連立一次方程式は
どこまで速く解けるのか？

Commodity PC clusterと
並列多倍長数値計算

静岡理工科大学
幸谷智紀
http://na-inet.jp/

概要

- **多倍長数値計算とは？**
- Commodity PC cluster**とは？**
- MPI**とは？**
- BNCpack**とは？**

教科書的な常識

- Conjugate-Gradient(CG)**法とは？**
- CG**法の最短計算時間は？**
- **まとめ**

ヨタ話

多倍長数値計算とは？

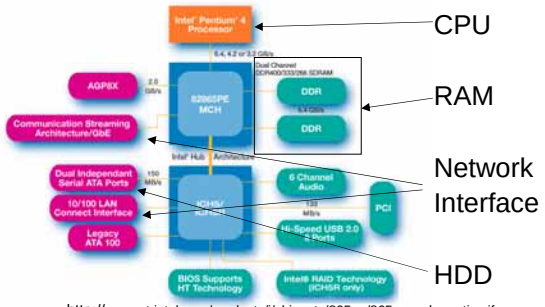
今回使用するPC (Cluster)

x86_32 **アーキテクチャ**

- Pentium4 ･･･ Intel Pentium IV 2.8cGHz (11 nodes, 11PEs, Vine Linux 3.2)
- Xeon･･･ Intel Xeon 3.0GHz(**名古屋大学三井研**, 8 nodes, 16PEs, Redhat 8)
- PentiumD ･･･ Intel Pentium D 820 (2.8GHz, 4 nodes, 8PEs, Fedora Core 5 x86_64)
- Athlon64X2 ･･･ AMD Athlon64 X2 3800+ 2.0GHz (**幸谷私物**, Fedora Core 4 x86_64)

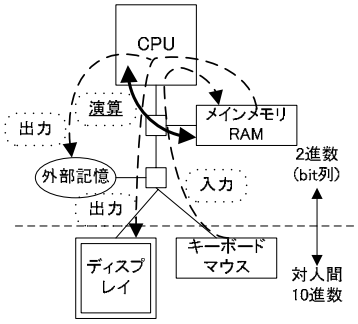
x86_64 **アーキテクチャ**

今のパソコンのアーキテクチャ
(Pentium 4 + 865PE chipset)

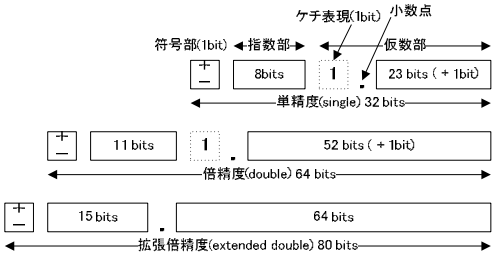


http://support.intel.com/products/i/chipsets/865pe/865pe_schematic.gif

コンピュータの演算



IEEE754-1985 standard



- **2進数ベースの浮動小数点**(FP)**数の国際規格**
- CPU**内のFP演算unitで高速処理**
- IEEE754 double(**倍精度**)**が通常使用される精度**：10**進約16桁**

IEEE754 double計算でも不十分

- ある大御所のお言葉「8倍精度で十分」・・・ホントか？
- 悪条件問題
  - 連立一次方程式の条件数が10^15以上
  - 近接根を持つ代数方程式の数値計算
- **実用になるかどうかはともかく，桁を任意に変動させてみないと，問題やアルゴリズムの持つ性質は分からない！**
- Krylov部分空間法・・・連立一次方程式
- Lanczos法・・・行列のreduction(固有値問題)

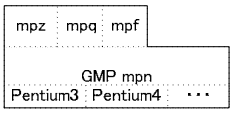
→double以上の精度(仮数部の桁数)を持つFP数をソフトウェアで実装する＝多倍長浮動小数点演算(多倍長計算)

多倍長計算の実装

- 「多倍長計算」「multiple precision」でGoogleると山ほど出てくる(円周率πの計算など)
- 「作るのは簡単，速くすることが難しい」
- IEEE754 FP数を繋げて桁を長くする
  - ARPREC (C++/Fortran90)
    http://crd.lbl.gov/~dhbailey/mpdist/arprec.tar.gz
- 多倍長「自然数」演算を積み上げて多倍長FP演算を実装する
  - GNU MP(GMP) とそのファミリーhttp://www.swox.com/gmp/
  - MPFR/GMP・・・GMPの自然数(mpn_*)ライブラリを土台にした多倍長FP数ライブラリ

GMPの特徴

- 高速な自然数演算(mpn_*)を，CPUアーキテクチャごとに最適化して実現
- ANSI Cレベルでの実装＋CPUアーキテクチャごとの最適化(アセンブラ)
- x86系CPUは，MMX/SSE/SSE2を利用して高速化
- 整数(mpz_*), 有理数(mpq_*), FP数(mpf_*)を実現

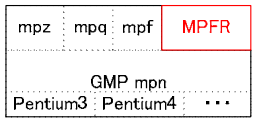


- GMPのFP数は実用的には機能が不足

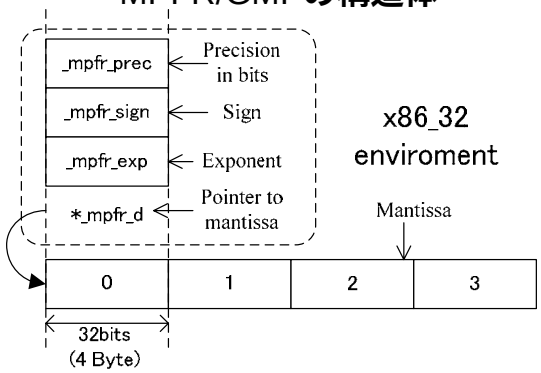
→MPFR

MPFRの特徴

- http://www.mpfr.org/
- ベースの自然数演算はGMPを利用・・・MPFR/GMPと呼ぶべき
- IEEE754 standardの上位互換
  - 4種類の丸めモード(GMPはRZのみ)
  - 無限大(±Inf), 非数(NaN)
  - 初等関数
- 特殊関数も実装
- 四則演算と初等関数では世界トップレベルの速さ
  http://www.medicis.polytechnique.fr/~pphd/mpfr/timings-220.html
- GMP Version 4までは同梱されていた→2004年1月以降は分離(GMP側の偏狭さが原因？)



MPFR/GMPの構造体



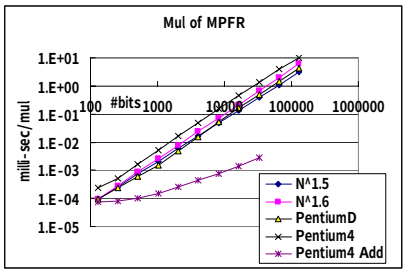
自然数の乗法アルゴリズム

Bit数 N に応じて4つのアルゴリズムを実装

1. Basecase乗算(筆算と同じ)・・・$O(N^2)$
2. Karatsuba乗算・・・ $O(N^{1.585})$
3. Toom-Cook 3way乗算・・・ $O(N^{1.465})$
4. FFT乗算(Option)・・・ $O(N^{1.333〜1.4})$

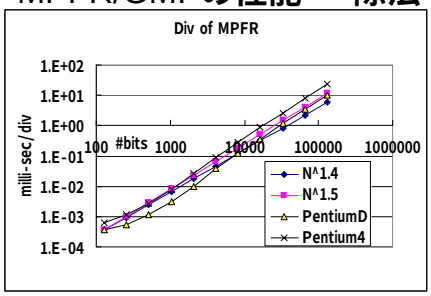
FFTが利いてくるのは少なくても約10000bits以上
(by GMP マニュアルより)

MPFR/GMPの性能・・・乗法(と加法)



# of bits	Pentium D 820	Pentium 4 2.8GHz
32768	**0.491**	**1.33**
65536	**1.48**	**3.84**

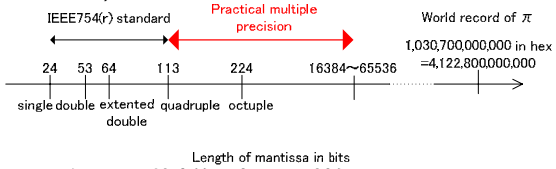
MPFR/GMPの性能・・・除法



# of bits	Pentium D 820	Pentium 4 2.8GHz
32768	1.16	2.65
65536	3.5	7.97

実用的な多倍長計算とは？

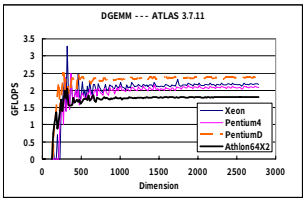
- 3つの多倍長FP数が全てキャッシュ(512KB〜4MB)に収まる(174761〜11184809bits → 何年かかるの？)
- 全ての四則演算が現状のCommodity PCで1ミリ秒以下 (10x10の行列積は1秒, 100x100だと1000秒=約17分) のうち，後者が最も実用的な基準(FFTが利くようでは非実用的？)

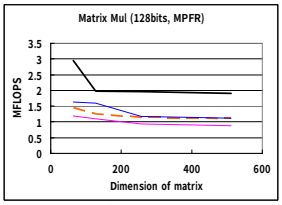


→以降は，この精度範囲内のみを対象とする。

でもやっぱり多倍長計算は遅い

- 行列積を計算し，FLOPS(乗算回数/sec)を計測
- IEEE754 doubleはATLASを使用
- MPFR 2.1.0+BNCpackを使用

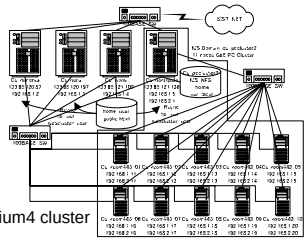




- 128bit計算でもIEEE754 doubleの1/1000以下

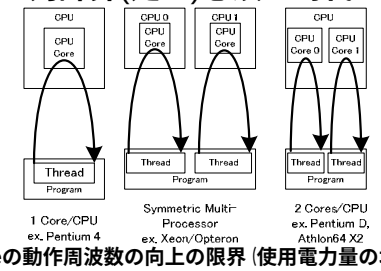
Commodity PC clusterとは？

- Commodity PC cluster = Commodity PC(ふつうのパソコン) + TCP/IP on Ethernet(安いネットワーク)
- Flynnの分類→MIMD, 分散メモリ



Pentium4 cluster

並列計算(処理)必須の時代へ



1 Core/CPU ex. Pentium 4

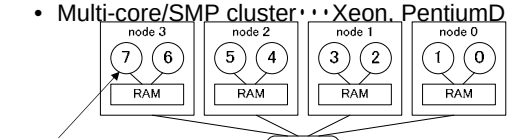
Symmetric Multi-Processor ex. Xeon/Opteron

2 Cores/CPU ex. Pentium D, Athlon64 X2

- CPU coreの動作周波数の向上の限界 (使用電力量の増大)
- 並列度を上げて性能向上を図る→ソフトウェアで対応する必要あり
- 小規模な並列計算→共有メモリ・・・Pthread, OpenMP
- 大規模な並列計算→(やっぱり)分散メモリ・・・MPI

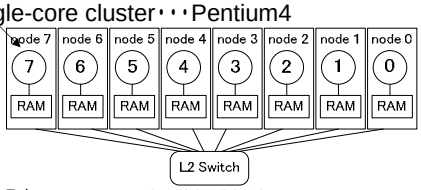
PC clusterの違い

- Multi-core/SMP cluster･･･Xeon, PentiumD

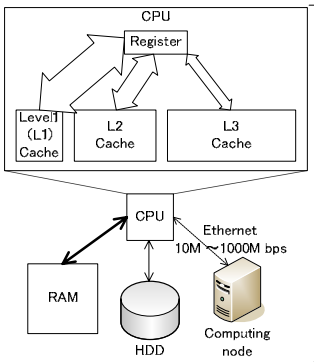


PE(Processor Element)

- Single-core cluster･･･Pentium4



メモリのヒエラルキー



データの転送速度は，PC内部においても一定でない

Register←→L2: 122GB/sec(P4 3.8GHz)

CPU←→RAM: 6.4GB/sec

CPU←→HDD:150MB/sec

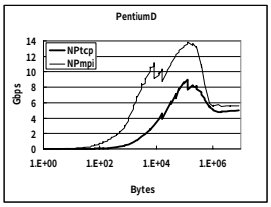
ネットワークを介すると更に遅くなる

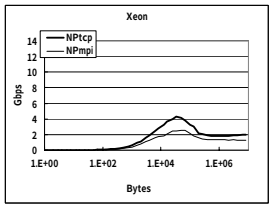
Ethernet: 10M～1000M(1G) bit/sec

PC内部のデータ転送は複雑

NetPIPEベンチマーク結果

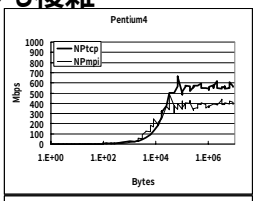
- Node内転送速度はCPU/Chipset/DIMMによって大きく異なる。

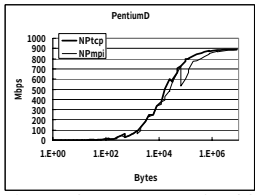


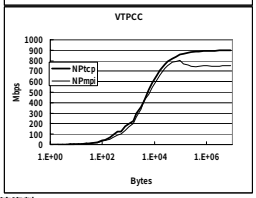


ネットワークも複雑

- Ethernet(1000BASE, GbE)性能は以下の二つに大きく依存
  - CPU性能
  - NIC chip性能
- 最高でも約900Mbps







MPIとは？

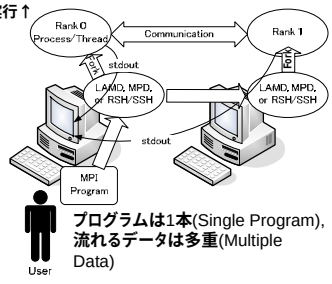
- Message-Passing Interfaceの略
- 分散メモリ環境における並列計算のための関数を規格化したもの
- Node間のデータのやり取りを全て指定する必要がある(「BSD socket関数に毛が生えた程度」)
- 代表的な実装
  - MPICH(Xeon), MPICH2(Pentium4) http://www-unix.mcs.anl.gov/mpi/mpich/
  - LAM(→OpenMPIへ統合予定)(PentiumD) http://www.lam-mpi.org/
- 一対一通信･･･同期通信と非同期通信
- 集団通信･･･一対一通信を土台に，コミュニケータ内のnode全体にデータを一括送受信

“print_process.c”

```
#include <stdio.h>
#include <math.h>
#include "mpi.h"  // MPI用のヘッダファイル

int main(int argc,char *argv[])
{
   int  namelen, numprocs, myid;
   char processor_name[2048];

   MPI_Init(&argc,&argv); // MPI初期化
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stdout,"Process %d of %d on %s¥n",
         myid, numprocs, processor_name);

   MPI_Finalize(); // MPI終了
   return 0;
}
```


SPMDアプローチ

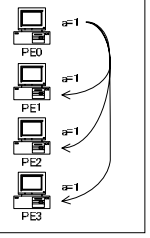
[user01@cs-room443-d01 ~]$ lamboot –v ←LAMの起動
(略)
[user01@cs-room443-d01 mpi]$ mpicc print_process.c ←コンパイル
[user01@cs-room443-d01 mpi]$ mpirun -np 8 ./print_process
“Print_process”プログラムを8PEsで実行↑
Process 0 of 8 on cs-room443-d01
Process 2 of 8 on cs-room443-d02
Process 4 of 8 on cs-room443-d03
Process 6 of 8 on cs-room443-d04
Process 7 of 8 on cs-room443-d04
Process 3 of 8 on cs-room443-d02
Process 5 of 8 on cs-room443-d03
Process 1 of 8 on cs-room443-d01
[user01@cs-room443-d01 mpi]$



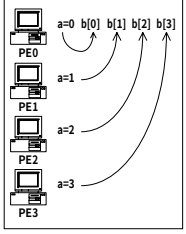
プログラムは1本(Single Program), 流れるデータは多重(Multiple Data)

集団通信・・・Bcast/Gather/Reduce

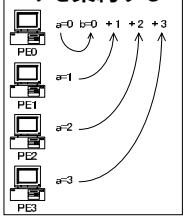
Bcast・・・同じデータを全PEに流す



Gather・・・特定のPEにデータを集約する

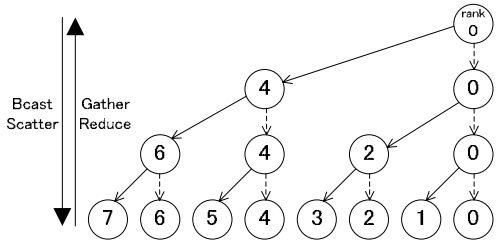


Reduce・・・特定のPEに，演算しながらデータを集約する



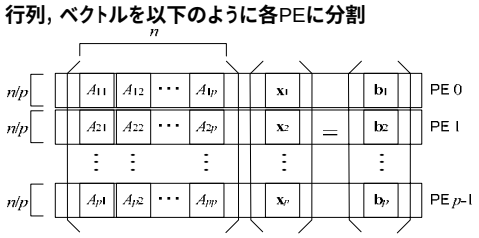
集団通信のアルゴリズム

- 通信回数はlog2(PE数)で済む



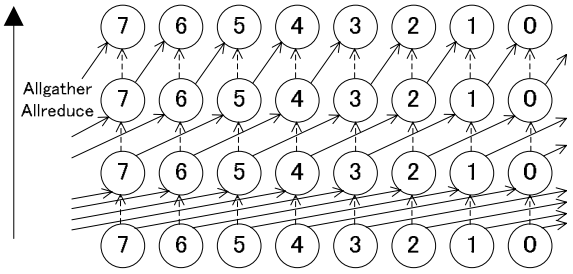
ベクトルの内積と行列･ベクトル積

- 行列，ベクトルを以下のように各PEに分割

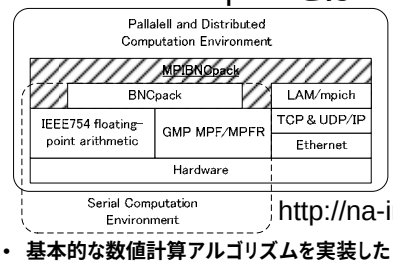


- ベクトルの内積計算の並列化→Reduce + Bcast = Allreduceを使用
- 行列･ベクトル積→Gather + Bcast = Allgatherを使用

Allgather/Allreduceの場合



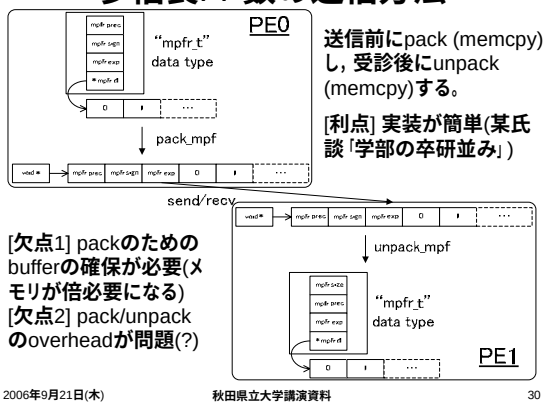
BNCpackとは？



http://na-inet.jp/na/bnc/

- 基本的な数値計算アルゴリズムを実装した自作ライブラリ
- ANSI Cで組んである(C++でも利用可能のハズ)
- IEEE754 double計算
- MPFR/GMPによる多倍長計算
- MPIによる両者の並列計算

多倍長FP数の送信方法



送信前にpack (memcpy)し，受診後にunpack (memcpy)する。

[利点] 実装が簡単(某氏談「学部の卒研並み」)

[欠点1] packのためのbufferの確保が必要(メモリが倍必要になる)

[欠点2] pack/unpackのoverheadが問題(?)

Conjugate-Gradient(CG)法とは？

- **連立一次方程式のアルゴリズム**
  - **直接法**(Linpack, Top500)･･･**LU分解**･**Gaussの消去法**
  - **反復法**･･･Jacobi**反復**, Gauss-Seidel, SOR**法**
  - Krylov**部分空間法**･･･CG, BiCG, CGS･･･**法**
- **正定値対称行列に適用可能**
- Krylov**部分空間内で解を構成する。**

CG法のアルゴリズム

連立一次方程式

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

ここで，$A^T = A \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$。

1. 初期値 $\mathbf{x}_0 \in \mathbb{R}^n$ を決める。
2. $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$，$\mathbf{p}_0 := \mathbf{r}_0$ とする。
3. $k = 0, 1, 2, \cdots$ に対して以下を計算する。
   (a) $\alpha_k := (\mathbf{r}_k, \mathbf{p}_k)/(\mathbf{p}_k, A\mathbf{p}_k)$
   (b) $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
   (c) $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A\mathbf{p}_k$(又は $:= \mathbf{b} - A\mathbf{x}_{k+1}$)
   (d) $\beta_k := \|\mathbf{r}_{k+1}\|_2^2/\|\mathbf{r}_k\|_2^2$
   (e) $\|\mathbf{r}_k\|_2/\|\mathbf{r}_0\|_2 < \varepsilon_r = 10^{-20}$ の時，停止。
   (f) $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

- **ベクトル単位の計算のみでアルゴリズムが構成されている→並列化が容易**
- **$\mathbf{p}_k$ が互いに直交する→理論的には有限回(<=n)の反復で終了する筈が･･･**

今回使用する例題

係数行列 (Frank 行列)A と解 $\mathbf{x}$ を

$$A = \begin{bmatrix} n & n-1 & \cdots & 1 \\ n-1 & n-1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ n-1 \end{bmatrix}$$

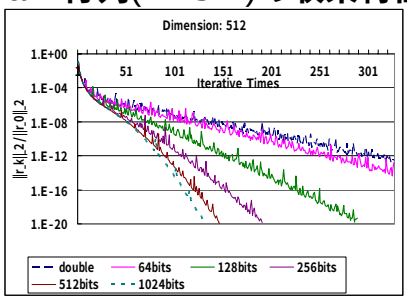
とする。但しこれだと計算時間が短くなる (整数×多倍長 FP 数) ので，

$$\sqrt{2} \cdot A$$

を係数行列として使用。

n = 512	行列	ベクトル	ベクトル/8
double	2 MB	4 KB	0.5 KB
64bits	7 (MB)	14 (KB)	0.33 (KB)
128bits	9	18	2.25
256bits	13	26	3.25
512bits	21	42	5.25
1024bits	37	74	9.25

Frank行列(n = 512)の収束特性



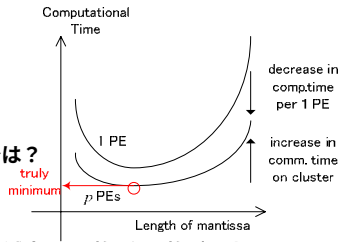
多倍長FP数の桁を増やすと反復回数が減少→単調減少に近づく

(最小反復回数(>2048bits)**：127回)**

CG法の最短計算時間は？

- **多倍長FP数の桁の増加**
  →1演算の計算時間の増加
  →CG法の反復回数は減少

並列化による計算時間減少は？



- PE**数の増加**
  →1 PEあたりの計算時間の減少 (PE数=次元数が最小)
  →通信時間の増加

計算時間 ＜ 通信時間 となれば，並列計算の効果は頭打ち

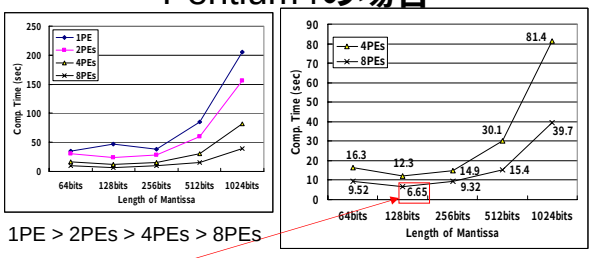
逐次計算の場合

- 1PE**で計算した時の反復回数と，計算時間**(sec)

#bits	#Iteration	Pentium4	Xeon	PentiumD
64	506〜513	34.4	16.9	22.9
128	286〜294	47.3	24.2	15.6
256	191	38.4	31.2	17.9
512	147	85.3	69.3	30.4
1024	131	205	173	60.6

- **→並列化でどこまで時間短縮が図れるのか？**

Pentium4の場合

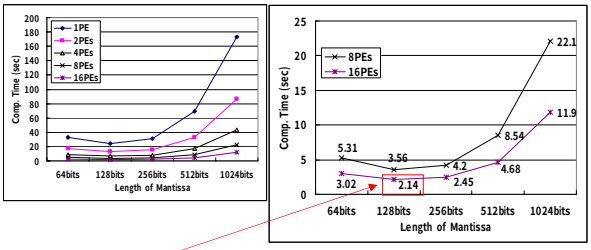


1PE > 2PEs > 4PEs > 8PEs

最小計算時間(128bits)：6.65**秒**(8PEs)

桁を倍に増やしたのに，時間は約1/5**！**

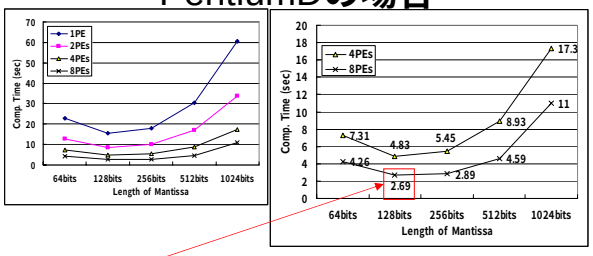
Xeonの場合



最小計算時間(128bits)：2.14**秒**(16PEs)

時間は約1/8**！**

PentiumDの場合



最小計算時間(128bits)：2.69**秒**(8PEs)

時間は約1/5**！**

本当の本当の最小計算時間は？

- PE**数の限界を超えて，最小計算時間を求めることはできないか？**
- **前提「全てのPEの計算・ネットワーク性能は同じ」**

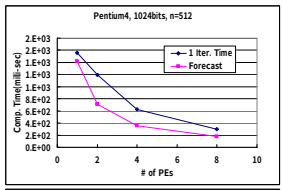
→計算時間の予測
→通信時間の予測

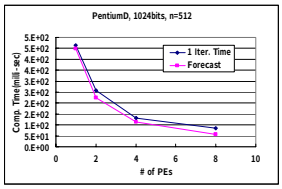
計算時間の予測は可能か？

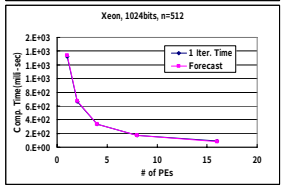
- **多倍長FP演算アルゴリズムから，計算時間のorderは分かるが，実測値との絶対的なズレは大きくなる**
- **多倍長乗法・除法・加法，1回あたりの計算時間を予め計測しておく**
- **計算回数×1回あたりの多倍長演算時間で計算時間を予測**

計算時間を予測してみると・・・

- Pentium4**のズレが大きいのは特殊要因？**（1PE**計算のズレが大きい**）
- PentiumD, Xeon**は良く合っている**







通信時間の予測は可能か？

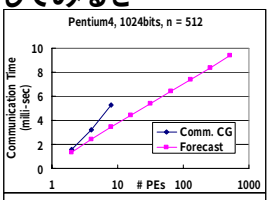
- 集団通信における一斉通信は，「各nodeから送られるデータを1回送信」と考える

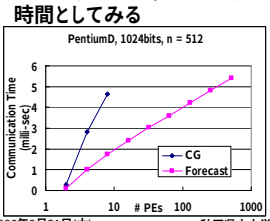
→通信回数=log2(PE数)

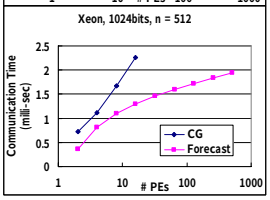
- 1PE/nodeの場合
  - NPmpiの結果を補間してThrough-put(send_receive_time, Mbps)を求める
  - 通信回数×send_receive_time　で予測
- 2PE/nodeの場合
  - 基本的には上記と同じ
  - Node内通信とNode間通信とで別々に算出して和を取る

通信時間を予測してみると・・・

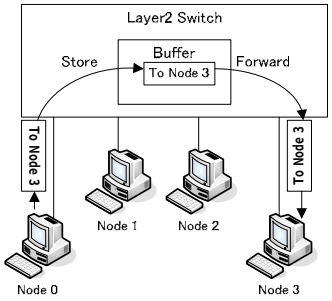
- PE数が多くなると，集団通信時間が急激に上がる→精度が落ちる
- 同じorderになっている，ぐらいの目安
- 過小評価なので，最低通信時間としてみる







L2 SwitchのStore & Forward方式



- Buffer容量
- 処理可能パケット数

が増加すると

転送速度が落ちる

- L2 Switchの性能を考慮に入れる必要あり

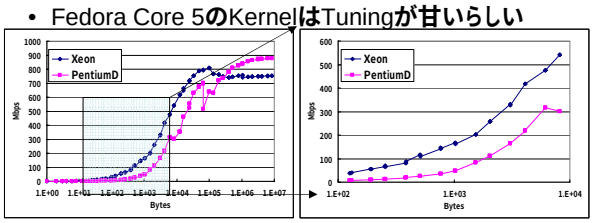
CG法の最小計算時間は？ (たぶん相当過小評価)

#bits	Min #Iter.	Pentium4	PentiumD	Xeon
64	506	6.44(milli-sec, 32PEs)×506 =3.26(sec)	3.48(32PEs)×506 =1.76	1.18(256PEs)×506 =0.60
128	286	7.2(64PEs) ×286 =2.06	3.75(64PEs) ×286 =1.07	1.31(512PEs)×286 =0.37
256	191	8.27(128PEs)×191 =1.58	4.4(128PEs)×191 =0.84	1.64(512PEs)×191 =0.31
512	147	9.97(256PEs)×147 =1.47	5.29(256PEs)×147 =0.78	2.5(512PEs)×147 =0.37
1024	131	12.2(512PEs)×131 =1.60	6.29(512PEs)×131 =0.82	4.59(512PEs)×131 =0.60

→殆ど全部，通信時間になってしまっている(実用的には意味のないレベル)

PentiumDがXeonに 実測値でも予測値でも負けている理由

- 転送データ量が小さい所で速度が遅い(実際遅い)
- Fedora Core 5のKernelはTuningが甘いらしい



→「小さいこと(データ)から，コツコツと！」

まとめ

- CG法のような，多倍長計算が有効なアルゴリズムはまだ研究する余地がある
- 実用的な多倍長計算のためには
  - 128bits〜16384bitsの精度で高速な演算
  - 100〜10000 Bytesのデータサイズで高いThroughput

  が重要
- 集団通信時間の予測は難しい (実測値を使うのが確実)
- 「πの世界記録」と「8倍精度」の間にある「実用的な多倍長計算」が必要な問題やアルゴリズムは存在する！