Windowsでささやかに PCクラスタリングしよう

高　橋　　俊

Ⅰ．Windowsで

1.1　Linuxをインストールするのは目的じゃない

・　低予算で大規模計算するなら複数のPCをつないで，PCクラスタ。
・　ネットでPCクラスタを調べると，Linuxをインストールして，C言語やFortranでMPIライブラリを使ってプログラムすると書いてある[1]。本[2]を調べてみても同じだ。
・　研究室にはPCがかなりあるが，今更Linuxをインストールするのは面倒そうだ[3]。だいたいHDDに空きがない。いや，そもそもLinuxなんかインストールしたこともない。ノートPCはパーツが特殊なものが多くて，Linuxのインストールは無理そうだ[4]。
・　それどころか，VB（Visual Basic）なら書けるけど，CやC++でプログラムを書いたこともない。
・　うーん，またの機会にしよう。

そんな方々に，PCクラスタリング業界ではあまり聞かないネタをご紹介しよう。

・　Windows版MPI
Visual C++やIntel Fortran for WindowsなどWindows開発環境でのMPIのインストール＆使い方を紹介する。プログラミング方法はネット上に情報[5]がけっこうあるし，解説本も数冊出版されているのでそちらを参照いただこう。

1　PCクラスタコンソーシアム[http://www.pccluster.org/]，同志社大学三木光範研究室「PCクラスタ超入門」[http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/index.html]など多数

2　石川裕ほか「Linuxで並列処理をしよう―SCoreで作るスーパーコンピュータ」（共立出版），2002.
トーマス・L. スターほか：「PCクラスタ構築法―Linuxによるベオウルフ・システム」（産業図書），2001など

3　10年前のLinuxといえばSlackwareだった。一発でインストールが成功することはなく，いろいろ調べないといけなかった。CD-ROMとTCP/IPを同時に使いたければ，カーネルの再構築が必要だった。あの頃ならLinuxのインストールが大変だというのは，とても正しい言い訳だった。

4　Akihito Adachi：「Linux on Note PC's」[http://www.st.rim.or.jp/~adats/WL/List/]などにあるように，たいがいのノートPCで，さまざまなLinux Distributionが正常動作している。とはいってもVAIO PCG-XR/7FのようにXFree86が非対応なグラフィックチップとPCMCIAなLANカードを使っていると，Linuxを動かすのは初心者にはとてもむつかしいものになるだろう。

5　http://ds20e.hpcc.st.keio.ac.jp/MPI_Training.pdf
http://www.hpc.co.jp/Technical/MPI/MPI-TechNote.htmなど多数ある。

・ VBで並列処理

分散処理のためにMicrosoftが作った技術 .NET Remotingを並列化のために使おうというお話。あまり世に知られていない方法なので，プログラミングやインストール方法も紹介する。

ただし，いずれも，

・ 大規模ではなく，10台程度までのPCを使う＝ありあわせのPCを利用する
・ 共同利用ではなく，一人で使うか，声の届く範囲の数人で使う＝バッチジョブ管理なんかしない
・ マシンダウンしたら，手動でマシンを再起動して計算をやりなおす

ことが前提。100台のPCを使って，多人数で共同利用しても自動的に最適な資源配分が行われ，マシンダウンした場合も自動で再起動して中断したジョブを自動的に再実行する…というような立派なものを用意しようとする準備だけでも大変。そんなことを考えると…「うーん，またの機会にしよう」になる。あるいは，つくることに意義と楽しみを見出してしまって，使うことはどうでもよくなったりする。

1.2 目標は環境構築に手間をかけないこと

というわけで，できるだけ手間をかけずに並列処理環境を整えよう。本稿では以下のようにする。

・ OSはWindows XP ProfessionalとWindows 2000 Professionalのみ[6]
  ➢ 高価なWindows Serverを買う必要もないし，同じWindowsとはいってもOS換装するのは手間だし，相性が悪ければ起動しなくなる。
  ➢ Windows98/98SE/MEは並列処理向きには作られていない[7]。
・ Software for Unix 3.5[8]やCygwin[9]などのUNIXエミュレーションソフトウェアは使わない。
  ➢ Linuxをインストールするよりは手間がかからないが，それでもそこそこ手間がかかる。
・ MPIはフリーソフトMPICH-NTを使う。
  ➢ フリーソフトではMPICH-NTとNT-MPICH[10]が，商用品ではMPI/ProとWMPIなどがあ

6 Web資料館[http://www.zerotown.com/webdata/access/os_01.htm]によれば，XPと2000の学校関係でのWindows内のシェア（2004年２月）は76.6%なので，他（98/98SE/ME）は気にする必要はないだろう。ちなみに，Macintoshのシェアは学校関係では2.5%となっており，無理にMacintoshをクラスタに参加させるほどのことはないだろう。

7 MPIはLinux/UNIXならデーモン，WindowsならWindows Serviceという形で実装されている。ところが，Windows95系（98/98SE/ME）はこのWindows Serviceという仕掛けを持っていない。

8 MicrosoftのUNIXエミュレーション環境で，Version 2.2は無償で，Version 3で有償化され，Version 3.5で無償化された。Gnuのツールがそろっていて，米国の入札基準ではUNIXとしての要件を満たしているらしい。というより，政府機関の入札にWindowsで対応するために，Software for Unixの前身であるInterixを作ったらしい。

9 かの有名なフリーのUNIXエミュレーション環境。[http://www.cygwin.com/] gccなどもいっしょにインストールできる。

10 http://www.lfbs.rwth-aachen.de/users/karsten/projects/nt-mpich/

る。MPI/ProはWindowsドメインを組まないといけないので手間。

- ➢ NT-MPICHではなくMPICH-NTを使うのは，たまたま先に目に止まったからで，特段の理由はない。

MPIを使う場合の開発言語はVisual C++ .NET 2003とIntel Fortran for Windows 8.0

- ➢ 既存のMPIなプログラムはほとんどCかFortranで書かれている。ここで新奇なものを使う必要はないだろう。
- ➢ gccやMicrosoft .NET framework SDK[11]など無償ソフトウェアでもよいが，IDE（統合開発環境）がないので使い勝手がよくない。

・ VBで並列処理するときはVisual Basic.NET 2003のみ。

- ➢ 特にIDEのインテリセンス[12]が効くのがVBなので，Microsoft .NET framework SDKで間に合わせるのは，あまりに非効率。

表１　すでにインストールされているものとここでインストールするもの

すでにインストールされているもの	Windows XP Professional SP1 Visual Studio .NET 2003（もしくはVisual C++ .NET 2003）
ここでインストールするもの	MPICH for Microsoft Windows 1.2.5 [MPICH-NT] Intel Fortran for Windows 8.0 [30日間試用版]

これらのうちVisual Studio .NETはインストールされているものとして，それ以外のインストールについて説明していこう。

また，実験材料には私の手持ちの３台のノートPCを使った。ありあわせなので，性能は一定ではない。

11　VC++, VB, C#のコンパイラ・リンカ・ライブラリは無償で提供されている。でも，ヘルプもなければ，IDEもないのであまりお奨めではない。インテリセンスとデバッガのご利益を考えれば，そこまでケチることもないだろう。

12　インテリセンスとはコードの入力中につぎに入力するコードの候補（クラス名）が自動的にリスト表示し，候補の中から適切なコードを選べる機能である。これによりプログラマがクラス名を覚える必要がなくなり，スペルミスもなくなる。

さらにVisual Studio .NET 2003では過去に使用したことのあるコードが自動選択されて表示される学習機能が追加された。この機能が最もよく効くのはVBでつぎがC#だ。やはり，Bill Gates最初のプロダクトであるBasicは今でも最優先開発言語なのかもしれない。（とMicrosoft社内でも言われているようだ）

表２　インストールするために使った実験材料

マシン名	Foundation	Trantor	Atlantic
Windows	XP Professional SP1	XP Professional SP1	XP Professional SP1
CPU	Pentium M 1.5GHz	Pentium III 850MHz	Pentium III 800MHz
RAM	１GB（DDR）	256MB	320MB
LAN	標準装備	標準装備	Buffalo LPC-3CLX[13]
（100BaseTX）	BUFFALO　WHR3-AG54 [IEEE802.11a,b,g対応 無線ブロードバンドルーター]		
Hardware	VAIO PCG-Z1V/P	VAIO PCG-SR7F/PB	VAIO PCG-XR7F/K[14]
開発言語	Visual C++ .NET 2003 Intel Fortran 8.0 Visual Basic .NET 2003		

Foundation, Trantor, Atlanticは互いに相手のマシンが“見える”ようになっていることは前提。同じワークグループかドメインに所属させておくのがよいだろう。

Ⅱ．MPICH on Windows

２.１　何はともあれMPICHをインストール

（１）MPICH.NTを入手

下記から自己解凍＋インストーラである“mpich.nt.1.2.5.exe”をダウンロードする。2004年５月時点の最新版は1.2.5である。

http://www-unix.mcs.anl.gov/~ashton/mpich.nt/

（２）MPICH.NTをすべてのマシンにインストールする

自己解凍＋インストーラのアイコンをダブルクリックすれば，あとは[SETUP][NEXT][YES]を押し続ければ，特に何事もなくインストールは完了する。私の実験環境では，同じことをFoundation, Trantor, Atlanticで行った。

13　Windows XPは自動認識する。Linux用のドライバはメルコのWebに載っていたが，今では見当たらない。メルコはLinuxへの関心を失ってしまったのだろうか。

14　このモデルのプリインストールOSはWindows 2000 Professionalである。グラフィックチップがS3 Savage/IX8で，当時のXFree86は非対応だったので，LinuxでX-windowを使うにはこのチップ専用のドライバだった。

図１　MPICH.NTのインストール過程（悩む場面はまったくない）

（３）初期設定

インストールしただけでは使えず，初期設定が必要。こっちはインストールと違って，まとめて設定することも可能。まずは，下記のようにメニューをたどって，mpich configuration toolを起動。

[スタート]→[すべてのプログラム]→[MPICH]→[mpd]→[mpich configuration tool]

１）Select the hosts to configure

まとめて設定するなら，ここでクラスタに参加するマシン全部を指定しよう。

２）Enter the password to connect to remote mpd’s

これはmpd（MPICHのWindowsサービス）に接続するためのパスワードで，実行時にユーザが入力することはない。勝手に，mpd間の会話で使われるだけ。

３）ApplyもしくはApply Single

全マシン一気に設定するなら[Apply]で，１台だけなら[Apply Single]

これで準備は完了。

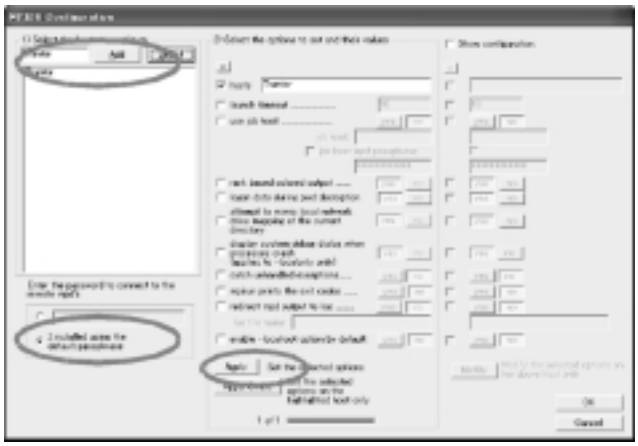


図２　mpich configuration toolの画面

２.２　Visual C++ .NET2003でMPICH.NTのサンプルをテスト

これでMPICHが使えるようになっているはずなので，試してみよう。とりあえずはMPICH.NTに付いているサンプルをビルドして動かしてみよう。なお，サンプルは以下のフォルダにある。

[mpich.ntをインストールしたフォルダ]\SDK\Examples\nt

（１）開始<マシン=Foundation>

[example.dsw]というファイルをダブルクリックすると，Visual Studio .NET 2003が起動する。サンプルはVisual Studio 6.0用に作られたものなので，プロジェクトを変換するか聞いてくる。[はい]を選べば，Visual Studio .NET 2003用のプロジェクトに変換される。特に気にする必要もなく，エラーも警告も出てくるわけでもなく，すぐに変換は終わる。

図３　Visual Studio ソリューションを開く

（2）ビルドする前にちょっと修正<マシン=Foundation>

6つあるサンプルのうちmandelはリンクエラーになる。これはライブラリのリンク順序の問題であり[15]，ライブラリの順序を変えれば解決する。

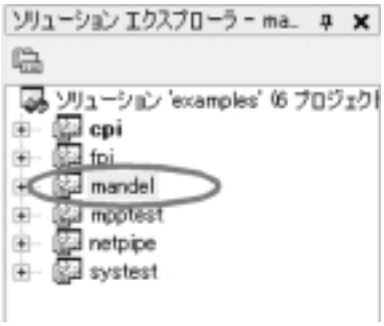


図4　MPICH.NTのサンプル（mandelはリンクエラーになる）

修正方法はつぎのようにプロジェクトのプロパティを選んで

ソリューションエクスプローラ→mandelで右クリック→コンテキストメニュー[プロパティ]選択

プロパティページを以下のようにたどって

[構成プロパティ]→[リンカ]→[入力]

図5　ライブラリのリンク順序の変更

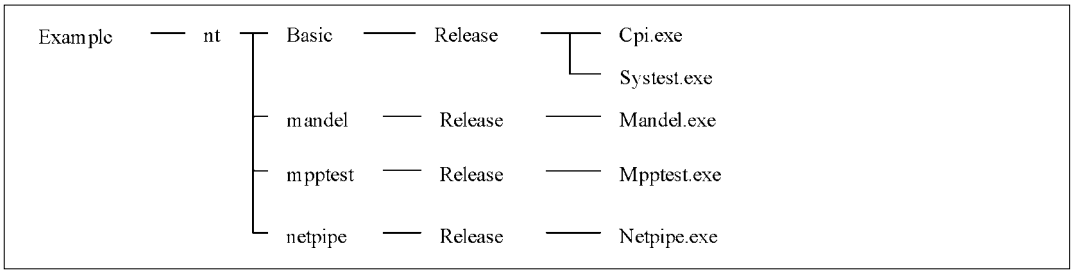
そこで，つぎの2つを修正して，libcmtlib→nafxcwの順序を逆順にする。

[追加の依存ライブラリ]に追記	nafxcw.lib;libcmt.lib
[特定のライブラリの無視]に追記	nafxcw.lib;libcmt.lib

15　http://www.kbalertz.com/Feedback_148652.aspx

（3）ビルド<マシン=Foundation>

ライブラリのリンク順序を変えてしまえば，ビルドは何の問題もなく完了し，以下のフォルダに実行形式が作られる。



（4）配布＜マシン＝Foundation, Trantor, Atlantic＞

並列実行するには，これらの実行形式（exe）を何らかの形で，FoundationからTrantorとAtlanticにコピーする必要がある。手っ取り早いのは，フォルダ[nt]をFoundationで共有設定（読み取り）して，Foundation, Trantor, Atlanticで同じドライブ名でネットドライブにマウントするのがよいだろう。もちろん，同じ位置（ドライブ名とフォルダの位置や名前）にコピーしてもかまわない。

（5）実行＜マシン=Foundation＞

これらの並列プログラムを実行するには以下のようにたどって，mpirunを実行する。

[スタート]→[すべてのプログラム]→[MPICH]→[mpd]→[mpirun]

[Application]で，実行形式（exe）をフルパスで指定する。

[Number of processors]で，何台のPCで実行するか指定する。

右端のウィンドウで，並列実行に参加するマシンを選ぶ。この例では FoundationとAtlanticで実行する。

[Run]で並列実行が始まる。

図６　並列実行開始!!

実行結果（標準出力）は左のウィンドウに示される。

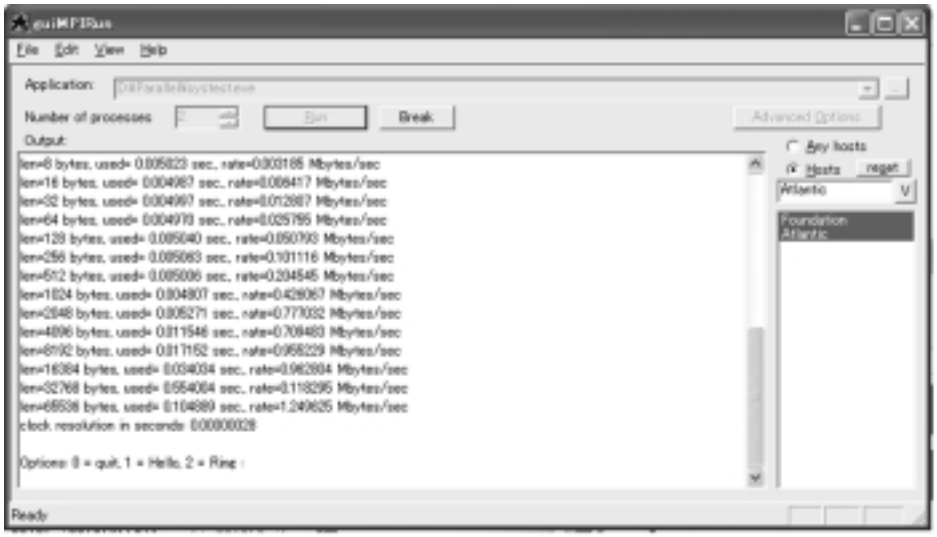


図７　並列実行終了（結果は左のウィンドウに）

２.３　Visual C++ .NET 2003で，いちからプログラムをつくるときは

サンプルは動いたけれど，サンプルはVisual Studio.NETのプロジェクトができ上がっているので，いちから作る場合についての例にはならない。

（１）新しいプロジェクト

[Visual C++プロジェクト→Win32コンソールプロジェクト]が，MPICHを使ったCプログラムを作るときのテンプレートである。Win32コンソールプロジェクトを選んで[OK]すると，ウィザードが立ち上がる。特に設定することもないので[完了]させる。

図８　新しいプロジェクト

（２）プロジェクトの設定

まずは，[C++→プリコンパイル済みヘッダー→プリコンパイル済みヘッダーの作成／使用]で，[自動的に作成する]に変更しよう。

図９　プリコンパイル済みヘッダーの設定

続いて，[C++→全般→追加のインクルードディレクトリ]で，[(MPICHをインストールしたフォルダ)\SDK\include]を指定する。サンプルの場合は相対パスで設定されているが，ここではもちろん絶対パス。

図10　インクルードディレクトリの指定

[リンカ→入力→追加の依存ファイル]でライブラリを追加する。追加項目はmpich.lib以外にws2_32.lib, odbc32.lib odbccp32.libである。全部必要かどうかはプログラムによるが，サンプルではそうなっていたので，同じように指定するのが安全だろう。

図11　ライブラリの指定

[リンカ→全般→追加ライブラリディレクトリ]でライブラリのあるフォルダ[(MPICHをインストールしたフォルダ)\SDK\Lib]を追加する。

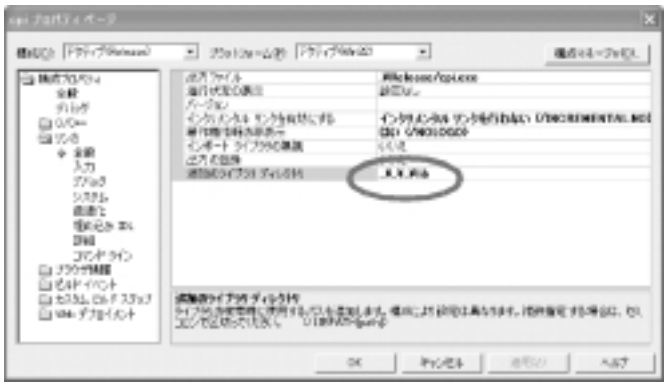


図12　ライブラリ探索パスの追加

2.4　Intel Fortran for WindowsでMPIなアプリQCDMPIを動かしてみる

サンプルだけでは面白くないので，手ごろなアプリを動かしておこう。例題にはソースコードの短く，並列化がシンプルなQCDMPI[16]を使おう。

http://insam.sci.hiroshima-u.ac.jp/QCDMPI/index.html

（1）その前にIntel Fortran 8.0 for Windowsをインストール

QCDMPIはFortranで書かれているので，Fortranコンパイラが必要[17]となる。今回はお試しだけなので，30日間試用版を使おう。以下のURLで順に，[Survey][License Agreement][Registration]と進めると，ライセンスファイルを添付したダウンロード先が本文に書かれたメールが送られてくる。

http://www.intel.com/software/products/compilers/fwin/eval.htm

添付されたライセンスファイルを[C:\Program Files\Common Files\Intel\Licenses]フォルダの下においておく。実際のインストールは，[NEXT][YES]の連続で済む。インストールの終わりあたりで，ライセンスファイルの位置を聞いてくるので，さきほどのフォルダにあるライセンスファイルを指定すれば完了。

16　日置慎治 . QCDMPI.　[http://insam.sci.hiroshima-u.ac.jp/QCDMPI/]

17　無償のf2c（Windows版）を使ってC言語に変換する手もある。ただ，エラーが出たときに，f2cが悪いのか，MPICH.NTが悪いのか，はたまたVisual C++が悪いかわからなくなる。ここでは手堅くIntel Fortranを使うことにする。

なお，Intel Fortranは英語版しか存在しない。これを日本語版のVisual Studio.NET2003に統合すると，Intel Fortran独自部分は英語で，それ以外は日本語でメッセージが表示されることになる[18]。

（2）とりあえず，ビルドの準備

makefileを使う方法もあるが，ここはWindowsの普通のやり方でいこう。ということでVisual Studio .NETを起動して，[Intel　Fortran Projects→Console Application]で新しいプロジェクトを作る。ウィザードが立ち上がってくるが，特に設定することもないので[Finish]させる。

図13　新しいプロジェクト

（3）FortranとLinkerの設定

これはVisual C++.NETと同じである。Fortran用のMPICHライブラリがあるわけでもないので，書き込む文字列も同じである。

18　日本語環境（日本語Windowsと日本語Visual Studio.NET）での動作はIntel日本法人が動作テストしているので問題はない。ただし，技術者の数に余裕がないので，とてもメッセージの日本語化までは手が回らなかったようだ。

図14　ライブラリ探索パスとライブラリの追加

プロジェクトのプロパティの[Fortran→Preprocessor]のところで，[Preprocess Source File]をYESにしておくこと。でないと「#ifdef」を処理してくれない。

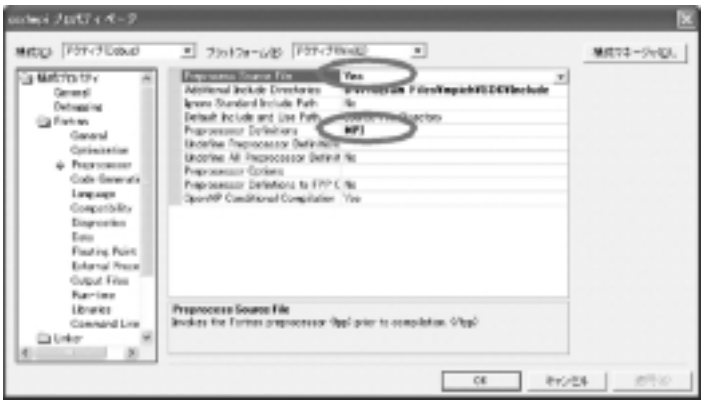


図15　インクルードフォルダの追加とプリプロセッサの設定

（4）QCDMPIのソースをプロジェクトに追加

[プロジェクト→既存項目の追加]で，QCDMPIのソースファイル（.F/.f）を読み込む。インクルードファイル「param」は明示的に読み込む必要はない。

ついでに，ここで，プロジェクトを作ったときにできてしまった[プロジェクト名.f90]というファイルを削除しておこう。残しておくと「MAIN」が２個になってしまうので，リンクエラーになる。

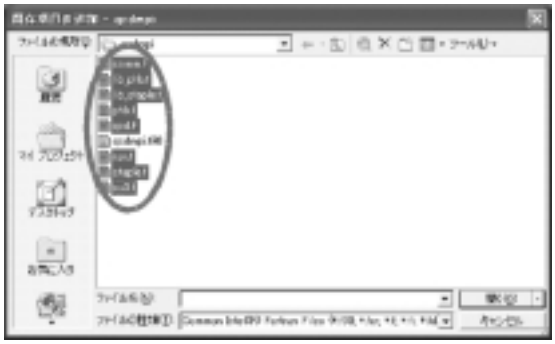


図16　既存項目の追加でソースファイルを読み込み

（5）実行

実行はMPICH.NTのサンプルプログラムと同じで，mpirunから実行する。指定すべきパラメータ類も同じである。

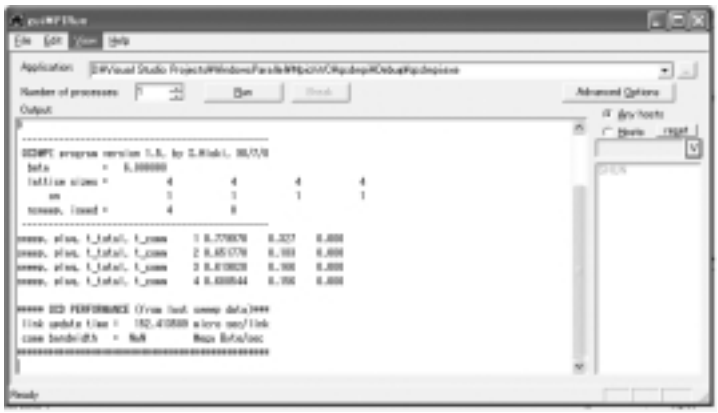


図17　QCDMPIをmpirunから実行[19]

2.5　Windowsで動くフリーなMPIアプリケーション

世の中にはWindows版MPIアプリケーション（ビルド済み）も存在している。とりあえず，Windowsでの並列動作を楽しみたい場合は，試してみるのもよいだろう。

（1）LANLの相同性検索プログラムのMPI並列版であるmpiBLAST

http://mpiblast.lanl.gov/index.html

19　あいにくと私は流体屋＆金融工学屋で，QCDを知らないので，結果が正しいのかどうかは定かでない。

（2）PEACH

産総研の古明地勇人氏の生体分子動力学プログラムPEACH[20]を群馬大学工学部の中田吉郎教授のグループがWindowsに移植し，MPIで動作させている[21]。

Ⅲ． NET Remoting

3.1 MPIだけが並列化じゃない

並列化といえば，1980年代末の「OSは各社固有のUNIXで，開発言語はFortranかCで，各社固有の関数で通信を書く」，そして1990年代半ばからの「OSはUnixかLinuxで，開発言語はFortranかCで，MPIで通信を書く」というメインストリームがそのまま2004年にいたるも変わっていない。ころころと流行が変わり続けるITの世界にあって，15年にわたりその姿を変えていない「並列化」。

もちろん変わらない理由はちゃんとある。いかなる科学技術計算プログラムも並列記述可能にするMPIと，性能を追求したFortran[22]という組み合わせにまさるものがないことは，今も否定しようがない。余分な処理を徹底的に排除しつくしたチューニング済みのFortranコードは冗長なオブジェクト指向コードの数倍の性能を発揮する。性能チューニングのプロフェッショナルたち[23]にプログラムをゆだねれば，４倍程度の高速化が実現することもしばしばだ。

しかし，世の中はITベンダの商売の都合で変えられる流行はさておいても[24]，過去15年で技術が変わったことは違いない。

- Webアプリケーションなどリモートマシンに仕事をさせることが多くなってきた。10台のWebサーバに同時に仕事をリクエストしたら，これは並列処理？
- VB/C#やJavaでアプリを作ることが多くなってきた[25]。過去の資産はさておき，新たに並列プログラムをつくるときも従来どおりCやFortranで書くのだろうか？
- ビジネス系の分散処理のための仕掛けJava-RMIやMicrosoftのDCOM（あるいはその後継であ

---

20 http://staff.aist.go.jp/y-komeiji/md/md.html [2004年６月にPEACH 5をリリース予定]

21 ダウンロードページなどはないようで，入手するには直接メールする必要がありそうだ。[http://www.sccj.net/publications/JCCJ/v2n4/a15/document.pdf]

22 コンパイラの最適化が最も効きやすいのは参照関係が明確なFortran77で書かれたプログラムである。高速なCPUに比べてあまりに低速なメモリ，多数の演算器を同時に動かすことで性能を上げようとするCPUというアーキテクチャにおいては，コンパイラの最適化次第で性能は数倍違ってくる。

23 日立・富士通・日本電気には自社スーパーコンを売るためにお客様プログラムをチューニングする専門部署がある。その部署でもキーとなるエンジニアの名前は互いにわかっている。こいつらを引き抜けば最強チームが作れる名簿ができているという営業部長がいたりもする。

24 CORBAはどこに行ったのだろうか？ オブジェクト指向のつぎはエージェント指向だという話はいつのことだ？ オープンソースの前にオープンシステムってあったような気がするが何のことだったのだろう？ ちょっと前まで，どんなソフトウェアのバージョン番号にも“i”とついていたが，いまでは何でもかんでも“g”だ。今では並列データベースも，PCクラスタも，SETI@homeのような分散コンピューティングも，みんな“g”だ。

25 ポインタ操作による外道なコーディングなどを許さないのが好まれて，プログラミング教育の対象言語も今ではJavaになっているようだ。Javaで大規模シミュレーションとかJava-MPIとか，AppletでDukeくんがおどっていた頃からは考えられないようなことになっている。

る.NET Remoting）などが使われるようになってきた。

特に大きいのは，ExcelとVBAで計算をするようになってきたことだろう[26]。古くからの科学技術計算屋から見れば性能度外視としか言えないシロモノだが，可視化などの後処理が簡単だったりするので結構使われている。さすがにExcelとVBAを並列処理する現実的な方法はないが，VBの並列処理なら可能だ。

ここでは，つぎのようなパターンで並列処理を実現してみることにしよう。

表３　VBによる並列化の材料

項　　目	内　　容	理　　由
OS	Windows XP	まわりのPCはだいたいXPか2000
開発言語	Visual Basic .NET 2003	.NET Remotingを使う インストーラを作る
リモートマシンへのソフト配布	インストールする	プログラムの修正と実行を繰り返す場合ではなく，固まったものでパラメータサーベイするなどを想定
待ちうけ方法	Windowsサービスを常駐	ISなどのWebサーバでは重過ぎる
通信の記述	.NET Remoting	バイナリで通信できる DCOMよりも簡単に書ける

リモートマシンに仕事をさせて，結果を回収するという処理の基本は何をやっても同じなので，MPICHと大きな違いがあるわけではない。

表４　並列処理の比較

	.NET Remoting	MPICH.NT（Windows）	MPICH（Linux）
配布方法	各マシンにインストール	共有フォルダもしくはコピー	NFSもしくはコピー
待ちうけ	リモートのアプリがWindowsサービスとして常駐	MpdがWindowsサービスとして常駐	Mpdがデーモンとして常駐
通信	.NET Remotingのメソッド	MPI関数	MPI関数

26　佐藤＆佐藤：「Excel VBAによる化学プログラミング」（培風館），2002

救急？『お天気』診察室・編集局：[http://www.geocities.co.jp/Technopolis/7126/excel_fluid.htm]（これはやりすぎか）

もともとの目的が分散処理である「.NET Remoting」なので，MPIのような柔軟かつ機動的な通信はできない。したがって，並列処理の流れも**図18**に示すように違っている。

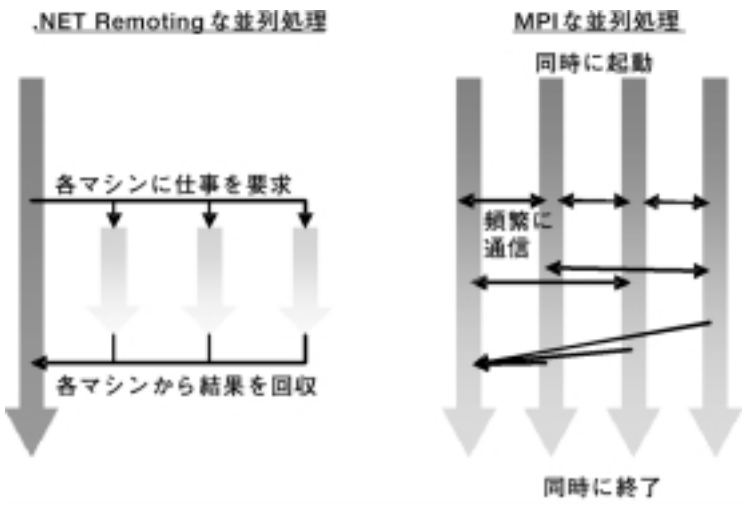


図18　.NET RemotingとMPIの並列処理の流れの違い

.NET Remotingは，並列処理すべきところまでプログラムの実行が進んだら，リモートマシンの仕事を要求する。リモートマシンとの会話は，要求した仕事の結果を回収する時点までない。感覚的にはわかりやすい並列処理の流れだが，連続体の計算にはなじまない。

一方，MPIは例え並列処理すべきものが何もなくても，各マシンで同じ処理を実行する形を取る。処理の途中で何度も通信できるので，多くの科学技術計算プログラムの並列記述が可能だが，感覚的にはわかりにくいかもしれない。

参考書としては以下のものがある。

[1]　Ingo Rammer: “Advanced .Net Remoting in Vb .Net (.Net Developer)”, Springer-Verlag, 2002.

[2]　Curan, Olsen, Pinnock: “Visual Basic .NET Remoting Handbook”, Wrox, 2002.

[3]　Corway et al.: “Visual Basic .NET Windows Service Handbook”, Wrox, 2002.

[4]　鄭立：「これからはじめる.NET Framework .NETリモーティング」，秀和システム, 2003.

[1]は日本マイクロソフトの.NET Remotingに詳しいコンサルタントもお奨めで，定評がある。[2][3]は[1]ほど詳しくないが，読みやすくわかりやすい。ただし，残念ながら[2]は2004年６月時点では品切れ。[4]は唯一の日本語の文献だがあまり役に立たないようだ。

３.２　Remotingプログラムをつくる

（１）全体構成

クライアント側のWindowsアプリケーションとリモート側のWindowsサービスを作る。こ

れを**表５**に示すように４個のプロジェクト（EXE/DLL）として作ることにしよう。

これらのうち，クライアントにはRemotingWindows.exe，RemotingMaster.dll, RemotingWorker.dll を，リモート側にはRemotingServer.exe, RemotingWorker.dllをインストールすることになる。

表５　サンプルプログラムのexe/dllとクラス

プロジェクト（Exe/dll）	クラス	役　　割
RemotingWindows.exe	Form1	GUI
RemotingMaster.dll	ClientEnvironments	ポート番号やマシン名をにぎるだけ
	RemotingMaster	クライアント側の処理クラス
	RemotingClientSC	クライアント側の並列実行管理クラス
RemotingServer.exe	Servicer	リモート側の受け口
RemotingWorker.dll	Class4SingleCall	リモート側の処理クラス
	RemoteEnvironments	ポート番号をにぎっているだけ
	A入力SC, A出力SC	引数クラス

（２）プロジェクトの作成

[ファイル→新規作成→空のソリューション]で複数のプロジェクトの入れ物をつくる。

続いて

[ファイル→プロジェクトの追加→新規プロジェクト]

で順に**表６**にある４つのプロジェクトを作る。ただし，RemotingServerはちょっと特殊な手順が必要なので（３）で別途説明する。

表６　ここで作るプロジェクト

プロジェクト（Exe/dll）	VBプロジェクトの種別	備　　考
RemotingWindows.exe	Windowsアプリケーション	
RemotingMaster.dll	クラスライブラリ	
RemotingServer.exe	Windowsサービス	サービスインストーラを追加しておく
RemotingWorker.dll	クラスライブラリ	

（３）Windowsサービスのプロジェクト(RemotingServer)を作る

[ファイル→プロジェクトの追加→新規プロジェクト]で

[Visual Basicプロジェクト→Windowsサービス]を選んで作成。

図19　Windowsサービスのプロジェクトをつくる

作成したら，Service1.vbというファイルが作られているはずなので，そこにツールボックスにあるタイマをドラッグ＆ドロップする。Windowsサービスは仕事がないと実行終了して姿を消してしまうので，タイマで姿が消えないように貼り付けてしまおうという裏技。

図20　何もしないタイマでWindowsサービスを貼り付ける

続いて
[プロジェクト→新規項目の追加→インストーラクラス]でインストーラを追加する。
Installer1が追加される。ここでは特に深い意味はないが，ProjectInstallerという名前に変更する。ここに[ServiceInstaller]と[ServiceProcessInstaller]をドラッグ＆ドロップする。

図21　インストーラクラスを追加

おそらく，ツールボックスに[ServiceInstaller]と[ServiceProcessInstaller]が見当たらないと思うので，
[ツール→ツールボックスのカスタマイズ→.NET Frameworkコンポーネント]でこの２つを追加する。そうすれば，ツールボックスに２つが追加されるので，ProjectInstallerに貼り付けてしまおう。

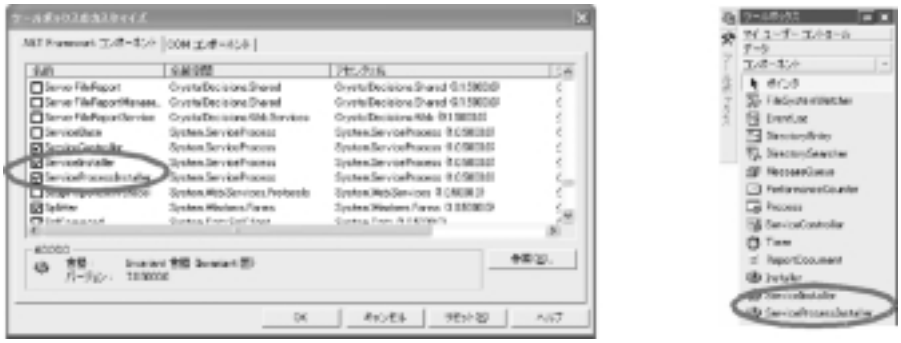


図22　インストーラクラスにサービスインストーラを追加する

図23　追加されたインストーラ

ここまでは儀式と思って，毎度同じことをするだけ。

（4）参照の追加

Visual Basic.NETでは必要な参照関係は自動的に設定してくれる。ただ，全部に気をまわしてくれるわけではない。ここでは“System.Runtime.Remotin”への参照を明示的に追加する必要がある。あと自分で作ったプロジェクト間の参照関係も明示的に追加する。

表７　追加する参照関係

プロジェクト（Exe/dll）	追加する参照
RemotingWindows.exe	*System.Runtime.Remoting*, RemotingMaster
RemotingMaster.dll	*System.Runtime.Remoting*, RemotingWorker
RemotingServer.exe	*System.Runtime.Remoting*, RemotingWorker
RemotingWorker.dll	特になし

（5）リモートの処理の本体

並列処理で呼び出される計算の本体をまず作ろう。といっても，サンプルなので素晴らしい計算をしても時間がかかるだけなので，以下のように積和演算１回だけ。

```
Public Class Class4SingleCall
Inherits MarshalByRefObject
    Public Sub New()
    End Sub

    Public Function OneProc(ByVal The入力 As A入力SC) As A出力SC
        Dim The出力 As New A出力SC
        The出力.answer = The入力.index * 2 + 1000
        Return The出力
    End Function
End Class
```

引数はあとで拡張しやすいようにクラスにしておく。

といっても，今は積和演算１個だけなので，以下のように変数１個だけのクラス。

```
<Serializable()> Public Class A入力SC
    Public index As Integer
End Class
<Serializable()> Public Class A出力SC
    Public answer As Double
End Class
```

（６）Windowsサービスの待ち受け口（Servicer）

誰が書いても同じで，ほとんどいつでもこう書く。

```
Imports System.ServiceProcess
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp‘参照の追加".NET:  System.Runtime.Remoting
Imports System.IO
Imports System.Runtime.Serialization.Formatters
Public Class Servicer
    Inherits System.ServiceProcess.ServiceBase
```

```
    “コンポーネント デザイナで生成されたコード”
    Protected Overrides Sub OnStart(ByVal args() As String)
        Timer1.ToString()    ‘決して割り込まないタイマ割り込みをセット
                             ‘これでWindowsサービスはサーバに張り付く
        Dim provider As New BinaryServerFormatterSinkProvider
        provider.TypeFilterLevel = TypeFilterLevel.Full    ‘必ずこう書く[27]
        Dim props As IDictionary = New Hashtable
        props("port") = RemoteEnvironments.ポート番号
        Dim chan As New TcpChannel(props, Nothing, provider)
        ChannelServices.RegisterChannel(chan)    ‘チャネルの設定・必ずこう書く

        RemotingConfiguration.RegisterWellKnownServiceType ( _
            GetType(Class4SingleCall), _
            "CalcSC", WellKnownObjectMode.SingleCall )‘クラス名をGettypeに書き
                ‘クライアントが呼び出すときの名前を“CalcSC”とする
    End Sub
    Protected Overrides Sub OnStop()
    End Sub
End Class
```

重要な点は４つ。

- Timer1.ToString()で絶対に発生しないタイマ割り込みをセット
- 通信するときのポート番号を設定（コンフィギュレーションファイルで設定する方法もある）
- 呼び出されるクラス名をセット
- 呼び出されるときの名前をセット（WebのURLみたいなもの）

もし複数のメソッドを呼び出したいのであれば，RemotingConfiguration. Register WellKnownSericeTypeを対応する個数分だけ書き足せばよい。

もののついでのポート番号をにぎっているだけのクラス（環境変数全部を管理するクラスとして，いずれは役に立つか？）

```
Public Class RemoteEnvironments
    Public Const ポート番号As Integer = 78215    ‘適当なポート番号
End Class
```

27　この記述は，Visual Basic.NET 2003用である。セキュリティ関連の変更により，Visual Basic.NET 2002とは違うものになっている。Web上の古いドキュメントや書籍にはVisual Basic.NET 2002用の記述が掲載されているかもしれないので，注意が必要。

（７）Windowsサービスを呼び出すクライアント側（RemotingClientSC）

```
Imports System.Threading
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp
Public Class RemotingClientSC
    Public Structure AStatus
        Dim error_message As String
        Dim error_flag As Boolean
    End Structure
    Protected TheStatus() As AStatus

    Public Function Exec(ByVal NumOfProc As Integer, ByVal The入力() As A入力SC) As A出力SC()
        Dim objs(NumOfProc - 1) As Class4SingleCall
        Dim objDelegates(NumOfProc - 1) As CalcSC_Delegate
        Dim results(NumOfProc - 1) As IAsyncResult
        Dim yets(NumOfProc - 1) As Boolean
        Dim iproc As Integer, i As Integer

        Dim machines(NumOfProc - 1) As String
        For i = 0 To NumOfProc - 1
            Dim iref As Integer = i Mod ClientEnvironments.マシン数
            machines(i) = ClientEnvironments.refMachines(iref)
        Next

        Dim The出力(NumOfProc - 1) As A出力SC
        ReDim TheStatus(NumOfProc - 1)

        For iproc = 0 To NumOfProc - 1
            Try
                Dim aConn As String = "tcp://" + machines(iproc) + ":" _
                    + RemoteEnvironments.ポート番号.ToString + "/CalcSC"
            objs(iproc) = CType(Activator.GetObject(GetType(Class4SingleCall), aConn), Class4SingleCall)
                objDelegates(iproc) = New CalcSC_Delegate(AddressOf objs(iproc).OneProc)
                yets(iproc) = True
                results(iproc) = objDelegates(iproc).BeginInvoke(The入力(iproc), Nothing, Nothing)

                    TheStatus(iproc).error_flag = True
                Catch e As Exception
                    TheStatus(iproc).error_flag = False
                    TheStatus(iproc).error_message = e.ToString
                End Try
            Next
```

```
        Dim multiAlready As Integer = 0
        While multiAlready < NumOfProc
            For iproc = 0 To NumOfProc - 1
                Try
                    If yets(iproc) And results(iproc).IsCompleted = True Then
                        The出力(iproc) =
objDelegates(iproc).EndInvoke(results(iproc))
                        yets(iproc) = False
                        multiAlready += 1
                    End If
                Catch e As Exception
                    yets(iproc) = False
                    multiAlready += 1
                    TheStatus(iproc).error_flag = False
                    TheStatus(iproc).error_message = e.ToString
                    Debug.WriteLine("EndInvoke=" + e.ToString + ControlChars.CrLf)
                    Debug.WriteLine("AddMsg=" + e.Message + ControlChars.CrLf)
                End Try
            Next
            Thread.CurrentThread.Sleep(1000)
        End While

        Return The出力
    End Function
    Delegate Function CalcSC_Delegate(ByVal The入力 As A入力SC) As A出力SC
End Class
```

（８）呼び出し前の処理と呼出し後の処理（RemotingMaster）

Newメソッドではチャネルの準備をする。

TestSCメソッドでリモートを呼び出すための引数の準備と，RemotingClientSCの呼び出しと，計算結果の集計を行う。

```
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp
Public Class RemotingMaster
    Public Sub New()
        Try
            Dim chan As TcpChannel
            chan = New TcpChannel()
            ChannelServices.RegisterChannel(chan)    ‘ チャネルの設定，いつでもこう書く
        Catch e As Exception
        End Try
    End Sub
    Public Function TestSC(ByVal NumOfProc As Integer) As String
        Dim iproc As Integer
        Dim SCobj As New RemotingClientSC()

        Dim The入力(NumOfProc - 1) As A入力SC ‘ 引数の準備
        For iproc = 0 To NumOfProc - 1
            The入力(iproc) = New A入力SC()
            The入力(iproc).index = iproc
        Next

        Dim The出力() As A出力SC ‘ 呼び出し
        The出力 = SCobj.Exec(NumOfProc, The入力)
        Dim TheAnswer As String

        For iproc = 0 To NumOfProc - 1 ‘ 結果の集計
            TheAnswer += The出力(iproc).answer.ToString()
            TheAnswer += ControlChars.CrLf
        Next
        Return TheAnswer
    End Function
End Class
```

（9）GUI（RemotingWindows）

アプリの入り口は特に何もいらないが，ボタンと戻り値を書く欄（RichTextBox）くらいはつけておこう。

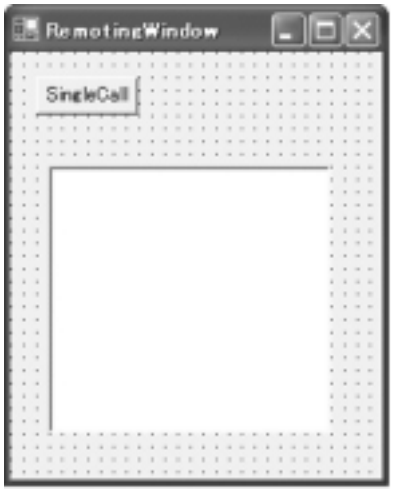


図24　Form1のデザイン

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    ‘ "Windows フォーム デザイナで生成されたコード"
    #Region " Windows フォーム デザイナで生成されたコード "

        Dim TheMaster As New RemotingMaster()
        Dim NumOfProc As Integer = 1

        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    Button1.Click
            NumOfProc = ClientEnvironments.マシン数()
            RichTextBox1.Text = TheMaster.TestSC(NumOfProc)
        End Sub

End Class
```

（10）環境設定用のClientEnvironmentsクラス

リモーティング関連コードよりも，なんだか長くなってしまったが，環境設定するクラス。[machine]というテキストファイルを読み込んで，そこに書かれているマシン名をリストアップ。そのマシンたちに向かって仕事の要求を投げることになる。

```
Imports System.IO
Imports System.Text
Public Class ClientEnvironments
    Protected Shared _refMachines() As String
    Public Shared ReadOnly Property refMachines() As String()
        Get
            If isLoaded = False Then
                loadfile()
            End If
            Return _refMachines
        End Get
    End Property

    Protected Shared _マシン数 As Integer
    Public Shared ReadOnly Property マシン数() As Integer
        Get
            If isLoaded = False Then
                loadfile()
            End If
            Return _マシン数
        End Get
    End Property

    Private Shared ClientEnvironments As New ClientEnvironments()

    Private Sub New()
        loadfile()
    End Sub

    Protected Shared Function ConvertStringToByteArray(ByVal s As [String]) As [Byte]()
        Return (New UnicodeEncoding()).GetBytes(s)
    End Function 'ConvertStringToByteArray

    Protected Shared filename As String = "machines"
    Protected Shared isLoaded As Boolean = False

    Public Shared Function loadfile() As Boolean
        Try
            Dim fsread As New FileStream(filename, FileMode.Open, FileAccess.Read)
            Dim str As StreamReader = New StreamReader(fsread)
            Dim buffer As String = str.ReadToEnd()
            Dim linebuffer() As String = buffer.Split(ControlChars.Lf)
            Dim oneline As String
```

```
            Dim it As IEnumerator = linebuffer.GetEnumerator()
            Dim i As Integer
            ReDim _refMachines(linebuffer.Length - 1)
            _マシン数 = 0
            While it.MoveNext() = True
                oneline = it.Current()
                Dim wordbuffer() As String = oneline.Split(ControlChars.Cr)
                If wordbuffer(i).Length > 0 Then
                    _refMachines(_マシン数) = wordbuffer(i)
                    _マシン数 += 1
                End If
            End While
            str.Close()
            fsread.Close()
            Debug.WriteLine("マシン数=" + _マシン数.ToString)
            isLoaded = True
            Return True
        Catch
            _マシン数 = 1
            ReDim _refMachines(1)
            _refMachines(0) = "localhost"
            _refMachines(1) = "localhost"
            Return False
        End Try
    End Function

End Class
```

3.3 Remotingをインストールする

クライアントアプリケーションはコピペでもインストールしたことになるのだが，Windowsサービスはそれだけでは済まない。そこで，インストーラを作ってしまおう。今回は，面倒なので，クライアントアプリとWindowsサービスの両方を同時にインストールするインストーラにする。

（1）デプロイメントプロジェクトの追加

インストーラもプロジェクトのひとつで，

[プロジェクトの追加→新しいプロジェクト→セットアップ/デプロイメントプロジェクト]の[セットアッププロジェクト]で新しいプロジェクトとして作る。

図25　セットアッププロジェクトを作る

（2）プロジェクトにインストールするものを追加

[プロジェクト→追加→プロジェクト出力]を選んで，RemotingWindowsとRemotingServerを追加しよう。

RemotingWokerとRemotingMasterは自動的に追加されるので余計なことをしなくてよい。

あとプロジェクトのプロパティにある[RemovePreviousVersion]をTrueにしておこう。こうすると，あとでインストーラを起動したときに，前のバージョンがインストールされている場合は，[修復 or 削除]を聞いてくる。削除を選べばWindowsサービスも含めてアンインストールしてくれる。

（3）セットアッププロジェクトをビルドすれば，インストーラができ上がる。

（4）Microsoft .NET framework 1.1のインストール

Visual Studio .NET 2003あるいはVisual Basic .NET 2003がインストールされたマシンにはすでにMicrosoft .NET framework 1.1[28]がインストールされているが，それ以外のリモートマシンたちにはインストールされてないかもしれない。インストールされてないときは，Microsoft .NET Framework Version 1.1再頒布可能パッケージを[Microsoft Download Center]からダウンロードしてインストールしておく。

（5）セットアッププロジェクトで作ったインストーラで，並列処理するすべてのマシンにRemotingServerやRemotingWindowをインストールする。

3.4　実行

（1）その前に

“machine”というテキストファイルに並列実行に参加するマシン名を，一行に1台ずつ書いておく。これをインストールしたフォルダに置く。電源の入っていないマシンや，インストールしていないマシンを書くと，実行時にエラーになるので，使えるものだけを書く。

28　Windows Server 2003は標準装備。.NETアプリを実行するためのランタイムの集合体で，JavaVM+JITに相当する。

（２）実行

RemotingWindow.exeをダブルクリックすれば実行開始。

ボタンを押せば，Remotingの呼び出しが始まり，終わるとRichTextBoxに結果が表示される。

３.５　つぎの段階へ向けて

今回は，ほとんどエラー処理というものを書いていない。

・　リモートマシンが呼び出しに応じられるかどうか？
・　リモートマシンがエラーを発生させたかどうか？
・　リモートマシンがいつまでたっても結果を返してこない場合は？

など例外処理が必要なことは，いろいろ思いつく。

とはいっても，ありあわせのPCを使った，声の届く範囲の並列処理の場合，例外処理をちゃんとプログラムする手間と，例外処理を書かなかったことによる被害のどちらが大きいかは微妙だ。

Ⅳ．まとめ or とりとめない

本稿ではWindowsでMPIを使う話とVBで並列処理をする話をした。「こんなものを使ってる人がいるのかあ？」と思った方々も多いだろう。でも，実際に使っている。

・　某都市銀行のデリバティブ値洗いシステム
　４台のPCで朝の取引前と，取引後の夕方に値洗いを実行
　これはMicrosoft AccessをGUIに使って，VBAからVB.NETを呼び出してRemotingで並列処理を行っている。稼動し始めてそろそろ１年になるが，特段の問題はおきていない。
・　天候デリバティブプライシングシステム
　最大８台のCPUを使ってモンテカルロシミュレーションを並列実行。
　実はこのプログラムから逆に本体から枝葉まで叩き落して，本稿のサンプルができている。
　私も信じられないことに，逐次処理部分を含めても８cpuで７倍の高速化を実現。

金融機関の業務システムのうち計算処理をしているようなものは結構Windowsで動いていることが多い。これを高速化するにあたって，UNIXに移植するのは手間がかかる。同じDLLをそのまま使えるなら，結果の検証という大変な手間をかけなくて済む。これは大きい。

科学技術計算でも，はじめはWindows PC１台であーでもないこーでもないとスクラップ＆ビルド。うまくいったら，全国共同利用センタのスーパーコンを使ったり，PCクラスタを買ったりすると思う。ただ，PC１台のつぎが，「Linux PCクラスタを買う」というのはちょっと冒険かもしれない。その前に，ありあわせのPCを動員して規模を大きくして試してみるのもいいかもしれない（図26）。

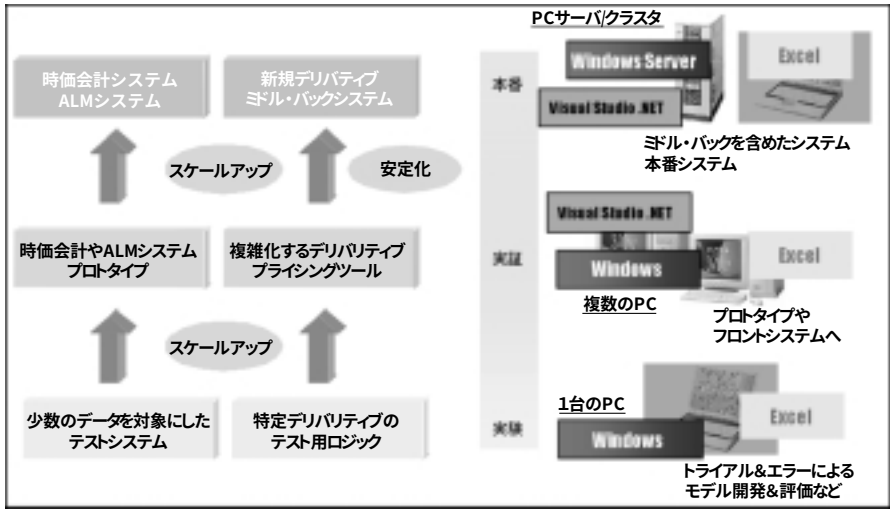


図26　Windowsで作り始めて，最後までWindowsということも

（たかはし　しゅん：日立製作所ビジネスソリューション事業部）