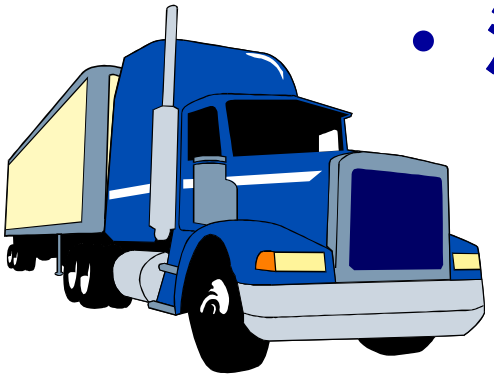
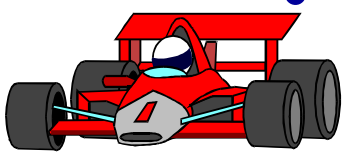
計算機プログラミングI 2002年度

(余談)プログラムの高速化の一方法: 部分計算 (partial evaluation)

プログラムの汎用性と速度

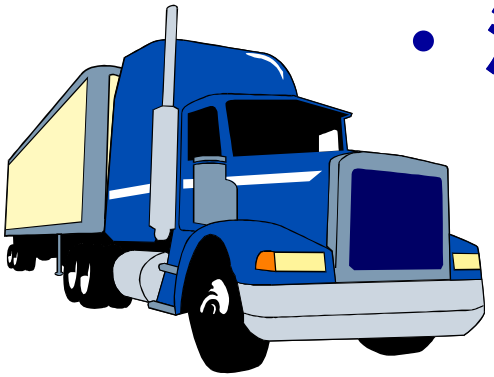


- 汎用プログラム
  - 色々な用途に使えるが、遅い
  - 例: 三次元形状編集ソフト



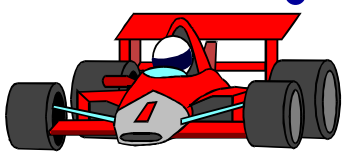
- 専用プログラム
  - 用途は限られるが、速い
  - 例: 三次元ゲーム

プログラムの汎用性と速度



- 汎用プログラム
  - 色々な用途に使えるが、遅い
  - 例: 三次元形状編集ソフト

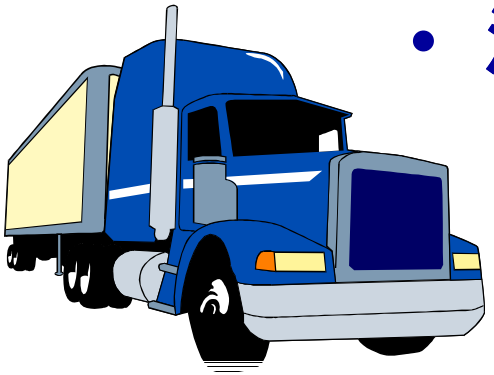
自動的に
作れないか?



- 専用プログラム
  - 用途は限定的
  - 例: 三次元ゲーム

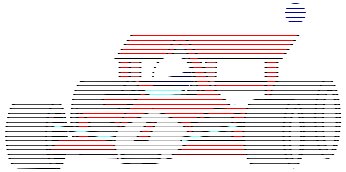
作るのは大変

プログラムの汎用性と速度



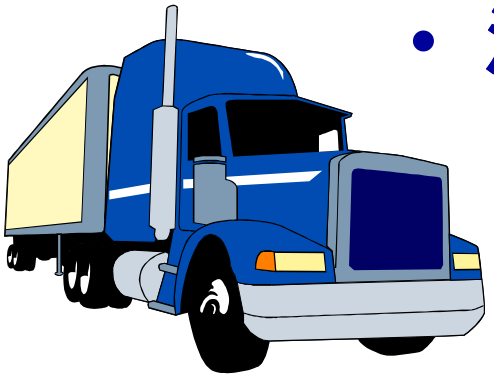
- 汎用プログラム
  - 色々な用途に使えるが、遅い
  - 例: 三次元

何故遅い?
- 機能が多い
- 機能を選びながら実行
- 例: 形・色・影・・・
- (ハード vs. ソフト)



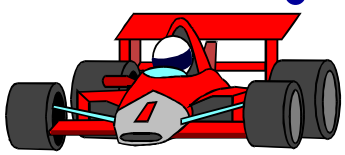
- 専用プログラ
  - 用途は限ら
  - 例: 三次元ゲーム

プログラムの汎用性と速度



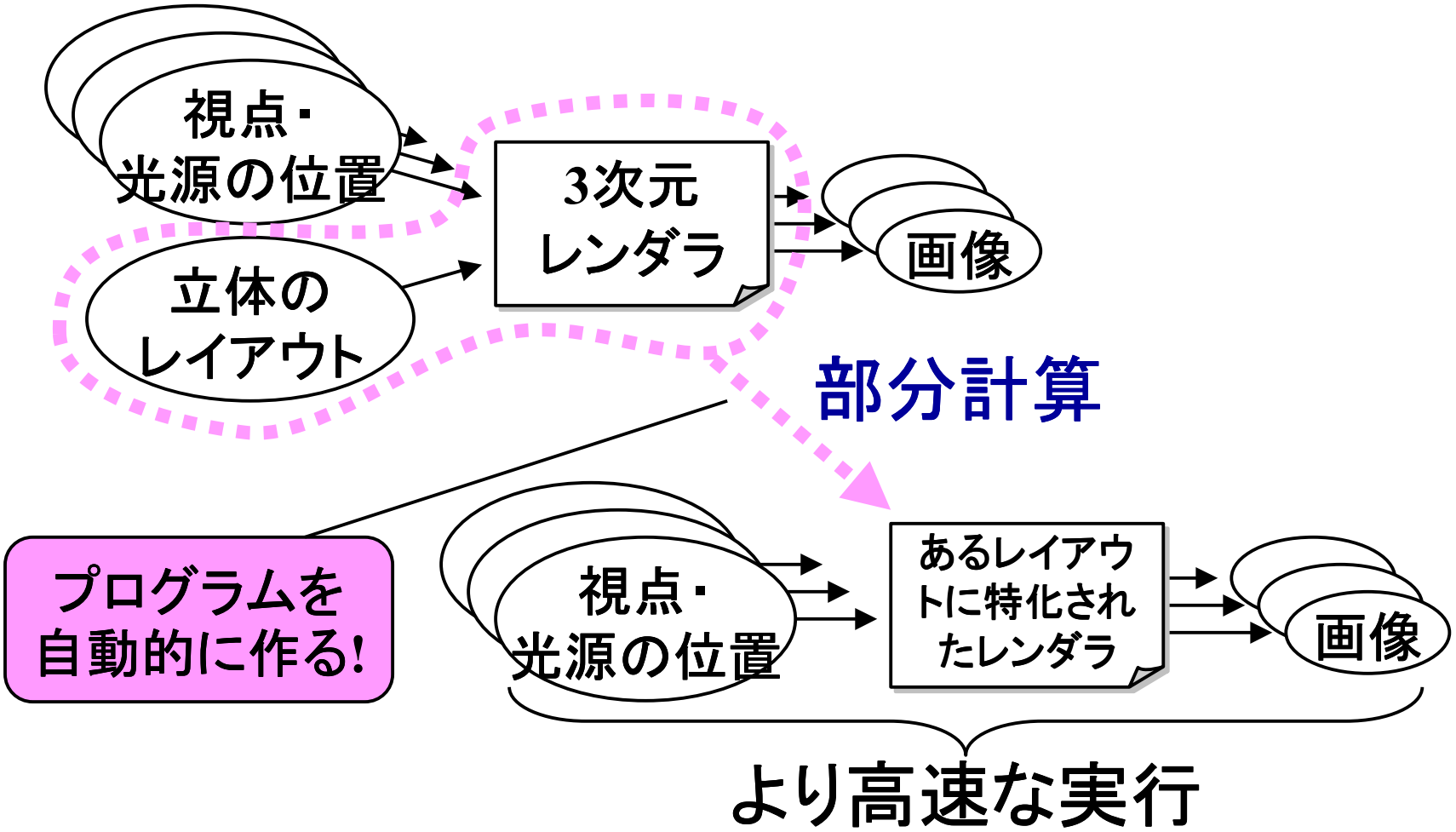
- 汎用プログラム
  - 色々な用途に使えるが
  - 例: 三次元形状編集ソ

不要な処理(機能)を削って自動的に作る

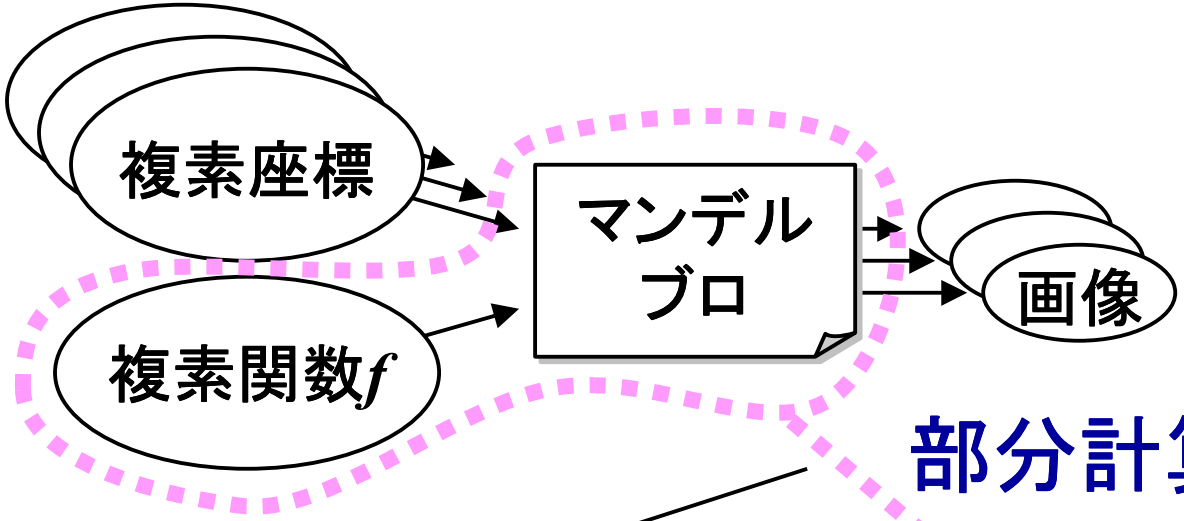


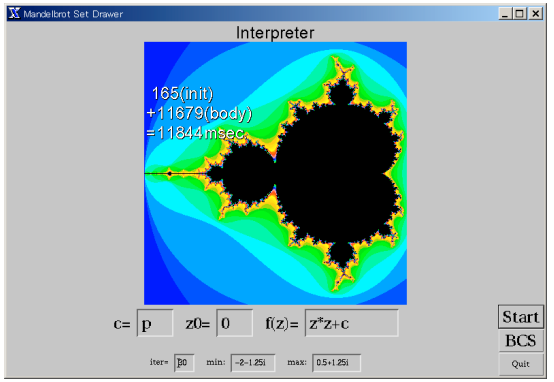
- 専用プログラム
  - 用途は限られるが、速い
  - 例: 三次元ゲーム

部分計算の例 [Guenter95]



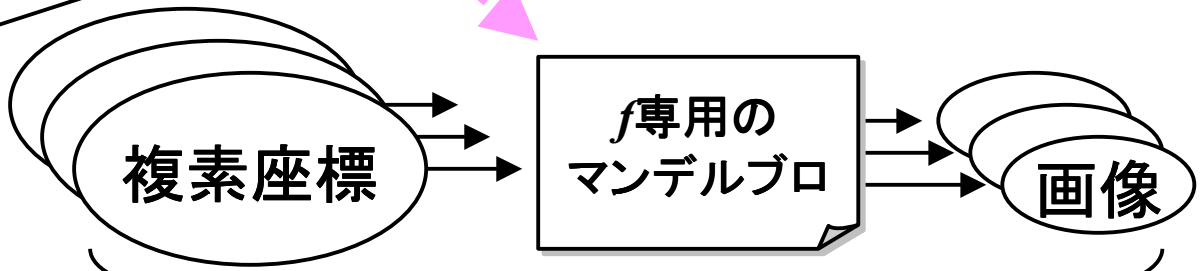
部分計算の例





部分計算

プログラムを
自動的に作る!



より高速な実行

部分計算: どうして速くなる?

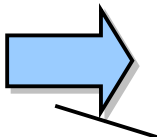
- 一般的に書かれたプログラム
  - 汎用性を高めるために
    沢山のパラメータを利用
  - パラメータの値を見ながら動作を変える
- 実際の使用
  - 特定のパラメータの組み合わせしか使わない
  - パラメータを「決め打ち」した
    プログラムがあれば速い

部分計算の方法

- どうやって自動的に特化するか?
  → 部分的に計算をする
  ・・・残った計算が特化されたもの

例題: “*x* の *n* 乗”を *n*=3 として特化

```
int power(x, n) {
  if(n==0) return 1;
  else return
     x*power(x,n-1);
}
```



```
int power_3(x) {
  return x*x*x;
}
```

値のかわりに
プログラムを
返す計算

部分計算の方法
まずは普通に実行してみる

```
int power(x, n) {
  if(n==0) return 1;
  else return
     x*power(x,n-1);}
```

- power(2,3)を計算
- x=2, n=3として
  if(n==0)... else ...
- n!=0なので
  return
   x*power(x,n-1);

- return
  x*power(x,n-1);
  - x ⇒ 2
  - power(x,n-1)
    - x ⇒ 2
    - n-1 ⇒ 2
    - power(2,2)
      - x=2,n=2として
        本体を計算
      - … ⇒ 4
  - 2*4 ⇒ 8
- ⇒ 8

部分計算の方法: 例

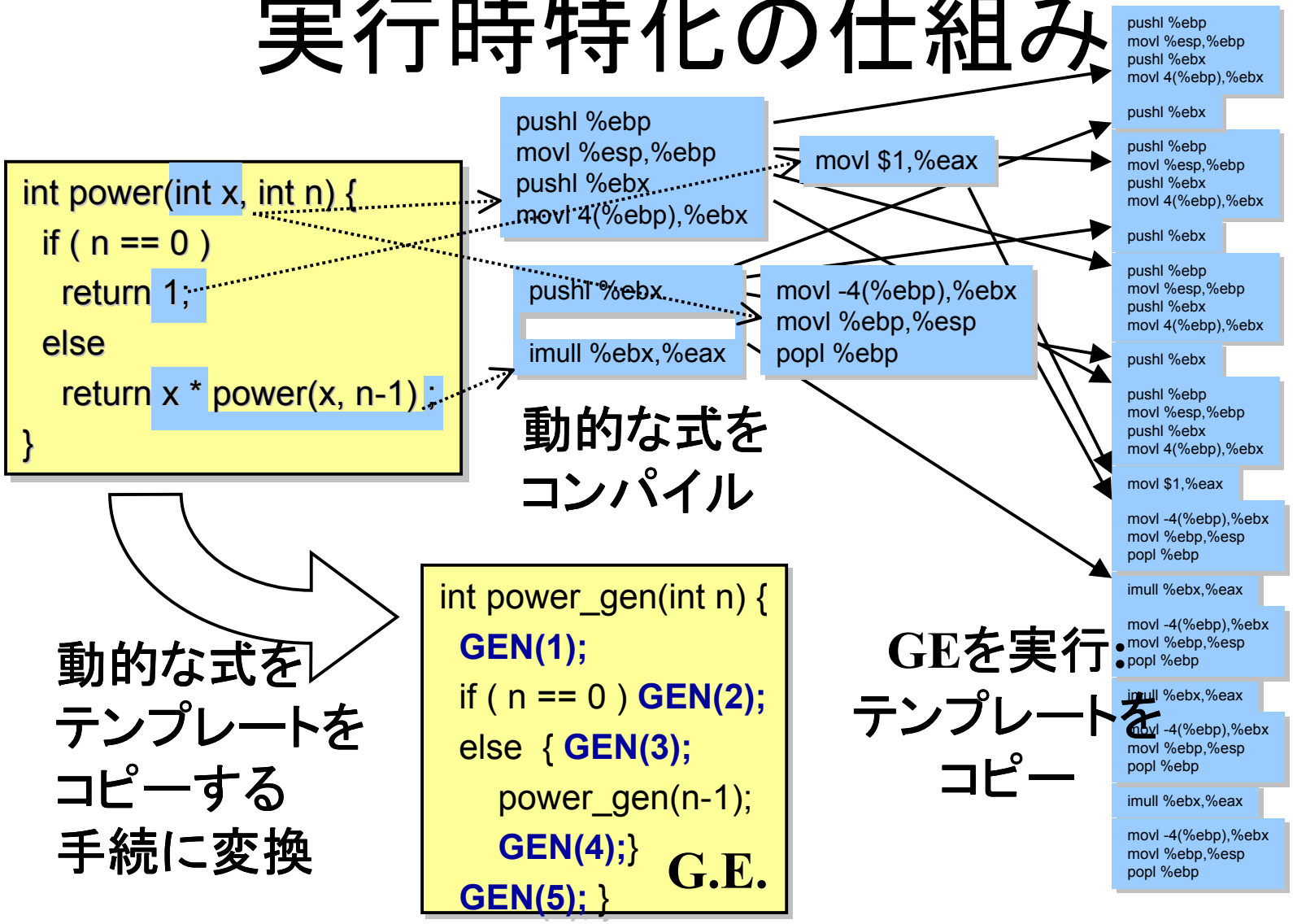
```
int power(x, n) {
  if(n==0) return 1;
  else return
     x*power(x,n-1);}
```

- power(“x”,3)を計算
- x=“x”,n=3として
  if(n==0)...else...
- n!=0なので
  return
   x*power(x,n-1);

「式」を値として計算

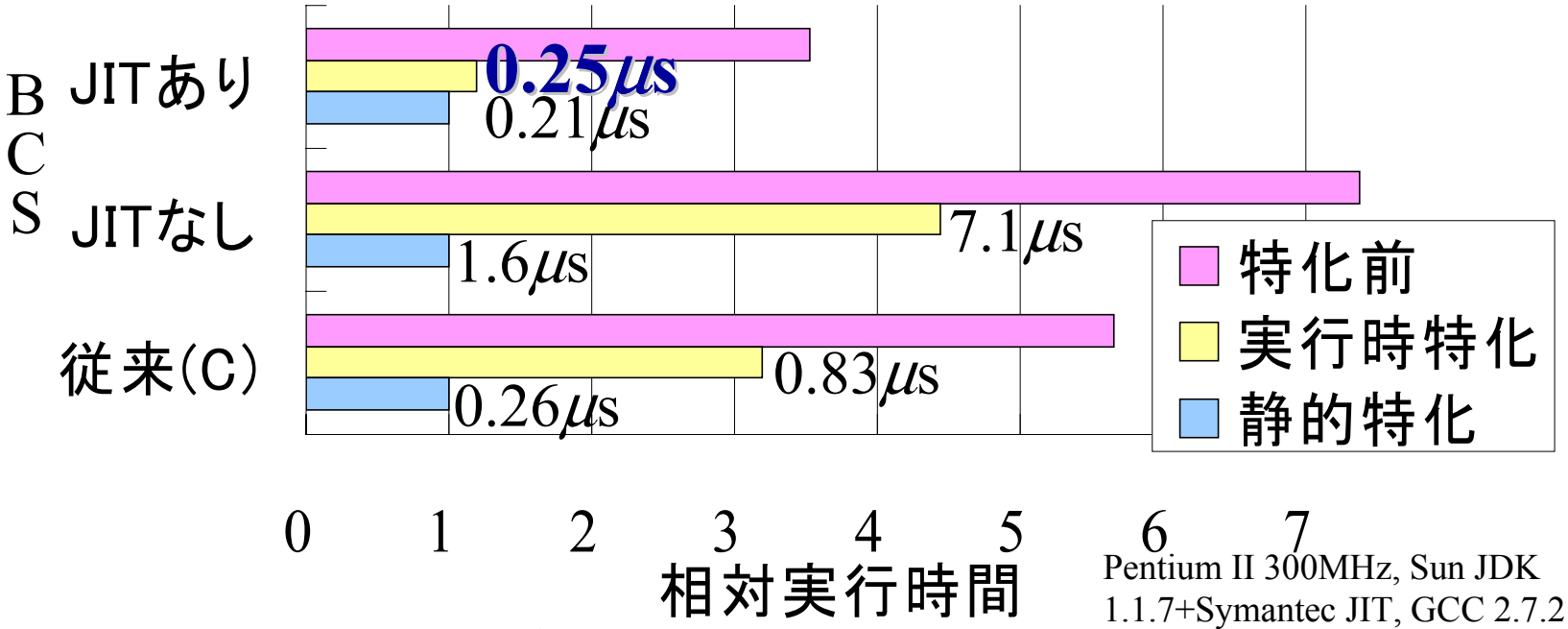
- return
  x*power(x,n-1);
  - x ⇒ “x”
  - power(x,n-1)
    - x ⇒ “x”
    - n-1 ⇒ 2
    - power(“x”,2)
      - x=“x”,n=2として
        本体を計算
      - … ⇒ “x*x”
  - “x” * “x*x”
    ⇒ “x*x*x”
- ⇒ “x*x*x”
- int power_3(x)
  { return x*x*x; }

実行時特化の仕組み


int power(int x, int n) {
if (n == 0)
return 1;
else
return x * power(x, n-1) ;
}
pushl %ebp
movl %esp,%ebp
pushl %ebx
movl 4(%ebp),%ebx
movl $1,%eax
pushl %ebx
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
imull %ebx,%eax
動的な式を
コンパイル
動的な式を
テンプレートを
コピーする
手続に変換
int power_gen(int n) {
GEN(1);
if (n == 0) GEN(2);
else { GEN(3);
power_gen(n-1);
GEN(4);}
GEN(5); }
G.E.
GEを実行:
テンプレートを
コピー

バイトコード特化処理系の性能

静的に特化した場合とほぼ同じ速度を得ている



Pentium II 300MHz, Sun JDK 1.1.7+Symantec JIT, GCC 2.7.2

※特化に要した時間は含まない