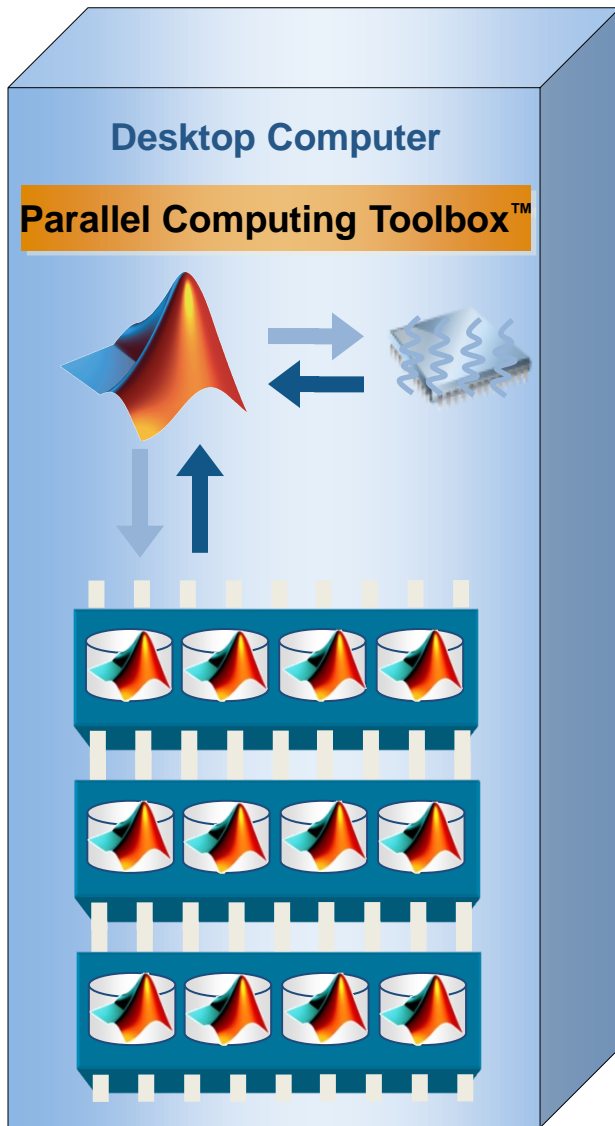
MATLAB Distributed Computing Server

クラスター、グリッド、クラウドまで...



Parallel Computing Toolbox

デスクトップ環境での並列計算



- Parallel Computing Toolbox™ の使用により、
- ローカルPC上でアプリケーションを並列化し
- マルチコアCPUやGP-GPU上で実行することによって高速化を行ないます
- R2011b : 最大ワーカー数を12に増加

Graphics Processing Unit (GPU)とは

- 本来はグラフィックス処理向けだが、近年は科学技術計算にも応用(GP-GPU)
- 整数および浮動小数点数をサポート
 - 1枚で数百のコアを搭載
 - 通常のCPUに比べ柔軟性は乏しい
- メインメモリとは独立した専用メモリを搭載



* Parallel Computing Toolbox は Compute Capability 1.3 以上を要求

Example 3: GPUを利用した画像処理

- エッジ強調フィルター
- CPUとGPUの比較



Example 3: GPUを利用した画像処理

- GPUでは2x
スピードアップ
- プログラミングが簡単
(gpuArray と gather)
- 利用できる関数が増えている



並列処理で解決できる問題:

Task-Parallel

計算時間が長い

- 独立した繰り返し

```
parfor i = 1 : n
    % i を使って何かをする
end
```

- 全く違う仕事も振り分けられる

Task 1**Task 2****Task 3****Task 4**

Data-Parallel

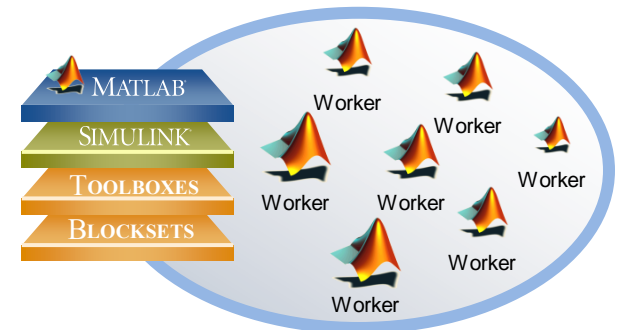
メモリが足りない

11	26	41
12	27	42
13	28	43
14	29	44
15	30	45
16	31	46
17	32	47
17	33	48
19	34	49
20	35	50
21	36	51
22	37	52



PCTと自動的に連携する製品

- Statistics Toolbox
- Optimization Toolbox
- Global Optimization Toolbox
- SystemTest
- Simulink Design Optimization
- Bioinformatics Toolbox
- Model-Based Calibration Toolbox
- ...



プログラミングなしに、直接Parallel Computing Toolboxを利用

Summary

- 事前割り当て、行列演算、特別関数を利用して、コードを高速化します
- プロファイラを利用して、コードのもっとも遅い部分を探す
- 並列計算を利用して、大きい問題の計算負荷を分けて取り組む

ご清聴ありがとうございました

