「一般の人々でも利用可能なグリッドプロジェクト」の実現

筑波大学附属坂戸高等学校

総合科学科　３年Ａ組　神林　亮

章構成

第１章　はじめに

第１節　問題提起

現在、家庭や学校などいたる所でパソコンという物は普及している。しかし、それらの全のパソコンが本当に有用に利用されているかというとそうではない。つまり、待機（電源が入っていない、利用していない）状態のパソコンが現代の世界には溢れていて、社会全体では処理能力を余らせているという事である。しかし、この状況というのは、研究施設などを除いた話で、現在でも研究施設などでは扱うデータの増大などに伴って非常に大きな処理能力を必要としている。そのため研究施設では「地球コンピューター」呼ばれる、体育館と同じぐらいの面積を必要とするスーパーコンピューター、複数台のパソコンをケーブルで接続して計算を行わせる「パソコンクラスタ」、複数のスーパーコンピューターを接続する、などの方法や技術を利用して大きな処理能力を得ようと努力している。

そこで、数年前から非常に注目されている技術が「グリッドコンピューティング」（この定義は非常に広く捉えられる場合もある）と呼ばれるものである。

この技術は世界のコンピューターがインターネットなどのネットワークで繋がっていることを利用し、遠隔地のコンピュータを何台も接続する事によって処理能力を得ようとするものである。この技術の非常に優れた点は多数あるが、私は最初に述べた社会全体で余っている処理能力を非常に有用に利用する事ができるという点と、コストの面から見てとてもリーズナブルであるという点を挙げたい。そういった理由からグリッドコンピューティングは非常に注目されるのである。

「グリッドコンピューティング」のプロジェクトとして有名なものを一つ挙げるとすれば「ＳＥＴＩ＠ｈｏｍｅ」などが非常に有名である。これは電波望遠鏡で観測した宇宙の映像を分割し、一般家庭のパソコンなどで解析してもらうというもので、全世界ですでに数百万人が参加していると言われている。このようにグリッドコンピューティングのプロジェクトに参加する事は、現在非常に簡単に一般の人々でもできるようになっている。しかし､その技術を利用する側に一般の人々が回る事ができているかというとそうではない。いまだにグリッドコンピューティングは研究者たちしか利用する事のできない技術的に高度な、高校生程度の技術では手の届かないものなのだろうか？

私はグリッドコピューティングについて調査をしていくうちに、公表されている情報が非常に難解で、研究者でなければ理解する事は困難であろうと考えた。そこで、その点を疑問に思い、できるだけ高度な技術を利用せずに、一般の人々でもグリッドコンピューティングは実現可能なのかという事を研究し、それを利用して、学校での余剰処理能力の有効利用と、高校生入学時から目指していたグリッドプロジェクトの実現を行った。加えて、それに伴う数学的アルゴリズムについても深く研究していった。

第２節　研究の動機

私は高校生活の間でグリッドコンピューティングという無限の可能性を持った技術と、いかに効率よく、またコンピュータに適合する形で計算を実現するのかという数学的なアルゴリズムの２つに非常に興味を持った。そして、私はこれらの二つを深めるために大学

に入学して、ＳＥＴＩ＠ｈｏｍｅを越えられるようなグリッドコンピューティングのプロジェクトを行いたいと考えている。その前段階として、大学に入学する前にグリッドコンピューティングを行う上でのノウハウを身につけておかなくてはならない。そして、それらはただ本を読む、調査をするだけではなく、実際にプロジェクトを実現していく過程での問題解決の中でこそ身につく事だと考え、準備を行っていた計画を実行に移そうと思い立った。

第３節　研究の意図

残念な事に、私の通っている学校では下のような配置でパソコンが一日の大部分を待機状態のまま置かれている。この点は第１章で述べた余剰の処理能力にぴったりと合致した。そこで、これらのリソースを利用しない手はないと考えた。しかし、大規模なグリッドコンピューティングを行う上で一番重要な、サーバとして扱える程の処理能力を持っているコンピュータは存在せず、また、「一般の人々でも実現可能」であるためには研究施設が利用するような高度なプログラミング言語というものも利用する事はできない。それらの制限の上で、一人の高校生がどこまでのプロジェクトを実現する事ができるのかという事を示した。

そのためには、研究施設では考える必要の無いような事象、また、まったく新しい考え方でネットワークを構築、プログラムを製作する必要があったが、そういった所にこの研究の価値を見いだしたのである。

※参考（私が調査した校内でリソースとして利用可能なパソコン台数）

メカトロ実習室	２４台	英語科教官室	５台
ＣＡＤ室	３１台	国語科教官室	４台
工業科教官室	２台	商業科教官室	７台
農業科教官室	４台	商業デザイン室	２１台
計測制御室	５台	家庭科教官室	５台
Ａパソコン室	４２台	化学準備室	２台
Ｂパソコン室	４３台	生徒会本部室	２台
数学家教官室	４台	工場準備室	５台
地歴公民科教官室	４台	事務室	５台
進路相談室	４台	校長室	１台
副校長室	１台	教務室	３台
保健体育科教官室	３台	物理準備室	１台
サーバー室	８台	図書室	１台
生物準備室	１台	保健室	１台

合計　２３９台

第４節　研究の進め方

校内のパソコン室と、インターネット上で、グリッドコンピューティングのモデリングを行った。しかし、それを実現するには情報がまったく存在しない事から、まったくゼロの所からスタートして自分独自の方法で製作を行わなくてはならなかった。そして、それらの過程で問題点、改善点を一つずつリストアップして、解決していき、速度や効率の変化をグラフ・数値など具体的な形で記録に残した。また、それらのソフトウェアの製作やネットワークの構築を行う上では、グリッドコンピューティングについての知識はもちろんの事、そこに至るまでの歴史、例えば、スーパーコンピュータの並列技術やパソコンクラスタなどの技術についての知識も活用した。さらに、ソフトウェアを作成する場面では数学のアルゴリズムが最も重要であったと言っても過言ではない。そういった所で、今までに得てきた知識を最大限に利用し、さらにそれだけではなく、積極的に新しい情報を集める事を行った。その家庭では、すでにだれかが書いた書籍や論文を調査する事ももちろん重要であったが、他の人々ではなかなかできないような調査の方法も必要となった。そこで、校内へのグリッドコンピューティングについての意識調査や、日本国内でグリッドコンピューティングを行っている研究施設で話を聞くなどの方法をとった。

研究は以上の様に、製作と調査研究の二つの方向性で進めた。

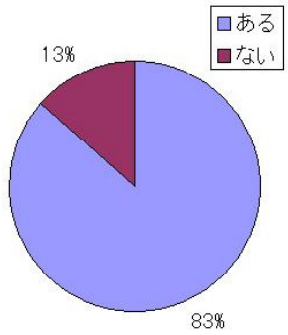
第５節　研究にあたっての調査

研究を行うにあたって、実際に人々のパソコンの利用状況、また、グリッド技術についての意識を把握しなくてはならないと考え、校内の３学年全１２クラス、約４８０名と、教員に対してアンケート調査を行い。その結果をエクセルを用いて集計し、グラフとして集計を行った。なお、無回答の場合は結果から除外した。

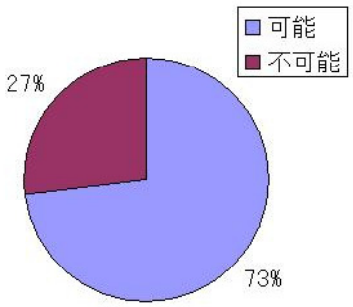
☆アンケート調査結果

・生徒に対して行ったアンケート調査の結果

パソコンが自宅に1台でもあるか（生徒）

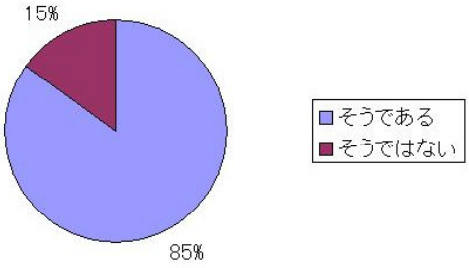


自宅でインターネットが利用可能か（生徒）

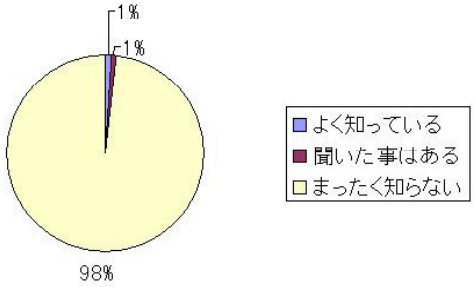


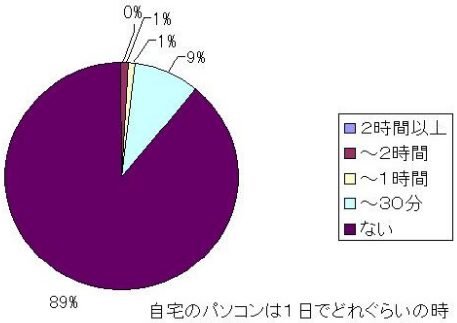
※インターネットが利用可能な場合の回答

接続は常時・定額接続であるか（生徒）



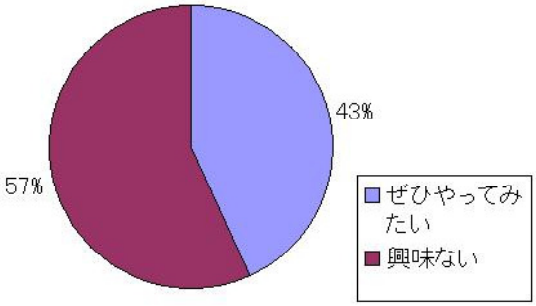
Seti@homeやGIMPSという言葉を聞いた事があるか（生徒）





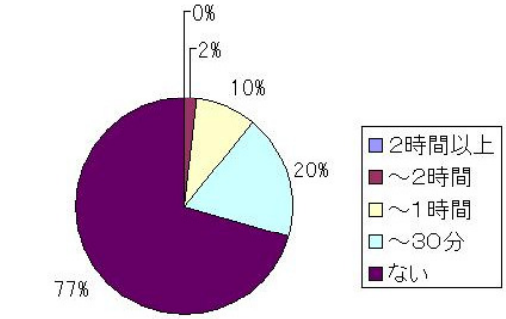
自宅のパソコンは1日でどれぐらいの時間、待機状態になっているか（生徒）

ちょっとしたソフトを入れておくだけで、宇宙人探しや白血病の治療法研究に参加できるとしたら、やってみたいか（生徒）

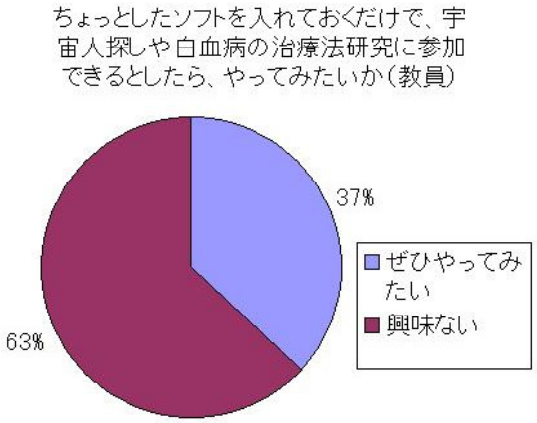


・教員に対して行ったアンケート調査の結果

パソコンの所有率、インターネット利用の可否、常時接続であるかという質問に対しては学校の教官室でパソコンが利用できるのですべて１００％となった。また、Ｓｅｔｉ＠ＨｏｍｅやＧｉｍｐｓという言葉を知っている教員は一人も存在しなかった。



自宅のパソコンは1日でどれぐらいの時間、待機状態になっているか（教員）



また、次のような質問に対しては以下のような回答があった。（生徒・教員合わせて）

質問：ユーザーをグリッドのプロジェクトに参加する気にさせるみかえりを挙げるとしたら？

・プロバイダ無料　・電話代が安くなる　・参加証がもらえる　・通信料無料
・提供した時間に対して現金還元　・賞金　・主催者が販売している商品を提供する
・発見者としての名誉　・何らかのサービスの提供　・特許権などの一部譲渡
・懸賞　・ポイント制でのプレゼント　・参加者を招いてのイベント開催
・ホームページサーバのエリアの提供　・パソコンの速度向上　・必要ない
など

質問：グリッド技術を用いて計算すべきと考えられる問題は？

・円周率を求める　・白血病、エイズ、がんなどの不治の病の治療薬　・宇宙人探索
・因数分解　・新しい元素の発見　・大規模な確率についての演算　・人工知能の開発
・天体についての研究　・ブラックホールの質量　・老化の解明　・ＤＮＡの解明
など

☆アンケート調査を行ってみて

生徒、教員両方において、８割以上がいつでもパソコンを利用できる状況にあり、また、インターネットも大きな割合で普及しており、まさに情報化社会を感じさせるような結果となった。また、定額・常時接続であるブロードバンドも安価に、また容易に導入する事が可能になった今、これも高い割合であった。

以上のような状況から、グリッドプロジェクトというものが普及していてもおかしくはないのだが、Ｓｅｔｉ＠ＨｏｍｅやＧｉｍｐｓなどのプロジェクトの名を知っている者は１～２％と非常に低い値となった。この事は、やはり、まだグリッドというものがあまり社会に普及していないという事を顕著に示している。しかし、グリッドのプロジェクトに興味を持つ者の割合は４０～５０％前後と高く、一般のパソコンユーザーにもっと情報があれば、グリッドはすぐに受け入れられていくのではないであろうか。

その他で、特筆すべき事としては、やはり、セキュリティの不安を示す者が多かった事である。これは、特に教員に多く見られ、漏洩してはならない情報を、生徒より、より多く管理しているという条件上、グリッドへの関心が生徒より低くなってしまった事は理解できる。この点を解決する事がグリッドの普及への一番の問題点なのかもしれない。

もう一つ特筆すべき事として、グリッドプロジェクトへの参加のための見返りとして何が考えられるかという問いに対して、必要ないと答えた者が非常に多かった事が挙げられる。

現在のグリッドのプロジェクトで抵は、何かを発見した場合に賞金を分与、抽選で何がもらえる、完全なボランティア３つのタイプに分ける事ができるが、これから先はどんどんグリッドが普及し、プロジェクトがあふれ、その中からユーザーが選択を行う中で、市場の競争原理のように、サービスによってプロジェクトが選ばれていくようになるだろうと予想できる。しかし、グリッドの利点は、ある程度無償で一般ユーザーから処理能力を借りる事ができる点であって、そこに対価が発生してしまう場合は、プロジェクトそのものが成り立たなくなってしまう可能性を生じさせてしまう。そうなるとやはりボランティアという形の参加者が多くなくてはプロジェクトは成り立たないのかもしれない。しかし、今回の調査の結果からわかる様に、すべてのユーザーが対価を求めていくのではなく、ボランティア精神による参加者はやはり多く存在するので、グリッドがサービスによって選ばれる時代は、まだまだ来ないと言えるだろう。

他にも興味深い意見が多くあり、アンケート調査などの生のデータを参考にすることは研究において大事だと感じさせられた。

第２章　グリッドアプリケーションの製作と実現

第１節　製作の概要

まずは、素数を発見するプロジェクトのためのソフトウェアの製作を行い、その後に他のテーマを考える事とした。そこで、なぜ題材を「素数」としたのかというと、素数を発見するという行為は、調べる範囲を分散させて複数のプロセスで計算をするという事が比較的容易であるということが大きな理由である。その事がが何を意味するのかと言うと、基本的に（例外もあるが）それぞれの計算が独立していて他の範囲の値などに依存する事がなかったからである。だから、例えばプロセスＡが１から１００の範囲の計算を終わらせていなかったとしても、プロセスＢはそれを待つ事なく、次の１００以降の範囲の計算をどこからでもはじめる事ができるのである。この事はあまり大事ではないように思えるかもしれないが、この事が成立しなければ分散して計算をさせるという事はできないのである。その点で、素数を発見するという事は始めの題材として適している。加えて、素数というのは暗号の分野で利用されており、インターネット上での認証などで使われる公開鍵暗号方式と呼ばれるものなども素数なしには成立しないという状況や、求めるアルゴリズムも非常に奥深く、それについての研究も同時に行えるという理由から、私は非常に興味を持った。

この製作を行う上で、まず最初に決めなくてはならなかった事は、使用言語を何にするかという事である。

「一般の人々が利用できる」という前提の上にこの研究は成り立っているので、あまり難解な言語を利用すべきではない、その条件の上で、現在高校生や一般の人々が利用している言語としてはＶｉｓｕａｌ　Ｂａｓｉｃが非常にポピュラーだと考えた。この言語は、プログラミングの入門用的な性格を持ったＢａｓｉｃをＷｉｎｄｏｗｓの環境で利用できるようにした言語である。だが、現在ではＶｉｓｕａｌ　Ｂａｓｉｃでもネットワークやデータベースに関する非常に高度なプログラムも作成が可能であり、実際、プログラミング業界でも小規模な業務用のプログラムの大部分はＶｉｓｕａｌ　Ｂａｓｉｃで作られていると言われている。しかしやはりこの言語のネックであったのは、それにより作成されたプログラムの動作スピードである。この言語の欠点は現在でも、改善されているとは言えない。

この部分がいかに克服されていくかがこのプロジェクトの成功・失敗を決めてしまう要素であったと言っても過言ではない。そこで、大部分はＶｉｓｕａｌ　Ｂａｉｓｃで製作したが、本当にスピードを必要とする部分は、Ｃ言語で製作を行った。この言語は、現在でも産業界の広い分野で利用されており、スピードという面で見ても、また機械語に近い事で、細かい部分をプログラミングする事ができるという面で見ても、非常に優れている。加えて、代表的なプログラミング言語であるので一般の人々でもある程度は扱う事ができると思われる。しかし、これでプログラミングのすべてを作ってしまったのではあまりにプログラムが難解になってしまい、「一般の人々が利用できる」というテーマに反してしまう。なので、この言語を利用する部分は必要最小限に押さえ、製作を進めた。

第２節　素数発見プロジェクト（パソコンクラスタ的方向性）

第１項　製作の２つの方向性

素数発見プロジェクトを実現する上で私はその形態として２つの方向性を考えた。まず１つ目はグリッドの技術を用いたパソコンクラスタのような方向性。２つ目はＳｅｔｉ＠Ｈｏｍｅのようなボランティアコンピューティングの方向性である。

まずは前者の方向性で製作を進めることにしたのだが、ここで、私が考える前者と後者の違いを挙げる。

・前者はコンピュータがある程度、集まって存在しており、プロジェクトの管理者の手の届く所にリソースがある（例えばＬＡＮ内などの規模の小さなネットワーク上でかつ、１部屋か２部屋のせまい範囲に存在する場合。一方、後者はコンピューターがネットワーク上に散らばって存在しており、管理者から直接そちらを操作するという事はできない。例えば、インターネットのような大規模なネットワーク上または、Ｌａｎ内でも一部屋の中に集まっているのではなく複数の場所に散在している場合)。

・後者では、クライアントのソフトを不特定多数のユーザーが利用するので、そういった所のユーザーの利便性などについて考える必要がある。

・前者は基本的に接続したまま作業を継続しても良いが、後者は、データの受け渡しの時だけ接続を行うというかたちになる。

・後者では、いつでもクライアントの台数や、返ってくるデータなどが把握できるわけではなく、また不特定多数の人々が関わるので、悪意を持った利用者に妨害や、データの改ざんが行われる可能性があり、データの信頼性についてよく考える必要がある。

・前者はクライアントの処理能力を１００パーセント用いる前提で製作を進めて良いが、後者では､クライアントの本来の利用者に負荷をかけないように配慮しなくてはならない。

・前者は、ある程度閉じられたネットワーク内で行われるので、それほどセキュリティについて心配する必要はないが、後者では、その点に十分配慮する必要がある。

・前者は、クライアント（コンピュータ）は基本的に管理者の権限で利用できるのでよいが、後者では、処理能力を提供してもらう立場であるので、いかにユーザーに参加意欲を与えるのかも問題となる。

以上のような違いを踏まえ、前者の製作を自宅で、調査をパソコン室において行い、後者の製作を、同じく自宅で製作し、インターネット上で利用できるようにした。

☆調査環境

・使用したパソコン

〇高校のパソコン室

機　種 ： Fujitsu FMV-6366DX2C

ＣＰＵ ： Intel Celeron 366MHz

メモリ ： 32MB

Ｏ　Ｓ ： Microsoft Windows 98

〇自宅

機　種 ： 組み立て型パソコン

ＣＰＵ ： Intel Pentium III 733MHz

メモリ ： 768MB

Ｏ　Ｓ ： Microsoft Windows Me

・使用したアプリケーションソフト

Microsoft Visual Basic 6.0 Professional Edition

Microsoft Visual C++ 6.0 Proffessional Edition

高校のパソコン室の風景

第２項　試作１「基本的な素数発見プログラム」

まずは、非常に基本的な素数発見のプログラムを製作した。このプログラムは本当に基本的なアルゴリズムで速度的にも非常に遅いものだが、まずは、このプログラムを利用してネットワークの構築に臨んだ。

☆製作過程

素数の基本的な定義を挙げておくと「その数と、１以外に約数を持たない自然数」という事である。つまり、定義に従うならば、ある数ｎが素数かどうか判別するためにはその数までの自然数で順番に割って、割り切れるか判定していけば良いのである。しかし、それではあまりにも能率が悪いので、２以外の偶数は素数ではあり得ない事より２以外の２の倍数は、ｎから除く事とした。

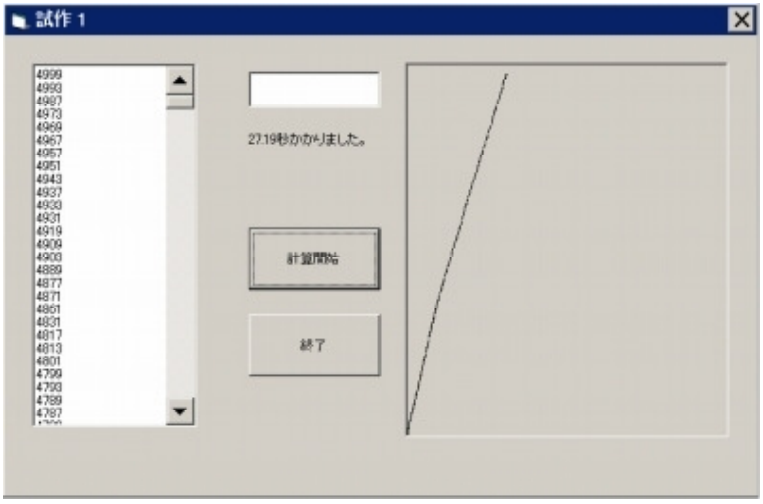
☆製作の過程での問題点

Ｉ：プログラムの性能調査はどのようにして行うか？

プログラムを作成していく上で、性能を調査するために、どれだけの時間で、どれだけの範囲の素数を発見する事ができたのかをグラフとして示す、そのためのグラフは実際にプログラムとして描画させる事も可能だが、データの加工の事も考え、プログラムの中で順次テキストファイルにその時点の経過時間と計算が終了した範囲を書き込み、エクセルで取り込んでからグラフを作成した。

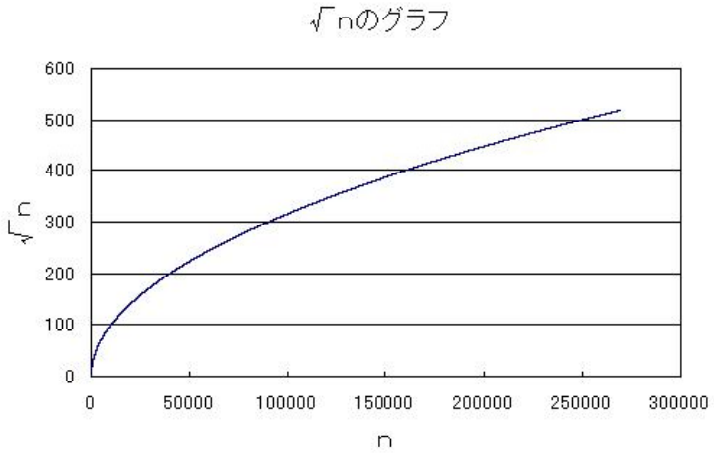
☆最終的なプログラムの流れ

計算させる範囲を入力する
↓
１から指定された範囲まででループを行う
↓
それぞれについて指定された範囲まで、３以上の奇数で割り切れるか判定する
↓
最後まで割り切れなかった場合、それを表示する



試作１の実行画面

☆改善すべき点



・上のグラフのように時間が経過するに従って、判定している素数の値も大きくなり、その結果、判定のための試行の回数が増え、素数を発見する速度は低下していってしまっていた。

第３項　試作２「通信プログラム」

通信プログラムを製作するにあたって、始めに基本的な通信プログラムを製作した。

☆製作過程

Ⅰ：通信を行うためのコントロールの選択

ただ通信を行うと言っても、Ｖｉｓｕａｌ　Ｂａｓｉｃの標準関数だけでは通信を行うプログラムを作成する事はできない。そこで、Ｖｉｓｕａｌ　Ｂａｓｉｃでは、さまざまな機能を有するコントロールというものが用意されている。

例えば通信を行うもので言えば、ＦＴＰ、ＨＴＴＰでの通信を行えるＩｎｅｔ（インターネットトランスファー）コントロールや、シリアル通信を行う事ができるＭｓｃｏｍｍコントロール、ＴＣＰ・ＵＤＰでの通信を行う事ができるＷｉｎｓｏｃｋコントロールなどがある。これらの中でＬａｎ内通信、また、インターネットを介してパソコンが相互通信を行うのに最も適するのはＷｉｎｓｏｃｋコントロールである。

○Ｗｉｎｓｏｃｋコントロールの概要

Ｗｉｎｓｏｃｋコントロールはプロジェクトの中核を成す非常に重要な部分なので、主なメソッド（関数)、イベント、プロパティを挙げてげておく。

・メソッド

ｏｂｊｅｃｔ．Ｌｉｓｔｅｎ

リモートコンピューターからの接続要求を待つ

ｏｂｊｅｃｔ．Ａｃｃｅｐｔ　ＬｉｑｕｅｓｔＩＤ

Ｃｏｎｎｎｎｅｃｔｉｏｎ　Ｌｉｑｕｅｓｔイベントから与えられる引数ＬｉｑｕｅｓｔＩＤを渡して、接続要求を受ける。（ＴＣＰプロトコルでしか必要としない）

ｏｂｊｅｃｔ．ＳｅｎｄＤａｔａ　ｄａｔａ

引数ｄａｔａで指定した変数やデータをリモートコンピュータに送信する。

ｏｂｊｅｃｔ．Ｃｌｏｓｅ

接続している、または接続要求を待っているソケットを閉じる。

ｏｂｊｅｃｔ．Ｇｅｔｄａｔａ　ｄａｔａ

リモートコンピュータから送られてきたデータを引数Ｄａｔａに格納する。

・プロパティ

ｏｂｊｅｃｔ．Ｐｒｏｔｏｃｏｌ［＝ｐｒｏｔｏｃｏｌ］

引数ｐｒｏｔｏｃｏｌにｓｃｋＴＣＰＰｒｏｔｏｃｏｌかｓｃｋＵＤＰＰｒｏｔｏｃｏｌを指定して、利用するプロトコルを指定する。

ｏｂｊｅｃｔ．ＬｏｃａｌＰｏｒｔ＝ｐｏｒｔ

引数ｐｏｒｔに指定したポート番号をローカルポートに指定する。（０を指定した場合適当な値が自動的に選択される

ｏｂｊｅｃｔ．ＲｅｍｏｔｅＰｏｒｔ＝ｐｏｒｔ
上と同じように、接続先のポート番号を指定する。

ｏｂｊｅｃｔ．ＲｅｍｏｔｅＨｏｓｔ＝ｓｔｒｉｎｇ
リモートコンピュータのアドレスを指定する。ホストネーム(例：ＦＴＰ／／ｆｔｐ．．ＸＸＸＸ．．ｃｏｍ)も IP アドレス（例：１００．０１．１１)も可能。

ｏｂｊｅｃｔ.Ｓｔａｔｅ
接続状態に対応する値を返す。

・イベント
ｏｂｊｅｃｔ＿ＣｏｎｎｅｃｔｉｏｎＲｅｑｕｅｓｔ（ｒｅｑｕｅｓｔＩＤ　ａｓ　ｌｏｎｇ）
リモートコンピュータからの接続を要求してきた時に発生する。引数ｒｅｑｕｅｓｔＩＤには接続要求の借別子が格納されている。

ｏｂｊｅｃｔ＿Ｃｏｎｎｅｃｔ（）
接続処理が完了した時に発生する。

ｏｂｊｅｃｔ＿ＤａｔａＡｒｒｉｖａｌ（ｂｙｔｅｓＴｏｔａｌ　ＡＳ　ｌｏｎｇ）
新しいデータを受信した時に発生する。引数ｂｙｔｅｓＴｏｔａｌには取得できるデータ全体のサイズが格納されている。

ｏｂｊｅｃｔ＿ＳｅｎｄＣｏｍｐｌｅｔｅ
送信処理が完了した時に発生する。

Ⅱ：使用するプロトコルについて
ＷｉｎｓｏｃｋコントロールではＴＣＰとＵＤＰプロトコルを使用する事ができるが、製作するソフトウェアの性格によってどちらのプロトコルを利用すべきかは変わってる。

・ＴＣＰプロトコル
ＴＣＰプロトコルはデータを送信する前に明示的に２台の間で接続を確立する必要があり、最初はどちらかが聞き手となるために、サーバとクライアントの区別が必要になる。
特性としては、エラー処理などが優れており、送信したデータの完全性は保障される。しかし、コンピュータのリソースを多く必要とし、また、送信処理を行う前にリモートコンピュータに確認をとるために送信する速度が遅くなってしまうという欠点を持つ。つまり、以上から考えて、ＴＣＰプロトコルを利用するのに向いているソフトウェアというのは、大量のデータを完全な状態で送る必要があり、エラー処理なども必要とする、デリケートな処理を行うようなサーバ・クライアントアプリケーションなどである。

・ＵＤＰプロトコル

ＵＤＰプロトコルはコネクションレス型のプロトコルで、明示的に接続を確立する必要がなく、一回の処理で複数のコンピュータにデータを送る事ができるという利点や、ソフトウェアを製作するのがＴＣＰと比べて容易になるという利点がある。また、送信速度という面でもＴＣＰプロトコルより優れている、しかし、エラー処理などの機能はほとんどなく、データが送信中に抜けてしまったりする場合も考えなくてはならなくなる。つまり、ＵＤＰプロトコルを利用するのに適しているのは、処理の完了を他のコンピュータに知らせるような場合や、一回の処理で複数のコンピュータにデータを送るチャットのようなアプリケーションなどである。

この時点では、まだどちらのプロトコルの方が適しているかの判断ができないので、ひとまず、どちらのプロトコルでも製作を行う事とした。

☆製作内容

校内のＬＡＮで、しっかりと通信が行えるか確かめるという意味も含めて、片方のテキストボックスに入力した文字が相手のパソコンに表示されるというプログラムを製作した。

☆最終的なプログラムの流れ

クラアント側で相手のパソコンのＩＰアドレスを入力し、接続要求を行う
↓
サーバ側のＷｉｎｓｏｃｋコントロールが接続要求を受け付け、接続を確立する
↓
Ｓｅｎｄｄａｔａメソッドでデータを送信する（どちらからも可能）
↓
データを受信した側がＧｅｔｄａｔａメソッドでデータを取得し、表示する

試作２の実行画面

第４項　試作３「１対１での素数発見プログラム」

試作２によって校内のＬＡＮで通信を行える事が確認できた。今度は試作２に試作１の素数を見つけるプログラムを取り込んだプログラムを製作した。

☆製作過程

サーバとクライアントが接続を確立してから、サーバがスタートのボタンを押すと、サーバがクライアントに計算させる素数の範囲の始りの値を与えて、それに従ってクライアントは与えられた始まりの値から、定められている値の範囲の素数を計算する（例えば、与えられた値が５で、定められていた値が１００であれば５～１０５の範囲の素数を求める)。そして、計算が終わったら、クライアントは見つけた素数の値をサーバに送り返す。それによって、サーバは戻ってきた素数をテキストボックスに表示し、また新しい素数の範囲の始まりの値をクライアントに送る。以上のループを繰り返して、素数を求めて行くというプログラムの製作を行った。

☆製作の過程での問題点

Ⅰ：サーバ側のテキストボックスに文字化けが表示されてしまう。

クライアント側で与えられた範囲に素数が無かった、つまり、データが空の状態で、サーバにデータを送ってしまい、それを表示した時に文字化けを起こしてしまっていた。

問題を改善するために、クライアントは与えられた範囲に素数がなかった場合、”１”という値をサーバに送信して、サーバも送られてきたデータが１だった場合のデータを無視するように
した。

Ⅱ：クライアントに計算させる範囲を与える方法はどうするか？

変数を用意し、初期値を５とする、そして、サーバがクライアントに素数の範囲を割り振る時は、その時点での変数の値を送信し、それから変数の値に「クライアントに一回で計算させる範囲の値」（例えば１０など）を加算する。これの繰り返しにより、５～の範囲を順番にクライアントに計算させる事ができた。

Ⅲ：データを一度しか送信していないのに、データを受信する側で何度もＤａｔａＡｒｒｉｖａｌイベントが発生してしまう。（データを受信してしまう）

最初は通信ケーブルの故障やノイズなどが原因なのかと考えたが、調べてみてもそういった点は見つからなかった。そこでプログラムの面も考えてみたが、やはりそこにも問題点を見出すことはできなかった。しかし、原因を調査していった結果。通信プロトコルの特性が原因だという事が解った。

ＴＣＰもＵＤＰも送信の処理が１回だったとしても、データが何度に分けられて相手に

届くかは予想できないのである。つまり、どちらのプロトコルもデータがどのように相手に届くかは保障しない、そのため、プログラムを作成する者がそこを考慮してプログラムを組む必要があったのである。

この問題を解決するためには、データが全て届いたか、それとも途中であるのか、の判断をデータを受ける側ができるようにする必要があった。そのため、データを送信する時に、データの末尾に終わりである事を示すために、＃をつけ送信し、それを、データを受信する側が識別するという方法をとった。

Ⅳ：ＵＤＰプロトコルで大きなデータを送ろうとすると「データグラムはバッファより大きいため切り詰められます」というエラーが出てしまう。

これもＴＣＰとＵＤＰのプロトコルの性質の差が現れたエラーだった。ＵＤＰプロトコルは送信するデータについてほとんど関知せず、ただ送るだけなので、相手側のパソコンのバッファの容量の事もプログラムの側で考えておかなくてはならなかったのだ。

これを解決するために、クライアント側が計算した結果をサーバに送る時に、ある程度のバイト数で区切って送信し、それをサーバ側でつなぎ合わせるという手順を挟むようにした。

☆最終的なプログラムの流れ

変数を読み込み、計算する範囲としてクライアントに送信する

↓

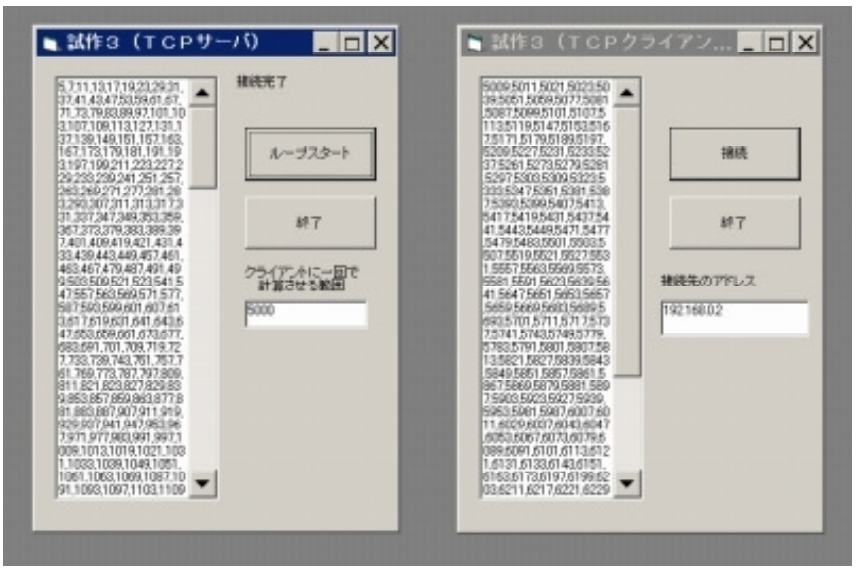
クライアントは受信した範囲に従って素数を探索する

↓

素数の探索が完了したら、発見した素数をサーバに送信する

↓

サーバは受信したデータが全て届いたか確認してからそれを表示し、再び計算する範囲をクライアントに送信する



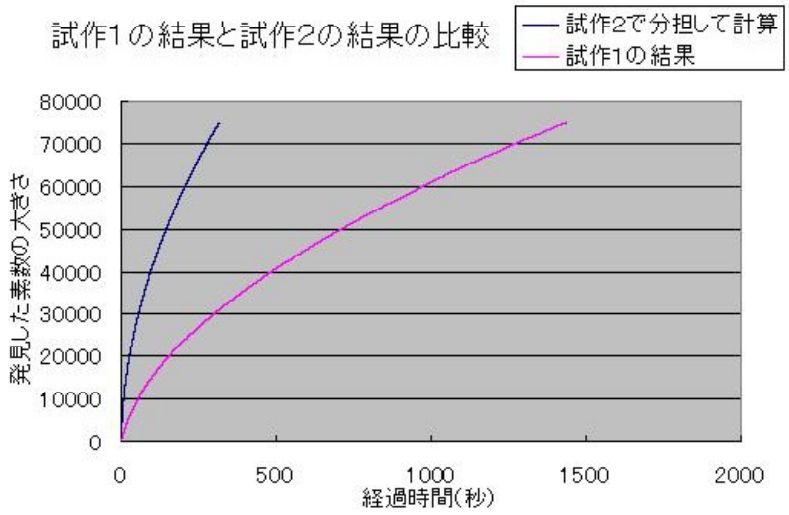
試作３の実行画面

☆性能調査

１：試作１との比較

ここで、サーバとクライアントの作業それぞれを１台ずつのパソコンで行い、合わせて２台で計算を行った結果と、試作１の１台で計算した結果を比較した。

グラフから２台のパソコンで作業を分担した方が早く計算できる事が分かる。（ここではサーバがクライアントに１００の範囲ずつ計算する範囲を与え、またプロトコルはＴＣＰで行った）



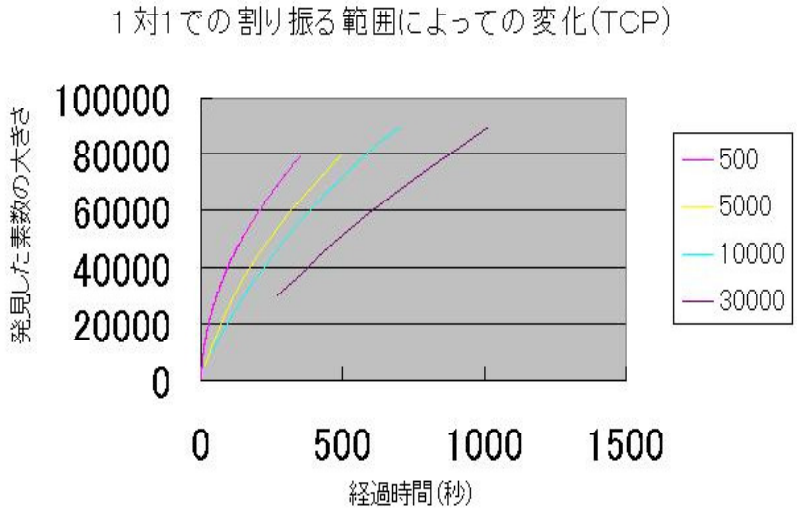
※グラフの傾きが急なほど、素数を発見する速度が速い事を示す

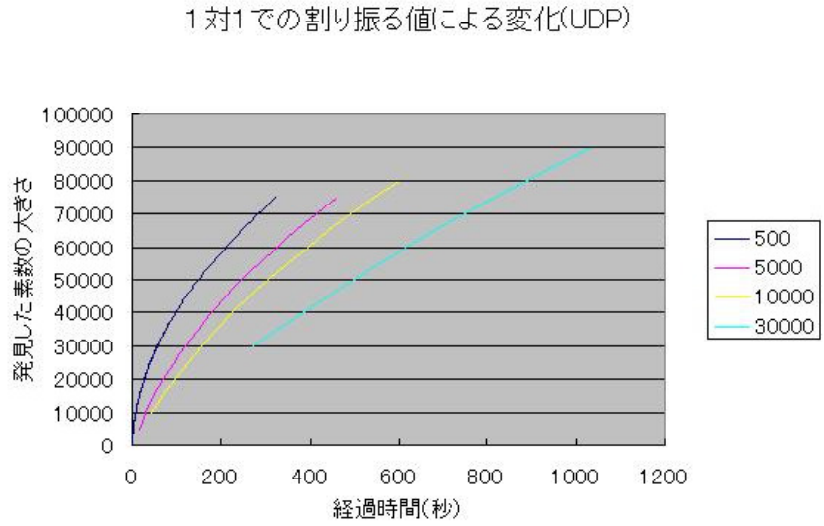
※グラフが曲線になっているのは試作１と同じように、時間が経過するごとに素数を発見する速度が低下している所から来ている。

Ⅱ：割り振る範囲の大きさによる変化

次にＴＣＰプロトコル、ＵＤＰプロトコルそれぞれで、クライアントに与える計算する範囲の大小によって、速度はどう変化するのかを調査した。

予想：与える範囲が小さい場合、それだけサーバとクライアントの通信を行う頻度が高くなるという事だから、通信の部分でのオーバーヘッドによって良い性能は見込めないだろう。つまり、与える範囲は大きい方が良いだろうという予想をした。

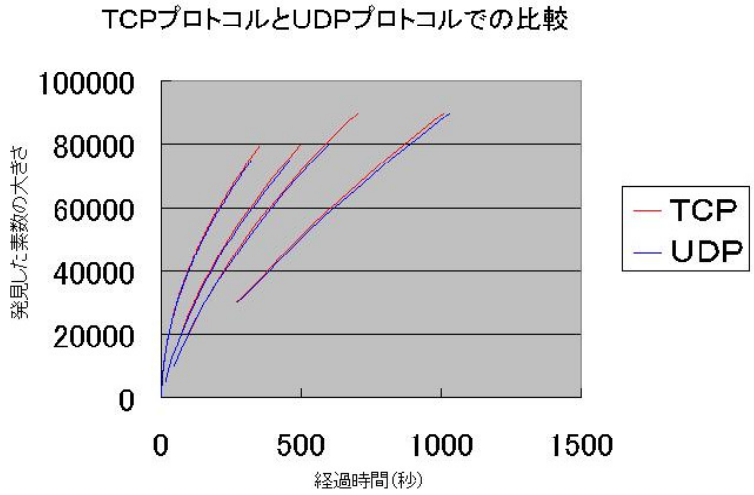




調査の結果、予想に反し、どちらのプロトコルでも与える範囲が小さい程、速度は速いという結果となった。それを受けこの結果の原因は何なのか考えた。私には、大きなデータを送るよりも、小さなデータを小分けにした方が通信でのオーバーヘッドが小さいのではないのではないだろうか、もしくは、クライアントの計算の中に、一度に計算する範囲が小さい方が、結果的に早くなるという要因がふくまれているのではないだろうかと思われる。しかし、いずれも予想でしかないので、この原因は引き続いて調査して行かなくてはならない。

Ⅲ：ＴＣＰ・ＵＤＰプロトコルでの比較

ここでは２つのプロトコルで製作を行ったので、２つのプロトコルで本当に速度の差は表れるのか調査した。（上記のグラフを２つ重ねた物）



本来ならば、ＵＤＰプロトコルの方が早いはずなのだが、微妙な差でＴＣＰプロトコルでの方が早かった。その原因は、おそらくＵＤＰプロトコルでは送信処理の部分で、数バイトずつに分けて送るという手順を経ているので、そこでのオーバーヘッドである。

調査の結果、とにかく今回のような場合では、ＵＤＰプロトコル・ＴＣＰプロトコルの間にそれほど性能の差が表れる事はなかった（これが、通信の部分役割の比重が大きくなるインターネット上、また、送るデータの量がもっと莫大な量になった場合は違うのかもしれない)。以上の結果と、実際に使用するにはＴＣＰが行っているようなエラー処理を行わなくてはならなくなり、さらに遅くなってしまうだろうという理由から、ＵＤＰプロトコルを使う利点はあまりない。そのため、ここから先の製作では、プロジェクトの軸としてはＴＣＰプロトコルを用いる事にした。

☆改善すべき点

・まだエラー処理を行っていない
・複数台で１台のサーバに接続できるようにする。

第５項　試作４「１対複数でのサーバ」

試作４では１対１での接続しか可能ではないので、サーバ側のソフトに手を加える事によって、１対複数台で接続を行えるようにした。

☆製作内容

サーバで扱えるＷｉｎｓｏｃｋコントロールを１つだけではなく、配列とし、０番のＷｉｎｓｏｃｋコントロールが接続要求の割り振り専用として、接続要求を受けるごとに新しいＷｉｎｓｏｃｋコントロールを作り、その新しいＷｉｎｓｏｃｋコントロールが、接続要求を送信してきたクライアントと接続する。その事によって一台のサーバで複数のクライアントに対して処理が可能となった。

☆変更された点の流れ

サーバ側のＷｉｎｓｏｃｋコントロール（０）が接続要求を受ける

↓

新しいＷｉｎｓｏｃｋコントロールを作成する（例えばＷｉｎｓｏｃｋコントロール（１）など）

↓

新しく作成されたＷｉｎｓｏｃｋコントロールが接続を確立する

☆改善すべき点

・複数台に対してループを行おうとすると、サーバ側の処理が間に合わず、ループを行っているパソコンのうち、最終的には、一台とだけしかループを続ける事ができなくなってしまう。

原因は、ＶＩｓｕａｌ　Ｂａｓｉｃはイベントドリブン型の言語であり、普通にプログラムを組んだのでは､処理を並行して行う事が出来ないという所から来ていると思われる。その証拠に、試作３のサーバのソフトを複数台起動し、それぞれクライアントと通信を行わせると、ループは途切れる事なく続ける事ができた。つまり、この問題の原因は、パソコンの処理能力や、通信経路の問題では無い。

第６項　試作５「マルチプロセスサーバー１」

まず、試作４の問題を改善するためにはどうすればよいのか考えた。そこで考えた解決方法は

Ⅰ：サーバ側が手作業で試作３のソフトウェアを複数起動する

Ⅱ：サーバ側で並列に処理を行わなくてはならなくなるほどの負荷を与えない

↑

※クライアントが計算する素数の範囲を大きくして、クライアントがデータを送信する頻度を小さくするか、データを送信するタイミングをずらすなどの方法

の２つである

前者は、問題解決の方法としてはあまりにも稚拙であり、また、解決方法がプログラムの方向から考えられていないので問題解決となっていない。

後者は、実際に試したが、それぞれのクライアントの処理能力がまったく同じであり、サーバにデータを送信してから到着するまでの間隔もまったく同じでないといけない。そうでないと、結局、頻度を大きくしたりタイミングをずらしたりしても、最終的には少しずつのズレから同じタイミングでサーバにデータを送る事になってしまう。この理由より、後者の方法も失敗。

ⅰ、Ⅱ、の方法をとる事ができない、つまり、サーバアプリケーションをシングルプロセスで機能させるという事は困難であるという事が以上より分かった。そこでＶｉｓｕａ

ｌ　Ｂａｓｉｃでマルチプロセスのアプリケーションを作成する方法を模索した。

○ＡｃｔｉｖｅＸ．ＥＸＥ

ＡｃｔｉｖｅＸ．ＥＸＥはＶｉｓｕａｌ　Ｂａｓｉｃでマルチプロセスのアプリケーションを作成するのに利用される技術で、ＡｃｔｉｖｅＸ．ＥＸＥとして作成したプログラムを他のプログラムから参照設定する事によって、利用する事ができ、それは呼び出し元のプログラムとはまったく別のものとして独立して処理を行う事ができる。

この方法を利用して試作４を作成した。

☆製作過程

このプログラムは、実行すると、一つのフォームが表示され、表示ボタンを押すと、サーバのフォームが一つずつ表示されて増える。それぞれのサーバのフォームは最初に実行したプログラムとは独立したプロセスで動いているので、試作３を複数起動した時のように接続し、ループを開始する事ができる。

☆製作の過程での問題点

Ⅰ：それぞれのサーバのフォームが接続要求を待つポート番号をどうするか？

呼び出し側のプログラムがサーバのフォームを呼び出す時に、順番に新しいポート番号を与えるように。

Ⅱ：複数に対して通信をする事が可能になった所でどのようにグラフを作成するか？

ここで再び性能調査のためにグラフを作成しようと思うのだが、一台で計算を行う、または一台を相手に計算させる場合のように安易にグラフを作成するためのデータを採取する事はできない事に気づいた。それはなぜかというと、複数に対して計算をさせているという事は、例えばクライアントＡが５００から１０００までの計算を終了したと通知して来た時に、まだクライアントＢが１から５００までの計算を終了させていなかったとする、ここで誤って、クライアントＡが計算を終了したからと言って１０００までの計算が終了した事にしたら、正確に計算の終了した範囲を示す事はできない。そのために、この問題を解決する方法を考えた。

この問題を解決するためには、計算終了の通知を受けた時に、通知をうけた値までのすべての範囲の計算が終わっているかの判断を行えばよいのである。そのために、グラフ描画用のアウトプロセスのフォームを用意した。それが行う流れは、

アウトプロセスのサーバのフォームが計算終了の通知を受ける

↓

ファイルに通知を受けた値を書き込み、グラフ描画のフォームに書き込み位置を送信

↓

グラフ描画のフォームはファイルの中の値を昇順に並び返る

例）５０・１５０・１００・３００→５０・１００・１５０・３００、という様に並び変える。この場合では１５０までの計算が終了しているという事になる

↓

計算が終了していると認識した値でグラフを描画し、また、エクセルでデータを扱うために、その値を専用のファイルに書き込む

データを利用できるように、その値を専用のファイルに書き込んでおく

↓

次回にグラフ描画のフォームが並び変えを行う時は連続でなかった値から、例の中の値で言えば３００から先の値を並び替える

というような流れとなった。

☆最終的な流れ

プログラムを実行する

↓

アウトプロセスのフォームを呼び出すためのボタンをクリックする

↓

アウトプロセスのフォームに適当なポート番号を与えてからそれを表示する

↓

それぞれのフォームが与えられたポート番号で接続要求を待ち、試作３のように独立して処理をすすめる

☆改善すべき点

①このプログラムはとりあえず、マルチプロセスでのソフトを作ってみるために製作したため、クライアントに割り振る範囲などについては考えていない。

②クライアントからの接続要求を受けるためのサーバのフォームを手動で表示させなくてはならなく、また、クライアント側で接続要求を求めるポート番号を入力しなくてはならない。

第７項　試作６「マルチプロセスサーバ２」

まず、試作５の問題点①を改善するための方法を考えた。

☆製作過程

・問題点①の改善

クライアントが複数になっても、順番に計算させる範囲の割り振りを行うには、試作２のように、変数を用意しておいき、その値によって割り振る方法を用いれば問題ないのだが、サーバのフォームが複数のプロセスで動作しているため、今回のような場合一つの変数を参照して、という方法は取ることができない。そこでいくつかの方法を試した。

ｉ：ファイルを変数の変わりとして用いる

それぞれのアウトプロセスのフォームがファイルの値を読み込む

↓

ファイルの値を書き換える

↓

読み込んだ値をクライアントに計算する範囲として送信する

上の方法を試した結果、クライアントが同じ範囲を計算してしまっている状況が確認できた。その原因は、複数のサーバのフォームが同時にファイルを利用しようとした場合に、ファイルに書き込む処理と、読み込む処理との間に生じるタイムラグのせいで、一つのプロセスが読み込みを終わらせ、書き込みを行う前に、他のプロセスが割り込んでまだ書き換えられていないファイルの値を読み込んでしまう。その結果、２つのプロセスがクライアントに同じ範囲を割り振ってしまうのだろうと予想した。しかし、残念ながらプロセス間でファイルをロックするなどの排他制御を行ってこの問題を解決しようとしたが、書き込みの頻度が高すぎるためか、その方法では解決できなかった。

ⅱ：クリップボードを変数の変わりとして用いる

この方法では、それぞれのプロセスが変数から読み込む、に書き込むのと同じように、クリップボードからデータを取得し、クリップボードに書き込む。この方法では、ｉの方法のような問題は起こらず、正常に動作する事ができた。

クリップボードの値を読み込む

↓

クリップボードに新しい値を送信して書き換える

↓

得た値をクライアントに送信する

①の問題が解決したことで、グリッドコンピューティングとして処理を行う事、つまり、処理を分散させて計算させる事が可能となった。しかし、その周りの諸機能の事がまだ考えられていない。試作５では、クライアントからの接続要求に先駆けて、フォームを手動で用意しておくという動作が必要となる。この事は、つまりクライアントが接続するという事を、事前にサーバ側が知っている必要があるという事である。しかし、多数のクライアントから接続要求を受ける可能性のあるサーバがそれぞれの一つ一つを把握しておくというのは不可能であるし、無駄な手間がかかってしまう。また、グリッドコンピューティングは複数台の接続を前提としているのでできるだけ人間が行わなくてはならない要素は極力減らすべきである。したがって②の問題を解決しなくてはならない。

・問題点②の改善

この問題を解決するためには人間が行う部分をそのままプログラムで置き換えてしまえばよい。そのために

Ⅰ：最初の実行フォームにＷｉｎｓｏｃｋコントロールを配置して、全てのクライア

ントは最初にそれに対して接続要求を行うようにする

Ⅱ：接続処理で行う通信回数を１回で考えるのではなく２回にする

という様に考え方の転換を行った。その結果、プログラムは

接続要求を受けた事に反応して、サーバ本体がクライアントにもう一度接続要求を行うべきポート番号を送り返し、そのポート番号で接続要求を待つアウトプロセスのフォームを新しく作成する

↓

クライアントは送り返されて来たポート番号に対してもう一度接続要求を行う

↓

新しく作成されたフォームとクライアントで接続が確立される

といったものになった

Ⅰ：ＵＤＰプロトコルで自動で接続処理を行うにはどうすればよいか？

すでに述べたが、ＵＤＰプロトコルはコネクションレス型のプロトコルなので、クライアントが接続要求を行って、サーバがその接続要求を受け付けるという手順で接続処理を行うのではなく、クライアント・サーバの両方で相手のアドレスを入力しなくてはならない（クライアントが接続処理を行っても、ＴＣＰプロトコルと違って、サーバにイベントは起こらない)。つまり、クライアント一方が相手のアドレスを知っていても接続処理を完了させる事ができないのである。これを解決するために、接続処理の部分だけにＴＣＰプロトコルを用いるというアイデアを考えた。主な流れは上の流れ図に従うが、プロトコルを２つ用いるという所が異なる。異なる点を示した簡略的な流れは、

クライアントはＴＣＰプロトコルでサーバに接続要求をする

↓

サーバはこの接続要求で得たクライアントのアドレスをＵＤＰプロトコルで用意したＷｉｎｓｏｃｋコントロールに渡して接続待ちを行わせ、クライアントには接続待ちをすべきポート番号を送り返す

↓

クライアントは､自分のＷｉｎｓｏｃｋコントロールのプロトコルをＵＤＰに切り替え、送り返されたポート番号で接続待ちをする

この手順を踏む事によって結果的にクライアントがサーバのアドレスを入力するだけで、ＵＤＰプロトコルでの接続を確立する(実際には接続はしていないが) 事が可能になった。

☆改善すべき点

①現在のプログラムではポート番号とアウトプロセスのフォームの配列の番号の割り当てで、順に新しい値を用いるようになっている、そのためクライアントから接続が切断されて、ポート番号やアウトプロセスのフォーム用の変数が利用されなくなってもその番号を

再利用しない。

第８項　試作７「マルチプロセスサーバ３」

試作６の問題点①が引き起こす事は、現時点のように、人間がサーバに付きっきりでいる間はよいが、自動で処理を完了させるようにプログラムして、サーバを稼働させたときに、複数台のクライアントが接続、切断を繰り返す事によって、有限である配列の変数や、ポート番号を再利用せずに進んでしまうことである。それによって、長い時間がたった時にサーバが動作不可能になってしまう可能性が生まれる。

☆製作過程

☆問題点①の改善

問題点①を改善するという事はつまり、クライアントから接続が切断された時に、そのクライアントと通信を行うために利用していた変数やポート番号を記憶しておき、次回、どこかのクライアントから接続要求を受けた時に記憶しておいたものを割り振れば良いという事である。そのために、いくつか考えなくてはならない問題があった。

Ⅰ：実行側のフォームとアウトプロセスのフォームでどのように通信を行うか？

試作６では、２つのプロセス間でクリップボードを利用する事によって通信（データ共有）を行ったが、今回の場合は、実行側はクリップボードの値を読み込むタイミングが分からないので、タイマーコントロールなどでファイルの値を絶えず監視する、などという事をしなくてはならなくなってしまう。その事は無駄な処理能力を消費してしまうのでクリップボードを利用する方法は用いるべきではない。そこでまず、ＷＩＴＨＥＶＥＮＴ変数を利用した方法を試みた。

１：ＷＩＴＨＥＶＥＮＴ変数を用いた方法

この方法はアウトプロセスのフォームからそのフォームを呼び出したプロセスへ通信を行うための一つの手法である。呼び出し元のプロセスのプログラムの中でＷＩＴＨＥＶＥＮＴ変数を宣言しておく事によって通信が可能となる。しかし、ＷＩＴＨＥＶＥＮＴ変数を配列で宣言する事ができないために、複数のプロセスを利用するプログラムの場合はあまり実用的ではない。また、それぞれのプロセスのために、対応するイベントを書かないといけないために今回のプログラムでは不可能であった。

２：ＤＤＥ（ダイナミック・データ・エクスチェンジ）を利用した方法

ＤＤＥはまったく独立した２つのアプリケーションでも通信が可能となる手法で、上記の２つの方法のような問題が起こらず、今回のようなプログラムにうってつけの方法であった。そのためこの方法を利用することとした。

Ⅱ：どのようにポート番号や変数を再利用するのか？

再利用を行うために、ポート番号では再利用専用の値を保持する配列を用意し、オブジェクト変数の方は利用し終わったオブジェクト変数を初期化するという方法をとった。それによって、新しいクライアントから接続要求を受けるまでに、いくつかのクライアントから接続が切断されても、実行側のフォームが次に利用すべき値を次々と記憶しておき、再利用する事が可能となる。

○変更された点の流れ

クライアントから接続を切断される

↓

クライアントと通信を行っていたアウトプロセスのフォームも終了すると同時に、そのフォームが保持している情報をＤＤＥで実行側のフォームに送信する

↓

実行側のフォームが受信したデータに従って、再利用専用ポート番号の配列にポート番号を記憶、アウトプロセスのフォームが利用していたオブジェクト変数を初期化する

↓

実行側のフォームが次に接続要求を受けた時に、再利用すべきポート番号・オブジェクト変数があるのならばそれを利用、そうでないのならば新しい値を利用する

☆改善すべき点

①クライアントでの素数の計算に時間がかかってしまっているので、そちらの改善が必要

第９項　試作８「Ｃ言語を取り入れたクライアント」

ここまで主にサーバについて考えてきて、一通りプログラムが完成してきた所で、クライアントが素数を発見する時間について考えた。

☆製作過程

この文章の冒頭でも述べたが、Ｖｉｓｕａｌ　Ｂａｓｉｃには動作速度が遅いという欠点がある。それを補わなくては、このプロジェクトの成功は見込めない。そのため計算を行う部分だけはＣ言語を用いる事とした。ここでは、本来素数を発見するためのアルゴリズムも吟味するべきだが、、それは後ほど研究するとして、とにかく、試作１と同じアルゴリズムをＣ言語に置き換えて、あたかも関数の一つであるように利用できるようにする方が先決であると判断した。

☆製作の過程での問題点

Ｉ：どうやって２つの言語で作られたプログラムを１つのプログラムとして扱うのか

ただＣ言語をプログラムの中に取り入れると言っても、言語が違うので、いままでのプログラミングとは勝手がちがう。そして、１つのプログラムとして扱うにしても、実際にはまったく違う２つのプログラムである。なので、２つのプログラムの間で通信を行う方

法を考えなければならない。これがもし、２つのプログラムとはいえ、両方Ｖｉｓｕａｌ　Ｂａｓｉｃで作られたプログラムであればＤＤＥなどを用いれば済む話だが、ここではそういった機能は用意されていない。そのため、まずその問題を解決しなければ製作を進めることはできなかった。

Ｉ－１　関数の呼び出しのようにプログラムを呼び出す

まず、あたかもＣ言語のプログラムを関数のように利用するためには、引数として、サーバから送られてきた計算すべき範囲を通知する必要がある。つまり、関数であるならば

ａ＝ｓｏｓｕｕ（１００）；

という風に利用する事を、別の方法で実現しなくてはならないのである。これを、Ｃ言語のプログラムを呼び出す時に、実行ファイル名の後に引数としてつけて実行できるようにするという事も可能であったが、ここではファイルを用いた。その方法は、

サーバから送られてきた計算すべき範囲をファイルに書き込む

↓

Ｖｉｓｕａｌ　Ｂａｓｉｃのプログラム側から、Ｃ言語のプログラムを実行する

↓

実行されたＣ言語のプログラムは計算を始めるために、ファイルから値を読み込む

という手順で、以上の手順を踏む事によって、引数を与えて関数を呼び出す事を、ファイルに値を書き込んでから、プログラムを実行するという事で代替させる事ができた。

Ｉ－２　関数の終了のようにプログラムを終了させる

次に、Ｃ言語のプログラムが計算を終えた時の事を考えなくてはならない。もしこれが関数であるならば、

ａ＝ｓｏｓｕｕ（１００）；

とした所の左辺に、計算によって得た値が代入されるのである。これを実現するためには、Ｖｉｓｕａｌ　Ｂａｓｉｃのプログラム側が、計算の終了を知るという事と、２つのプログラム間で計算して得た値を伝達するという事が必要。

以上の２つを実現するために、前者にはＩ－１で用いたファイルを利用し、後者には新しくもう一つのファイルを用意した。

まず、Ｖｉｓｕａｌ　Ｂａｓｉｃのプログラムの側では、タイマーコントロールを用意して一定時間毎に、Ｃ言語のプログラムの計算が終了したかを監視するという方法をとった。その流れは

Ｖｉｓｕａｌ　Ｂａｓｉｃのプログラム側がＣ言語のプログラムを呼び出すと同時に、２つ目のファイルの値を一定時間毎に監視するタイマーコントロールを起動する

↓

計算が終了する

↓

Ｃ言語のプログラムは、一つめのファイルに計算終了を示す”＃”、２つめのファイルに発見した素数を書き込む

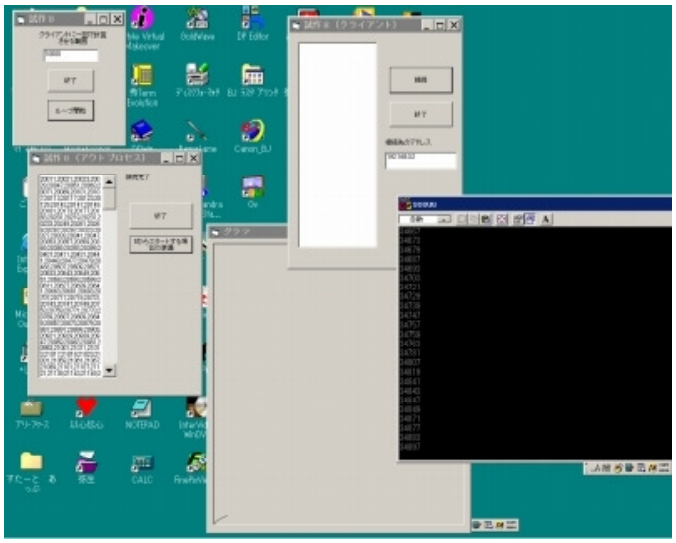
↓

タイマーコントロールが一つ目のファイルの”＃”に反応する事によって、Ｖｉｓｕａｌ　Ｂａｓｉｃ側のプログラムが２つめのファイルに書き込まれている発見された素数を読み込む

以上の手順を踏む事によって計算された値をＶｉｓｕａｌ　Ｂａｓｉｃのプログラムで読み取ることを実現できた

製作での過程での問題点Ｉを解決した事によって、Ｖｉｓｕａｌ　Ｂａｓｉｃのプログラムの中で、Ｃ言語のスピードで計算を行えるようになった。そして、この事によって素数を発見する速度は劇的にスピードアップした

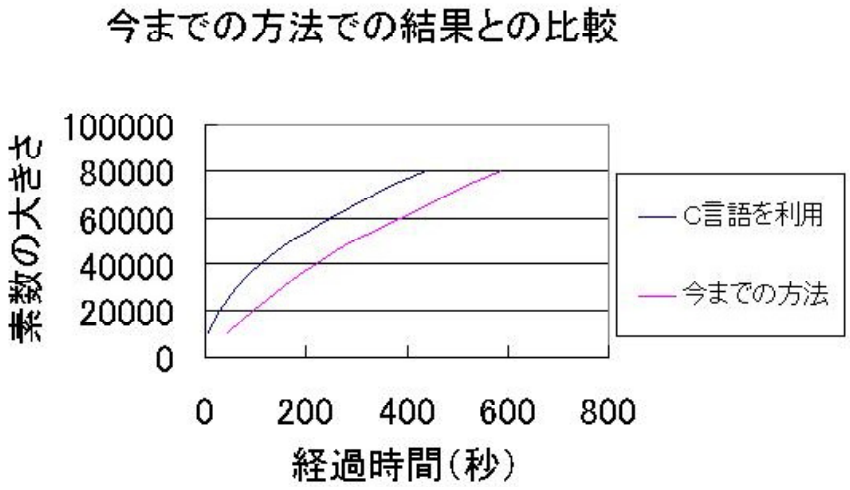
☆実行画面



☆性能調査

１：今までの方法との比較

Ｃ言語を取り入れた結果、今までの様にＶｉｓｕａｌ　Ｂａｓｉｃのプログラムによって計算していたのと比較して、どれほどの違いがあるのか調査した。

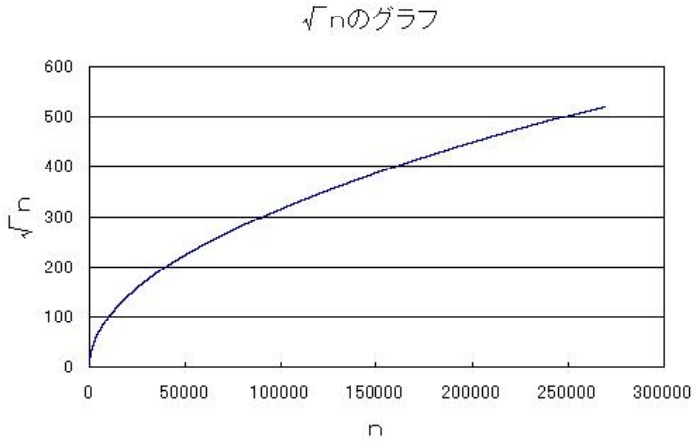


比較した結果、やはり、Ｃ言語を利用した方が格段に速い速度で素数を発見する事ができた。また、クライアントで計算を行う時に、ファイルを介して通信を行うという手順を余計に踏んでいるでいるにもかかわらずこれだけの差が現れたということは、やはりＣ言語がＶｉｓｕａｌ　Ｂａｓｉｃより計算の速度という面で優れているという事を端的に示している。

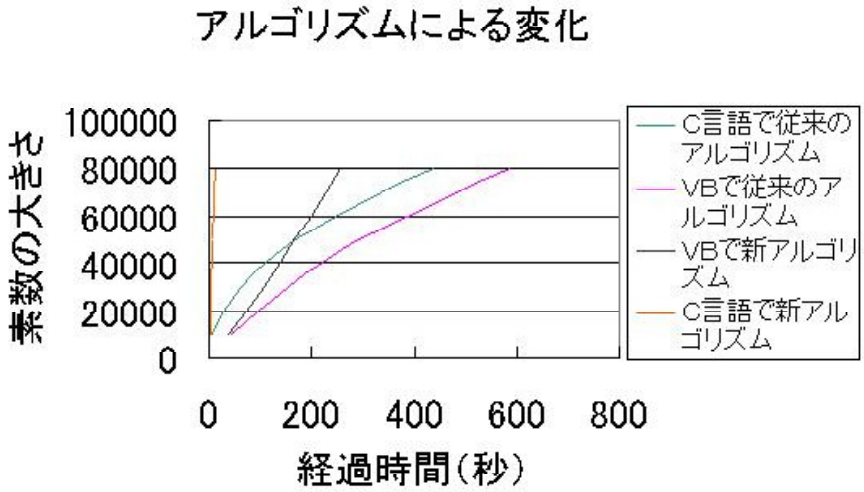
２：複数台による調査のための準備

ここで今までの製作してきたソフトの性能調査を、複数台接続して行うことにした。しかし、試作１の時にも述べたが、現在のアルゴリズムでは時間が経過するごとに素数を探索する速度は限りなく遅くなっていってしまう。つまり、このまま素数を探索したのでは、複数台で接続しても、それぞれの素数を探索する能力が限りなく下がっていく事によって、顕著な差というものが表れにくい（複数台での恩恵が表れにくい）と考えられる。そのため、ひとまず速度の低下を防ぐために少しアルゴリズムを考え直すことにした。

そこで、今までは自然数の定義どおり、ｎに対して、ｎ－１までの数で割り切れるか判定していたが、ｎに対してｎの平方根までの数で割り切ることができるか判定すれば良いという事から、そのようにアルゴリズムを切り替えた。そして、このアルゴリズムの利点は、



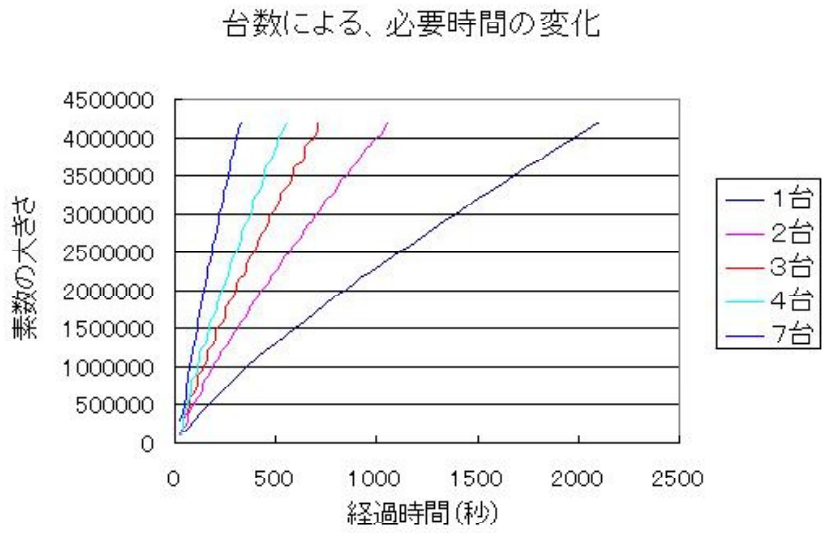
このグラフで分かるとおり、ｎが増加すると、√ｎの増加率は限りなく０に収束する、つまり今までのアルゴリズムに比べ、値が大きくなっても、そのための試行の回数はあまり増えないで済むという事である。その結果を、それぞれ比較すると、



といったようになり。Ｃ言語で新しいアルゴリズムを採用したものは圧倒的に早く、また、新しいアルゴリズムはいずれも、ほとんど速度の低下が起こらなかった（グラフがほぼ直線)。つまり、改良によって、複数台による分散処理の恩恵をより顕著に見ることができるようになったと言える。

３：複数台による調査

実際に複数台による分散処理を行い、その効率の違いを調査した。



４００万を計算し終えた時点での経過時間と１台の場合との比較

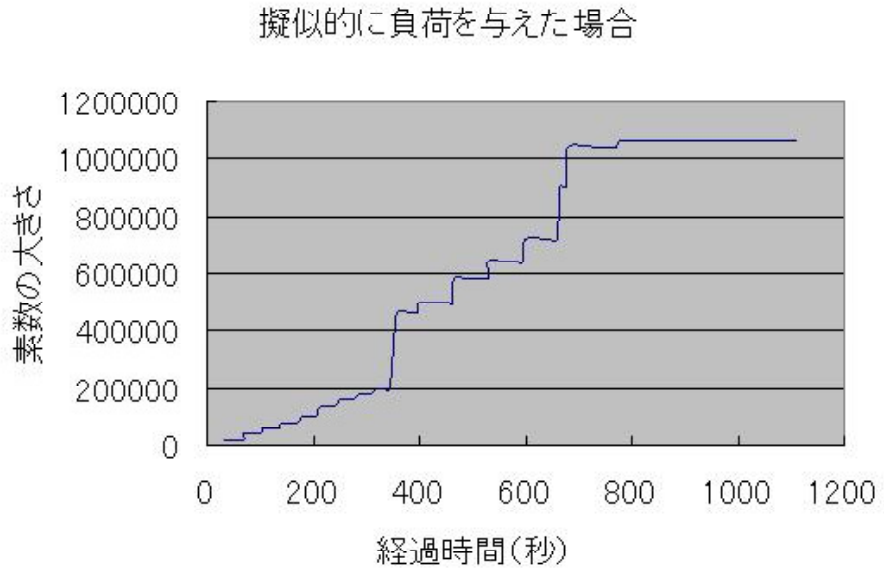
台数	経過時間（秒）	１台の時と比較しての％	効率
１台	１１４９．７２		
２台	６０１．８２	１９１％	９５．５％
３台	４２５．０７	２７０．３％	９０．１％
４台	３４２．６２	３３５．６％	８３．９％
７台	２７３．５３	４２０％	６０．０％

結果は、非常に興味深い結果となった。２台ではほぼ１００％の効率で分散する事ができているが、台数が増えるに従って、効率は下がっている。この原因は、最初にクライアントに計算する範囲を与える時に間隔を空けたため、全てのクライアントが一斉に計算を開始する事ができていないという事が一因になっていると考えられる。この部分は、今回性能調査のために、グラフのデータを収集する部分がシングルプロセスという理由からそうせざるおえなかったが、性能調査を行うのではなければ、間隔を空ける必要はない。またそれだけではなく、通信部分でのオーバーヘッドが増加してしまう事による低下も当然考えなくてはならない。

※この調査を行うに当たって、それぞれのクライアントが同時にデータを送り返してきてしまうとサーバのシングルプロセスの部分への負荷が強くなってしまいエラーが起こる恐れがあるので、最初に、クライアントに計算する範囲を与える時に、順にそれぞれ３０秒ずつ間隔を置いた。
※ここでは、クライアントに１回で計算させる範囲を１０万ずつとした。
※ここでの１台は、通信をせず、計算プログラムのみにより一台で計算を行った事を示す。
（以降明示しない限りは１台はこの事を指すとする）

４：クライアントの性能が異なる場合を想定

クライアントの性能がそれぞれ違う場合を想定し､擬似的に性能の違いを再現するため、最初にクライアントに１０～１の値を順に与えておき、計算結果をサーバに送信する時にその値×３０秒の間待機してから送信を行う事として調査した。



結果を見ると、今までのものとは明らかにグラフが異なる事が分かる。

このグラフにおいて、水平になっている区間は、データが返ってきたにもかかわらず、その素数の大きさまでのすべての範囲が探索し終わったわけではないことにより、その大きさまで探索終了とカウントされていない区間である。つまり、それぞれに異なる負荷を与えたために、先に計算を開始したクライアントより先に、後に計算を始めたクライアントの方が先に計算結果を返してきたことが原因なのである。

これまでは、それぞれクライアントは同程度の時間で計算を行っていたため、計算結果が返ってくる順序が逆になるということはなかったが、速度の違うパソコンを扱った場合にはこういった事象が生じる事も考慮するべきである。

※結果を顕著が表れるようにするため、台数を１０台、待機時間は最大５分とした。

第１０項　試作９「ファイルソートプログラム」

ここまではファイルに発見した素数をそのまま保存してきたが、それらはクライアントが不規則な順番で計算を完了して送り返してくるので、ファイルに書き込まれた素数はランダムな順序となっている。そのため、例えば、発見した素数のデータに従って、素数の判定を行うというように利用するにしても、二分探索法などを用いるためには昇順か降順に並べ変えられている必要がある。しかし、ＥＸＣＥＬなどでデータを並べ替えようとしても非常に時間がかかってしまう。それならば、その部分も自作すればよいと思い、製作にとりかかった。

☆製作過程

ファイルをソートするという処理は、意外と処理能力を必要とする作業で、Ｖｉｓｕａｌ　ｂａｓｉｃで並び替えるのでは、使い物にならないと思われた。そのため最初はＣ言語で作成することにした。

Ｉ：Ｃ言語でのファイルソートプログラム

Ｃ言語でファイルソートを行うに当たって考えた方法は

ファイルに書き込まれている素数を一つずつ読み込む

↓

それぞれを、配列に代入する

↓

配列に代入されたデータを標準関数であるｑｓｏｒｔ関数を用いて昇順に並び替える

↓

それらを、再びファイルに書き込む

というものであった。しかし、数万、数十万という要素に対して使用してみると、配列の宣言の所でメモリの容量が足りずこの方法は断念せざるおえなかった。（後になって、この問題はテンポラリファイルを用いて一時的にデータを場所を用意すれば回避する事ができると分かった）

☆Ｖｉｓｕａｌ　Ｂａｓｉｃでのファイルソートプログラム

上記のＣ言語を用いた方法では、一つ重要な事が忘れられていた、それはファイルの中の要素がまったくのランダムではないという事だ。、ファイルの中の素数は、一回の通信でクライアントから送られて来た素数の集合としては昇順に並んでいるのである。それならば、すべての要素を並べ変えるのではなく、それらの集合を単位として並べ替えればよかったのである。そこで、やはりできれば、プログラム本体の中に含めたいという理由からも、プログラムの流れを見直してＶｉｓｕａｌ　Ｂａｓｉｃで製作し直した。

新しい方法は、

ファイルからそれぞれの集合の先頭の値を取り出し、配列に代入する

↓

もう一つ用意しておいた配列に集合の値全部を代入する

↓

配列に代入しておいた、先頭の値を、自作のＦｏｒのループを用いた方法で並び替える

↓

並び替えられた先頭の値にしたがって、もう一つの配列に代入しておいた値を並び替える

並びかえられた値の集合を再びファイルに書き込む

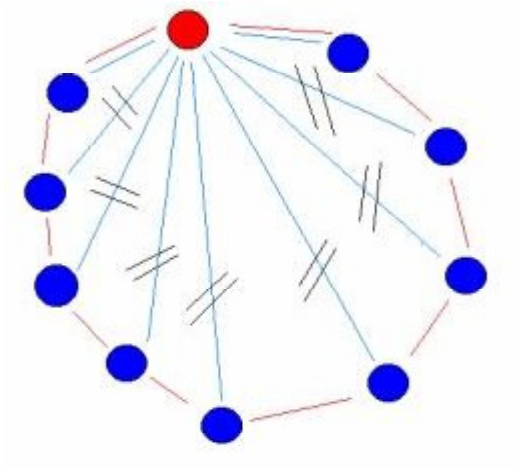
この方法によって、Ｖｉｓｕａｌ　Ｂａｓｉｃで製作した事による速度低下を考慮しても、こちらのプログラムの方が高速に動作する事ができた。また、用意すべき配列の要素数も大幅に少なくする事ができた。

第１１項　試作１０　「円状ネットワーク」

大抵の場合グリッドという技術の上では、ネットワークの形態として、一台のサーバに複数のクライアントが接続しているという、サーバ・クライアント型というネットワーク形態が主流である。この事は、通信で通過する経路を短くするという意味、つまり、通信部分でのオーバーヘッドを最小にするという事を考えれば当然の事だが、サーバの負荷はとても高くなってしまう。なぜなら、一台のサーバに対して複数のクライアントが同時に通信を開始してしまう状況が発生し得るからである。しかし、今回の私の手元にあるサーバは一般のパソコンであり、特別高い処理能力を持っているわけではない。また、今回の実験のように規模が小さいからこそ、普通では考えないようなネットワークを試す事が可能になる。

そこで、サーバの負荷軽減という観点から３つのネットワーク形態を試してみる事にする。

最初に試すのは、下の図ような円状のネットワーク形態である。



赤丸：サーバ
青丸：クライアント
赤線：実際のデータ送信用
水色線：補助用

このネットワークの特徴は、円状に接続を介して、一方向にデータ送信を行うという点である。そして、サーバにデータを送信する事ができる経路が、円状の部分からは一本しか存在しない。その事から、複数のクライアントから同時に要求を受けるという必要がなくなるという利点がある。しかしながら、サーバ・クライアント型であるならば一本の送信経路をデータが通過するだけで送信が完了するが、この場合、クライアントの台数に比例して通過する本数が増加するためオーバーヘッドが生じてしまうであろうと予想する事ができる。

☆製作の過程での問題点

まずはどういった手順でデータ送信を行うか、計算範囲を伝達するか、また、接続を介するかを考える必要があった。

ｉ：どうやって円状に接続を介するのか？

円状に接続を介するにあたって必要な情報は、接続を行うべき隣のクライアントのアドレスである。しかし、クライアント間でアドレスを取得する手段は存在せず、また、ネットワーク的に見れば隣にあるが、物理的にはまったく関係なく配置されたパソコンの中では、隣のクライアントがどのパソコンであるのかすらも判断する事ができない。そのため、クライアントが円状に接続を介するためにはサーバが仲介役を行わなくてはならない。

以上の理由から、まず最初に、各クライアントがサーバと接続を介するようにした（図で言えば水色線である）。そして、全てのクライアントが接続を完了した後、サーバは配列に保持しておいたクライアントのアドレスを用い、それぞれのクライアントに接続すべき対象のアドレスを送信する。それによって円状接続は完了することをできるようにした。

Ⅱ：計算範囲の伝達をどの様に行うか？

計算範囲の伝達を円状に行うという事は、サーバが伝達するのではなく、バケツリレーのように次のクライアントへ伝えて行けばサーバの送信が１回だけでも全てのクライアントにデータ範囲の伝達を行う事を意味する。また、計算する内容が一定の範囲によって区切られるという今回の計算の内容により、次のクライアントに伝達を行う時に、一定の値を加算して渡せば容易に実現する事ができた。また、円状なので最後にサーバへデータが返ってくることにより全てのクライアントに伝達が完了した事を確認する事ができる。

Ⅲ：データの送信をどうするか？

データの送信をサーバへ直接行わずに、次のクライアントへ送信しなくてはならないので、その点で新しいロジックを考えなければならなかった。

まず、計算範囲を得たクライアントは今までと同じように、Ｃ言語のＥＸＥを用いて計算を開始する。ここで、今回のプログラムではあえて計算部分をＤＬＬにして呼び出すという方法ではなく、Ｃ言語のＥＸＥで計算を行っているという処理方法により、計算中でもプログラムがホールドされてしまう事がないという利点があり、、イベントを受け取る事ができる。その結果、計算中でも、円の上で前にあたるクライアントからのデータ送信を処理が可能である。その利点を利用し、「データ送信許可」というアイデアによってこの問題を解決する事にした。

このアイデアは、それぞれのクライアントは、前のクライアントからの「データ送信許可」がなければデータの送信ができないというものである。そして、その許可は、前のクライアントの計算終了による、自分への計算データ到着によって得られる事とする（届いた計算データは自分の計算が終了した時に結合して送信する）。

このアイデアによって、上流から計算が終了するたびに、順にデータが結合されていき、最終的にはサーバへ円上の全てのクライアントの計算結果が返るというデータ送信が可能になる。意味合いは違うが、動きのイメージとしてはそれぞれの工程において少しずつ部品が付きながらベルトコンベアの上を流れて行き、最終的には巨大な車として完成するというようなものが近いであろう。

以上の手順経て、計算結果がサーバに返ったら、サーバは次の計算範囲をクライアントに割り振り、上記の手順を繰り返して行く。

以上のようにデータ送信の問題を解決した。

Ⅳ：諸問題の解決

- 円上において一番最初と最後のクライアントは他のクライアント処理が変わる必要があるので円状に接続を介する時にサーバからその事を伝達しておくことにした。
- １台のクライアントが３つの通信経路を持つので、Ｗｉｎｓｏｃｋコントロールを３つ用意し、それぞれ役割別に利用することとした。
- 必要がなくなったアウトプロセスのプロセスを終了させる事がプログラム上のエラーの原因となっており、円状に接続を介する事ができなかったが、そこの部分を修正することによって接続が完了するようになった。
- サーバがクライアントに他のクライアントへの接続命令を出し、接続完了のメッセージを受け取るという部分で、あまりにも接続の完了が早すぎるために、サーバが接続完了のメッセージを処理する事ができなかった。これを解決するために、多少接続が完了するのが遅くなってしまうが、接続完了の後、クライアントは一定時間待機してから接続完了のメッセージをサーバへ送信するようにした。

☆最終的なプログラムの流れ

全てのクライアントがサーバへ接続を行う（接続と同時にサーバから一度に計算する量などの情報も取得する）

↓

サーバは新しいプロセスで接続要求を受け、接続要求をして来たクライアントのアドレスを配列に順に代入していく

↓

全てのクライアントが接続を介したら、サーバの円状接続ボタンによって円状接続命令を開始する

↓

サーバは配列の添え字に従って、Ｎ番目にアドレスが格納されているクライアントに対して、Ｎ＋１番目に格納されているアドレスを送信する（接続完了が返ってきたら次のクライアントに移る）

↓

接続すべきアドレスを受けたクライアントは、受信したアドレスに従って接続を行い、接続完了をサーバへ送信する。（サーバは利用し終わった補助用の経路を切断）

↓

サーバの計算開始ボタンによって計算を開始させる（円の一番最初のクライアントに範囲の開始値を送信する）

↓

範囲を受け取ったクライアントは、その値に、事前に取得しておいた定められた値を加算し次のクライアントへ送信し、自分は計算を開始する

↓

円上を一周し、サーバへ計算範囲のデータが返ってきたら、計算範囲伝達完了を表示する。

↓

計算を終了させたクライアントは「データ送信許可」を得ている場合、次のクライアントへ計算結果を送信し、そうでなければ前のクライアントからの「データ送信許可」を待つ（計算を終了して待機している状態で、「データ送信許可」を得た場合をすぐに送信を開始する）

↓

サーバへ計算結果が到達したら、それをテキストファイルに書き込み、次の計算範囲を送信する。

☆考察

実際に製作を行っていく上で分かった長所と短所を挙げる。

長所

・サーバは大きなデータを受け取るという意味での負荷はかかるが、同時に複数のクライアントからの要求に答える必要がないので、要求の取りこぼしが無く、サーバの働きに信頼が持てる

・円の上流から順に計算結果が結合されて送られてくるので、サーバ側で受信データをソートしなくても、すでにソートされた状態になっている。

短所

・計算を終了し、計算結果を次のクライアントに送信してしまうと、そのクライアンより後のクライアント全てが計算を完了させないと次の計算範囲を得る事ができない。また、計算を終了したとしても、自分のところまでデータ送信許可が来なければ、待機していなくてはならない。つまり、どうしても一番処理能力の低いパソコンに合わせて計算を行わなくてはならなくなってしまう。この事はつまり、計算結果がサーバに到達するというポイントを境に、バリア同期を行っているのと同じような状況が生まれてしまう。

・単純に考えても、クライアントの数に比例して通過すべき通信経路が増えてしまうのに加え、クライアントのデータが結合されていく事によりデータは肥大化し、下流のクライアントになるにしたがって、通信に負荷がかかってしまう。

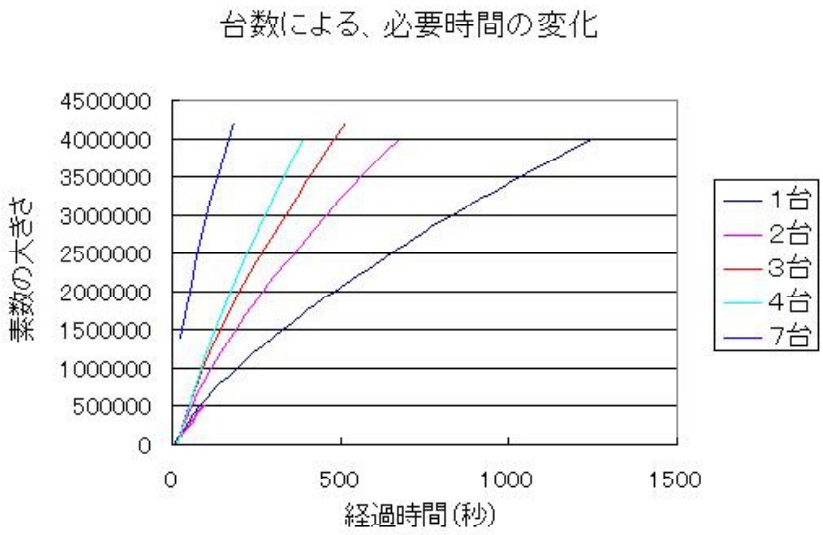
・データが最終的にサーバに届くまでの間、クライアントが何らかのアクシデントでストップしてしまった場合。ネットワークは成り立たなくなってしまう。

☆性能調査

上記の考察を踏まえ性能調査を行った。

条件は試作８の性能調査の時と同じ条件で行った。

※サーバから直接計算範囲を送信するわけではなく、また、サーバに要求が集中することもないので、故意に計算開始の時間をずらしたりすることは、ここではしなかった。



４００万を計算し終えた時点での経過時間と１台の場合との比較

台数	経過時間（秒）	１台の時と比較しての％	効率
１台	１１４９．７２		
２台	６７３．０５	１７０．８％	８５．４％
３台	５１４．７７	２２３．２％	７４．４％
４台	３９１．５６	２９３．６％	７３．４％
７台	２８２．４３	４０６．７％	５８．１％

調査の結果、以上のようなデータが得られた。

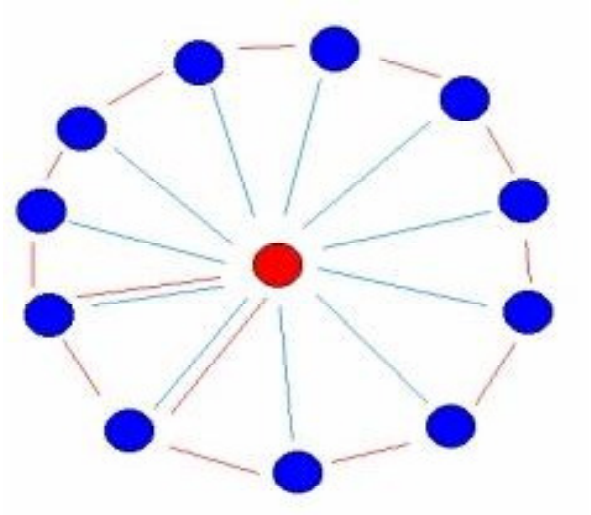
試作８の結果と比較してみると、グラフの形は大差ないものの、やはり効率の値では通信のオーバーヘッドのせいかこちらの方が低くなっているようである。

また､当然のように台数が増えるに従って急な変化率で下がっていってしまっている(７台の時点で試作８とあまり大差がないのは、試作８では、それぞれのクライアントの計算開始の間隔を開けたことから、大幅に低下してしまったからであろうと予想できる。つまり、それがなければ試作８はもっと高い効率を誇っていたであろう)

第１２項　試作１１　「円状ネットワーク改良版」

試作１０の「円状ネットワーク」はサーバの負荷軽減という観点からは利用価値があるが、それが実際に利用するに値するかどうかはには、疑問符がつかざるおえない。そのた

め、円状ネットワークというアイデアを発展させ、利用可能なレベルまで改良すべきだと思い、効率を上げる方法を考えた。



赤丸：サーバ
青丸：クライアント
赤線：データ送信許可伝達用
水色線：計算結果送信用

このネットワークは試作１０「円状ネットワーク」の”計算結果の送信の時に本来ならば１つの経路の通過だけで済むのに、クライアント数に比例した経路を通過せねばならない”という部分を改良するために考えたものである。具体的にはサーバを円の上から切り離し、試作１０では補助用であった通信経路（上図では水色線）を利用して計算結果送信を行おうという事である。つまり、計算結果送信の経路をそれぞれに用意し、クライアント間で折り合いをつけ、順にデータの送信を行う事によって、同時に計算結果を送信してしまうという状況を無くせばよいのである。

具体的な処理は、赤線の経路を用いて、試作１０と同じように「データ送信許可」を順に次のクライアントに送信していくのだが、計算結果の送信は次のクライアントへ行うのではなく、直接サーバへ送信するという所が今までと異なった。

☆製作過程

ほとんどの問題は試作１０の内容の流用により解決してしまったので、特に挙げるべき問題解決は存在しなかった。

☆最終的なプログラムの流れ

試作１０と同じように円状に接続を介する。（補助用の経路を切断しない）
↓
試作１０と同じように、クライアントに計算範囲を割り振る
↓
計算範囲を受けたクライアントは計算を開始し、終了した場合「データ送信許可」を得ていればサーバへ計算結果送信、次のクライアントへの「データ送信許可」を行う
そうでなければ、待機する
↓

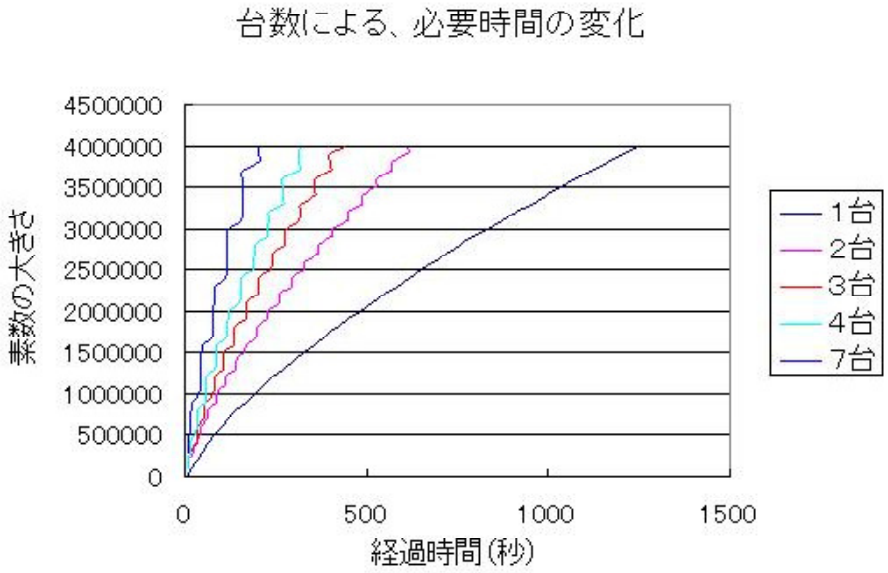
以降も試作１０に同じ

☆考察

長所・短所共に試作１０とほぼ同じであるが、計算結果を送信する部分でのオーバーヘッドは解決する事ができた。

☆性能調査

試作８，１０に引き続き同じ条件で性能調査を行った。



４００万を計算し終えた時点での経過時間と１台の場合との比較

台数	経過時間（秒）	１台の時と比較しての％	効率
１台	１１４９．７２		
２台	６０１．８２	１９１％	９５．５％
３台	４２５．０７	２７０．０％	９０．１％
４台	３４２．６２	３３５．６％	８３．９％
７台	２７３．５３	４２４．２％	６０．０％

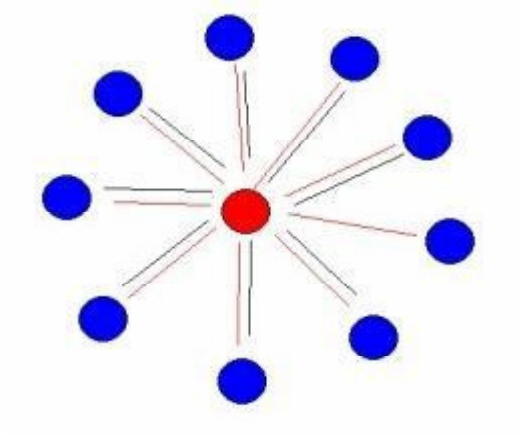
このグラフをみてまず気づく点はグラフが波線のようになっているという点である（１台での結果は通信を介していないので例外)、この現象はこの調査だけで見られた独特の形であった。

次に実際に効率は改善されたのか試作１０の結果と比較してみると、４台までの結果に対しては１０％程度程向上した事がわかる。しかしながら、７台まで台数が増加すると、どちらも６０％前後の値になってしまった。このことは、試作１０の通信部分でのオーバーヘッドの増加を考えると、本来ならば効率の差が増加していくはずなのだがそうはならなかったようである。この事は、この程度のデータ量では、データ送信のオーバーヘッド

よりも、プログラムの中でのロジックによるオーバーヘッドの影響の方が大きいという事を意味している。

第１３項　試作１２「逆方向サーバ・クライアント型」

ここまで２つの新しいネットワーク形態を考えて来たが、独創的なアイデアなだけに、なかなかサーバの負荷軽減と計算効率の両立をさせることは難しいようである。そこで、最初の形にたちかえり、サーバ・クライアント型での解決について考えてみることにした。
※赤線と黒線は便宜上２本となっているが実際には１本の通信経路である。



赤丸：サーバ
青丸：クライアント
赤線：計算範囲の送信と
計算終了の確認用
黒線：計算結果送信用

サーバ・クライアント型で今までと違う方法を考えて行くうちに、計算結果をクライアントが自発的に送信するのではなく、サーバ側からの指示によって行われるようにすれば、サーバ側からデータ送信を制御する事ができるのではないかという事に気づき、それを実現する方法を考えた。

☆製作の過程での問題点

ｉ：サーバ側からの指示によって計算結果送信を行わせるにはどうすればよいか？

サーバ側から結果の送信を行えという命令を行ったとしても、実際にクライアントが計算を終了させていなければそれは意味を持たない。しかし、クライアントからの通知が無い限りはサーバは計算の終了を把握する事もできない。よって、サーバはクライアントにデータ送信の命令を直接出してしまうのではなく、まずは、計算が終了したかをクライアントに確認し、クライアントは計算が終了している場合だけ計算結果を送信するという手順を踏まなくてはならない。そして、計算結果を受信した時に限り、クライアントへ新しい計算範囲を与えるのである。

次に、サーバがどういった順序で計算終了を確認していけばよいか考えなくてはならない。本来であれば、サーバがすでに計算を終了している確率が一番高いであろうと予想するクライアント（他のクライアントは結果を返したが、まだ、計算に時間がかかるというクライアント）に対して集中的に確認を行った方が効率的なのだが、1 回の確認にそれ

ほど時間を要しないので、多くても１０台という条件の中では、そういった判断を行った方がオーバーヘッドが増加してしまうであろうと予想した。よって、ここでは最も単純に、配列に保持しておいたアドレスの、添え字の若い方から順に確認していき、配列の末端に達した場合は、先頭に戻る、という手順によって順に確認していくという方法をとった。

Ⅱ：諸問題の解決

・配列に保持しているアドレスに従って確認を行うために、Ｄｏ～Ｌｏｏｐ文を用い、確認を行っている間、結果が返ってくるまで空ループをさせ、結果が返ってきたら次のクライアントへ移るというようにした。

☆考察

長所

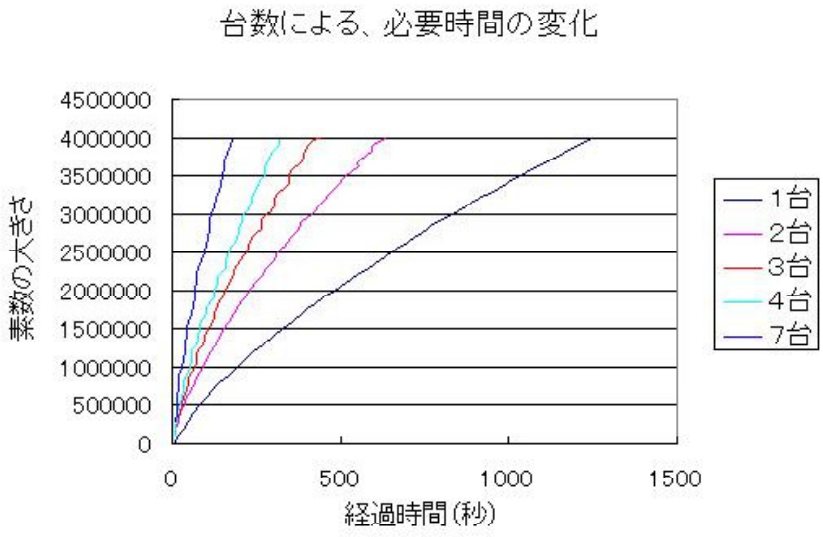
・１０台程度であれば全てのクライアントを１回確認するのにそれほど時間がかからないため、送信待機状態になる時間はあまり考えずに済む。
・今までのようにある点でバリア同期をとらなくてはならないということが無いため一台のクライアントの処理能力が低くても他に影響を与えない。
・複数のクライアントから同時に処理を求められる事が無い。

短所

・あまりにも大規模なネットワークになった場合、計算終了後に待機をせざるおえないクライアントが増大する。
・サーバはループで確認をし続けるので、同時に処理を求められる事はないが、高い負荷が継続的にかかってしまう。

☆性能調査

以上を踏まえ再び性能調査を行った。



４００万を計算し終えた時点での経過時間と１台の場合との比較

台数	経過時間（秒）	１台の時と比較しての％	効率
１台	１１４９．７２		
２台	６１４．１３	１８７．２％	９３．６％
３台	４４５．４５	２７０．３％	９０．１％
４台	３１６．　１	３４４％	８６．０％
７台	２０５．０９	５６０％	８０．０％

まず、グラフを見ると、試作１１ほど顕著ではないが、多少の波形のようなものが見られる。これはサーバの確認というステップによって、計算結果の送信を行っているため、ランダムな要素が介入せざるおえないからであろうと予想できる。

また、効率を見ると、４台までは今までとあまり大差ないが、７台では他の６０％前後と比べ２０％もの改善が見られる。この意義は非常に大きいものであり、このレベルであれば現実的に十分利用可能であると思われる。

☆試作８、１０，１１，１２の性能調査を終えての考察

試作８では、クライアントの計算の間に間隔をおいてしまっているので、単純に比較を行う事はできないのだが、おそらく、それがなければ一般的な形態である試作８が最も高い効率を得る事ができるであろうと思う。しかしながら、他の３つの形態を試した結果、いくつかの改善によって利用可能なレベルまで効率を上げる事ができた。よって、サーバの負荷軽減という視点を重視するのであれば、試作８の他の形態を利用する意義は十分にあるであろうと思う。しかしながら、利用するクライアントの台数において、今回のような小規模な場合ならば利用してもよいかもしれないが、もっと大規模になった場合は一考を要するという点に注意しなければならない。

第１４項　パソコン室での素数発見プロジェクトを終えての考察

以上の研究では、ネットワークによる効率の変化、また、ネットワーク構築の技術などについて主に扱ってきたが、実際に素数発見という点で意味があったかと言えば、それはあまりないと言わざるおえないであろうと思う。素数発見のアルゴリズムを改善すれば、１億程度までの素数表も１台のパソコンでほんの数十秒あれば作成する事が可能である。よって、素数発見という事を題材にはしたが、振り返ってみるとネットワーク構築についての調査であったという事になる。

そこで、素数発見という題材は「ボランティアコンピューティング的方向性」の方で引き続き研究を続け、リアルタイムに処理能力を得られるパソコン室は数値計算という方向性での利用を考えていくこととした。

第３節　分散型多倍長行列計算ソフトウェアの製作

第１項　製作動機

第２節で素数発見を題材に製作を行ったが、パソコン室のリソースを有効に利用するのであれば、リアルタイムに処理能力を得る事ができる事を生かすべきである。一方で、素数発見という題材は、各要素に依存関係が存在しないという特徴を考えると、ボランティアコンピューティングという方向性でこそ有効に計算を行う事ができるので、一度研究を休止しておくことにした。そのため、ここではテーマを一時変え、まず、パソコン室でどのような数値計算を行うべきかと考えた。

そこで、以前自宅で行列計算を行おうとした時に、簡単に使えるが、それなりに本格的な計算ができるというソフトウェアが提供されていなかった事を思い出した。本格的に行うのであれば、数値計算用の高度な物などが提供されているが、使い方が非常に難解であり、また、大抵の場合コンソールアプリであるのでユーザーインターフェースがあまり良くない。逆に、簡単に使えるという物では、教育用と書かれたような物しかなく、扱えるのはせいぜい５×５行列程度までであった。

以上より、簡単にある程度本格的な計算を行える行列計算ソフトでかつ、多倍長と分散というアイデアを組み合わせて製作を行う事に決めた。

第２項　多倍長ルーチンの製作

プログラミング言語で扱える値は、特に工夫しない限り６４ビットという制限から、たかだか２の６４乗（18446744073709551616）までである、この事は、Ｃ言語でもＶｉｓｕａｌ　Ｂａｓｉｃであろうと例外ではない。しかし、プログラムで工夫をすれば、何桁であろうと利用可能にする事ができる。それを実現するのが、多倍長演算のルーチンという物である。

多倍長演算のルーチンは、数値を単独で扱うのではなく、配列に分割して保持し数列とみなすというアイデアの上に成り立っており、その内容は、筆算の方法をプログラムで再現しているという単純な物である。（除算・剰余算などは例外、また、筆算を再現するだけではなく、内部で特殊な処理を行って演算を高速化している物も存在する）

このような多倍長ルーチンをプログラムで利用するため、ＤＬＬや関数として提供されているルーチンをインターネット上で探したがＶｉｓｕａｌ　Ｂａｓｉｃで利用できる物は提供されておらず、また、Ｃ言語用のものなどでも、専用の構造体などを用いた高度なものばかりで、文字列を渡せば計算ができるというような簡単に利用できるものは存在しなかったので、利用を見合わせざるおえなかった。

以上の理由により、まずルーチンを自作する事になった。

☆製作過程

まずは加算・減算・乗算の３つを製作し、仕様としては、２つの数値を文字列として渡し計算結果を文字列として返すというようにした。

ｉ：加減算を行うにはどうすればよいか？（整数）

試行錯誤の後、決定した手順は下のようである。

文字列の符号を取り除き絶対値を比較する

↓

符号と絶対値の関係により、絶対値を足し合わせるか、引くか決定する

↓

文字列を数値として配列に代入していく

↓

配列の要素をそれぞれ同じ桁同士の部分で演算する

↓

演算で、繰り上がり、または繰り下がりが生じた場合には
上の桁の演算の時に１加算か１減算する

↓

計算結果を代入した配列の要素を文字列として結合し、符号も結合し出力する。

Ⅱ：乗算を行うにはどうすればよいか？（整数）

乗算も筆算と同じ考え方で再現する事ができた。

※加減算の場合と一致する部分は省略する。

※A[],B[],を乗算しC[]に代入するとする

符号を取り除き、符号が一致する場合は＋、しない場合はーを記憶しておく

↓

筆算と同じように A[1]× B[1],B[2],B[3]....、A[2]× B[1],B[2],B[3]....と計算し、適当なC[]の要素に代入・加算していく

↓

C[]の要素をすべて結合し、符号も結合して出力する.。

Ⅲ：加算、減算、乗算で小数を扱えるようにするにはどうすればよいか

行列の演算では当然の様に小数の値も扱えなくてはならない、そのため、小数を扱えるようにルーチンを改善する必要があった。

この問題は乗算に対しては容易に解決する事ができた。乗算では、それぞれ小数点を取り除いてその位置を保持しておき、Ⅱのように乗算を行った後、保持しておいた２つの位置を加算した位置に小数点を置けばよいだけである。しかし、加減算で小数が扱えるようにするにはいくぶんの手間がかかった。その手順は、

２つの数値の小数点を取り除き、その位置を保持しておく

↓

小数点以下の桁数を一致させるため、その数を比較し、足りない場合には０を埋める

↓

Ⅰと同じように加減算を行う

↓

小数点を加え、小数点以下の末尾に０が存在する場合には取り除き出力する

という方法であった。

Ⅳ：除算を行うにはどうすればよいか？

加算、減算、乗算は筆算の再現を行えば、ある程度簡単にプログラムを組む事ができたが、除算は特別な方法を考えなければならない。その理由は、私たちが行う筆算での除算は、どの程度の値をかけて引けば、繰り下がる値が最小になるかという予想を伴いながら行われているからである。この予想をプログラムとして再現するには、１倍、２倍・・・９倍と順に倍数を求めて、割られる数を超えないかたしかめるかめるというほぼ総当りに近い手順をとらなければならない。これでは、いくら除算とはいえ、あまりに時間がかかり過ぎてしまう。そのため、筆算での除算とは違う方法を考えなければならない。

そこで、Ｎｅｗｔｏｎ－Ｒａｐｈｓｏｎ法を用いた近似計算によって除算を行うというアイデアを利用する事にした。

このアイデアは、Ａ÷Ｂの計算を、Ａ×（１/Ｂ）と考え、１/ＢをＮｅｗｔｏｎ－Ｒａ

ｐｈｓｏｎ法による近似値として求める事によって計算可能にするというものである。

ここでＮｅｗｔｏｎ－Ｒａｐｈｓｏｎ法による近似計算は、Ｂの逆数の１/Ｂを求めるために

$$A_{n+1}=2\times A_n-B\times (A_n)^2$$

という数列を適当な回数繰り返し（繰り返すごとに正確な近似値が求められる)、その結果、Ａで１/Ｂの近似値を得ることができるというものである。しかしながら、Ａの初期値を適切に選ばないと繰り返した結果は近似値へ向かって行かないという問題があった。だが、その初期値の選び方の情報は見つからず、その問題をまず解決せねばならなかった。

Ⅳ－１　Ｎｅｗｔｏｎ－Ｒａｐｈｓｏｎ法のための初期値をどう得るか？

初期値を得るという事は１/Ａ を求めるのであれば、１/Ａの値がおおまかにわかれば良いという事である。そして何度か試した結果、ここでの初期値はそれほどの精度がなくても収束する事が分かった。しかし、Ａは多倍長で計算しなければ扱えない値であるので、１/Ａを求めると言ってもそのまま計算する事はできない。そこで、Ａという数の最上位の一ケタを用いて逆数を計算し、その逆数を本来のＡの逆数であればなるであろう桁数に調整するという方法を考えた。

例えば、**Ａ＝５４３８２９７５９３２４５８９４３５７３４６２３４０４３**である時、

まず、最上位桁の数である５を用いて１÷５＝０．２と計算し、Ａの桁数から算出して、０．０００００００００００００００００００００００００００００２という近似値を得るという事である。

以上より、除算をするための流れは

Ｘ と Ｙ という値を受け取る

↓

乗算と同じように符号を取り除き、その情報を保持しておく

↓

Ｙ の最上位桁と Ｙ の桁数から初期値A_nを決定する

↓

$A_{n+1}=2\times A_n-Y\times (A_n)^2$ という数列を、多倍長の減算と乗算の関数を用いて適当な回数繰り返し、１/Ｙ の近似値を得る

↓

Ｘ ×（１/Ｙ）を計算し、符号を結合して出力する

となった。

Ⅴ：諸問題の解決

・乗除算において無駄な演算を避けるために、１で乗除を行う場合などは計算をせずに返すようにし、また、多倍長でなくても可能な演算はそのまま計算して返すように工夫した。

☆考察

今回自作したルーチンは、文字列を渡しただけで計算ができるような非常に使いやすいルーチンではあるが、内部で文字列を配列に分割するなどの処理が必要になるので、速度という面で見ればそれほど高速とは言えない。高速化を考えるのであれば、文字列ではなくバイト配列などの独自のデータ形態で値を保持する事にし、それを計算できるルーチンという形で作成した方が高速化が可能となる。

第３項　試作１「通常桁行列計算ソフト」

多倍長のルーチンが作成できた所で、まずは通常桁を扱う行列計算ソフトから作成した。

☆製作過程

まずアプリケーションのユーザーインターフェイス＆機能とプログラムの方向性を決める事にした。

〇プログラムの方向性＝シンプルな作りであるが、本格的な計算も十分可能。校内のリソースを活用するために製作する

〇ユーザーインターフェイス＆機能

・除算以外の四則演算が式を作る事によって可能
・代入モードと計算モードの２つが切り替え可能
・行列ボタンに行列データを記憶させておく事ができる
・逆行列を求める事ができる
・多倍長演算と分散処理が可能　　とした

Ｉ：行列をどうやって表示・入力するか？

まず、行列をどのように表示・入力するのか考えなければならなかった。

そこで、最初に何の考えなしにテキストボックスを多数用いて表示するという方法を考えたが、動的なオブジェクトの追加を行っていくと、１００個程度でメモリが足りなくなってしまい、実用には程遠い数しか用意する事ができず、また、スクロールなどもできないので、画面に表示できない程の要素数だった場合扱う事すらできないという事態となった。

以上の問題から行列表示に適するコントロールを模索していると、ＭＳＦｌｅｘｇｒｉｄコントロールが適当であると気づいた。このコントロールは本来、表作成などに用いるが、行列を扱うのに十分な機能をたずさえている。よって、このコントロールを利用する事とした。しかしながら、ＭＳＦｌｅｘｇｒｉｄコントロールを用いて製作を進めていくと、エクセルのようにクリックしたセルがアクティブになって入力できるような機能は用意されていないという問題が出てきた。表示はできるのだが、入力はできないのである。

入力が出来ないというのは盲点であった、当然できるものと考えていたからだ。そこで、ウェブで知った「テキストボックスをセルの部分に移動する事によって、あたかもエクセルのように、入力が可能になるアイデア」を取り入れ、入力も可能とすることができた。

Ⅱ：行列データをどのように保持するか？

行列の要素に値を与え、１つの行列ができたとしても、それを何らかの形で保持しておかないと計算式を作成する時に利用できない。また、そのたびに同じものを入力するのではあまりに不便であると言わざるおえない。そこで、Ａ～Ｋの行列ボタンというボタンを用意しておき、それをクリックする事によって、入力しておいた行列データをそっくりそのまま保持しておけるようにした。また、何行何列行列かという情報なども同時に保持しておけるように工夫した。

Ⅲ：計算式をどのように評価するか？

式を入力する事によって計算ができるようにすると仕様を決定したが、計算式の評価というのは非常に煩雑な手順や配慮が必要となる。

まず、計算式の入力は行列ボタンと計算記号ボタンによってしか入力できないように制限した。その理由は、キーボードからの入力を可能にすると、誤った計算式の入力を防ぐような制御が非常に困難になるからである。しかしボタンによる入力だけにすれば、フラグを立てるなどの方法で、計算記号が２つ並ぶ、計算式の最後が計算記号になってしまう、などの誤った計算式の入力を防ぐ事ができる。

問題である計算式の評価であるが、計算式作成用のボタンを用いて計算式を作成しておけば、計算式用テキストボックスに対応する計算式が文字列で表示されているので、その文字列を１文字ずつ切り出した情報から、それが行列名なのか、また計算記号なのかを比較判断し、計算記号に対応した計算関数を両隣の行列名を引数で呼び出す事によって可能とした。加えて、当然ながら計算式が何行にも続く事があるので、計算結果を保持する行列も用意し、次の関数呼び出しの時には、１個目の引数である行列名を前回の計算結果にする事で、２つ以上の行列を用いた計算も可能とすることができた。

例　　Ａ×Ｂ＋Ｃ－Ａという式があったとする

１：まず、左から１文字切り出し、Ａを現在の計算途中のデータ（ＷＯＲＫ）とする。
２：次に、２文字切り出す事によって、”×Ｂ”という演算を行えばよいので、ＡとＢを引数として行列乗算の関数を呼び出し、結果をＷＯＲＫに代入する
３：再び２文字切り出し、”＋Ｃ”を得るので、同じようにＷＯＲＫとＣを引数として、行列加算の関数を呼び出し、結果をＷＯＲＫに代入する。
４：上記と同じ手順を繰り返す

以上のような手順で計算式の評価ができると考えたが、大事な事が忘れられていた。計算式には結合規則という要素があるという事である。つまり、例での式がＡ×Ｂ＋Ｃ×Ａだったとしたら、上の手順では誤った回答を導いてしまう。この結合規則の問題を解決するのに非常に手がかかった。

Ⅲ－１　結合規則という問題をどう解決するか？

結合規則という問題を解決するという事は、つまり、多項式の中でのそれぞれの項を個

別に求めておき、その後にそれぞれを式に従って加減算するという手順を可能にすればよいのである。そこで、**例**での手順の改良を考えた。その手順とは、例でのＷＯＲＫに複数の行列データを保持できるようにして、次のように行う。

例２　　Ａ＋Ａ×Ｂ＋Ｃ×Ａという式があったとする

１：まず、左から２文字切り出すが、”×”という記号が存在しないため次に進む
２：また２文字切り出すと、”Ａ×”で”×”記号が存在するので、もう１文字切り出し、”Ｂ”を得るので、Ａ、Ｂを引数に行列乗算の関数を呼び出し、ＷＯＲＫ[1]に結果を代入し、式の”Ａ×Ｂ”の部分を置き換えて、**”Ａ＋Ｗ１＋Ｃ×Ａ”**とする
３：”Ｃ×Ａ”も同様にＷＯＲＫ[2]に代入し、式を置き換え**”Ａ＋Ｗ１＋Ｗ２”**とする。
４：その後は例と同じように、３つの項の計算を行う
５：結果を出力する

以上の手順が可能となるようにプログラムを組み、結合規則の問題を解決した。

Ⅳ：逆行列をどのようにして求めるか？

２次正方行列以外の正方行列の逆行列を機械的に求めるには、高校の数学Ｃで学ぶガウス・ジョルダン法（俗に掃出し法と言う）を発展的に用いる事によって可能となる。

掃出し法についての詳しく説明は省略するが、逆行列を求める方法を簡単に説明すると

$$\begin{pmatrix} 94 & 54 & 64 \\ 14 & 67 & 83 \\ 23 & 75 & 92 \end{pmatrix} \rightarrow \begin{pmatrix} 94 & 54 & 64 & 1 & 0 & 0 \\ 14 & 67 & 83 & 0 & 1 & 0 \\ 23 & 75 & 92 & 0 & 0 & 1 \end{pmatrix}$$ と置き、

$$\begin{pmatrix} 1 & 0 & 0 & a & b & c \\ 0 & 1 & 0 & d & e & f \\ 0 & 0 & 1 & g & h & i \end{pmatrix}$$ と掃き出し $$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$ を得る

という方法である。

このガウス・ジョルダン法による逆行列を求める手順をそのままプログラムにする事で逆行列を求める関数を作成した。

Ⅵ：諸問題の解決

・要素数が増えると１つ１つ値を与えるのも手間がかかってしまうので、単位行列と乱数ボタンを作成し、調査をスムーズに行えるようにした。
・行列の要素の桁数が大きくなると、そのセルをクリックしてアクティブにし、手動で

スクロールして値を確認するという手間がかかってしまい､非常に利用しにくかった。それを改善するため、セルの上にマウスを置くとポップアップを出すようにして、値を簡単に確認できるようにし、また、何行何列の要素かもわかりやすく表示するようにした。

・数値を仮数部と指数部によって扱うように改良し、有効桁以上の無駄な数値を保持したり、計算する事のロスを無くした

第４項　試作２「多倍長行列計算ソフト」

本節第２項で製作した多倍長ルーチンを試作１に加え、多倍長計算を可能にする。

☆製作過程

まず、単純に試作１での行列加減算・行列乗算・逆行列計算の関数中の演算を、多倍長のルーチンで置き換え、多倍長計算を可能にした。そこで生じた問題点は、多倍長でなくても計算できる場合まで多倍長で計算するのでは非常に能率が悪いという点である。

その問題を解決するため、新たに多倍長モードと通常モードという区別をつくり、ユーザーが選択して計算を行えるようにした。

以上の改善によって、ユーザーの意志によって、多倍長演算が可能となったのだが、Ｖｉｓｕａｌ　Ｂａｓｉｃでルーチンを用いるのではあまりに時間がかかり過ぎ、とても利用可能なレベルとは言えなかった。

そのため、Ｖｉｓｕａｌ　Ｂａｓｉｃのルーチンを Ｃ言語で書き直し、今までと同じ手法でそのプログラムを利用することにより、多倍長演算を行う事にした。

Ⅰ：Ｃ言語で書き直す上での問題解決

・除算において、内部有効桁を定めなければ、近似値を求める時に、利用しない桁数まで近似値を求めてしまい、無駄な計算が増加することに気づいた。そのため与えた数の桁数から内部有効桁を求めて計算するように改善した。

・加減算を同じ関数で計算するので、減算を行う時は２個目の引数に"－"を結合してマイナスの値にしてから加減算の関数を呼び出すのだが､”－”の符号を結合する前からすでに符号がマイナスだった場合、符号が”－－”となってしまう。この部分の処理が抜けていたため予期せぬ計算結果が出てしまう事があった。そのため符号が”－－”の場合には、２つとも取り除きプラスとして扱うように改善した。(他の乗算、除算などでも同じように改善を行った)。

Ⅱ：Ｃ言語のプログラム呼び出し上での問題

Ｓｈｅｌｌ関数で呼び出したＣ言語のプログラムによって計算を行うという今までの手法を今回も用いようとしたが、今までと違い、プログラムの呼び出しの頻度が非常に高い高いためパソコンに負担を強いり、パソコンが不安定になってしまう状況が確認された。また、プログラム呼び出しのオーバーヘッドも生じてしまう。そのため、一般の人々でも

利用可能というテーマから、極力難しい技術は用いたくなかったのだが、Ｖｉｓｕａｌ Ｃ＋＋を用いてＤＬＬとして作成し、ＡＰＩ関数として呼び出す事にした。

Ⅱ－１　ＤＬＬ作成上での問題

・ＤＬＬ作成の上で、文字列を渡し、文字列を返すという処理は、非常に難解な技術が必要という事が分かったので、ここではそれを避けるため、引き続きデータの受け渡しにはファイルを介して行うようにした。
・ＤＬＬを作成したが、動作せず、原因が何かと試行錯誤した結果、ＤＬＬが戻り値を呼び出し元に返すように変更することによって動作するようになった。ファイルを介するという都合から、盲点となってしまっていたのである。

Ｃ言語のプログラムをＤＬＬとして呼び出すようにした結果、パソコンが不安定になる事もなくなり、呼び出しのオーバーヘッドが減少したため速度も向上した。

第５項　試作３　「分散型行列計算ソフト」

試作２までの製作によって、逐次で行列計算を行うソフトは完成した。次に、このソフトを当初の目的である分散型にする方法を考えていく。

そこで、まずは、このプログラムの中でどの処理を分散させて計算させるか決めなくてはならなかった。

☆製作の過程での問題点

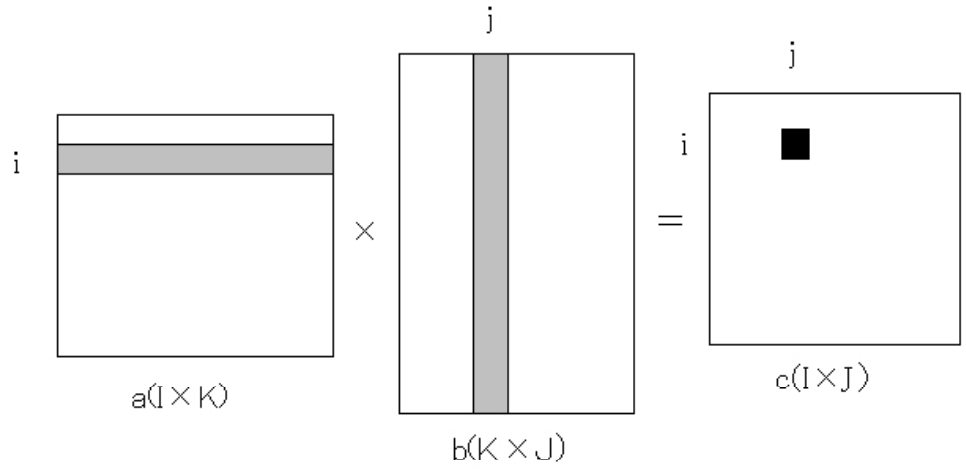
Ⅰ：どの処理をどの程度の粒度で分散させるか？

行列の加算・減算は１００×１００行列程の大きな行列であっても、たかだか１００００ステップ程で計算が可能であり、あまり分散する効果というものは得られず、オーバーヘッドのために効率が低下してしまうであろうと予想した。そのため、ここでは乗算と逆行列を求める部分を分散させる事にした。

分散を考える上では、どの程度、計算をくだいて分散させるかという粒度なども、非常に重要な要素となる。そのため、次にどの程度の粒度で分散を行うかという事を決めた。

今回の場合では、おおきな粒度であれば、式の中の項を単位に分散し、最後の加算をサーバで行うという選択があり、小さな粒度であれば、行列計算の中の加減乗除の単位で分散するという選択もある。しかし、これでは両極端過ぎでベストのパフォマンスは望めない。(項の単位で分散するというのは速度は速いかもしれないが、台数と項数の都合が合わなければ利用する事ができない)、以上の理由を加味し、検討した結果、Ａ×Ｂなどの単項式でも分散でき、その中では粒度が高いため比較的高いパフォーマンスが期待できることから、基本的な単位としては、行単位、列単位で分散を行う事にした。

Ⅱ：乗算の分散をどのように行うか？



乗算の分散は比較的容易であった。

行列の乗算というのは、計算結果の（Ｉ，Ｊ）成分があった時、それは１通りの行ベクトルと列ベクトルの組み合わせによって求める事ができ、その他の情報や値を必要としない。そのため他の部分の計算結果などに依存する事がない。よって、行ベクトルと列ベクトルをクライアントに与えれば、計算の順序や、計算結果が戻ってくる順序などに配慮せずとも分散を行う事ができるのである。

ここで再び粒度について考える。計算結果の行列Ｃの要素数はＩ×ｊであるので、単純にクライアントの台数で割った場合、１台のクライアントが計算する要素数は（Ｉ×Ｊ）/クライアントの台数、という事になる。そういった分散を行うと、それぞれのクライアントが計算する範囲がＣにおいて、行の途中で終わってしまうというようになる。また、要素数が割り切れない場合もある。こうなると、プログラミングが非常に難解になり、また、そのための処理などにも時間がかかってしまい、あまり好ましくない。

そこで、ａ、ｂで行・列ベクトル単位で分散する事は述べた通りだが、Ｃにおいても行単位で分散するという事にした。つまり、１～３行まではクライアントＡ、４～６行まではクライアントＢ・・・・という様に分散を行うという事になった。

Ⅱ－１　計算結果をどのように集めるか？

製作を進めるうち、計算結果がクライアントから返ってきた時に、どのように計算結果を一つの配列に代入するかという事が問題となった。

クライアントとの通信はアウトプロセスのプログラムが行っているので、クライアントから返ってきたデータをアウトプロセスのプログラムから呼び出し元のプログラムへ伝えなくてはならない。しかし、Ｖｉｓｕａｌ　Ｂａｓｉｃでは呼び出し元の変数を、ＡｃｔｉｖｅＸ　ＥＸＥからは利用できないので、直接配列に代入するという方法はとれない。また､以前利用したＤＤＥを用いて呼び出し元にデータを送信するという方法も考えたが、データの量が多いので、シングルプロセスのサーバに処理を行わせては、他のプロセスからもデータが送られてきた時に対応する事ができない。よって、複数のプロセスで配列を共有する方法が必要となった。

そこで、その方法を模索した結果、呼び出し元でオブジェクト（ＡｃｉｔｉｖｅＸ　ＤＬＬによるクラス）を作成、アウトプロセスのプログラムに参照渡しで渡し、その参照をアウトプロセスのプログラムの方で代入して用いる事で、複数のプロセスでデータを共有できる事が分かった。しかし、そこには一つ問題があり、その方法では配列はＰｕｂｌｉｃな変数として宣言できない、つまり、共有できないということである。そこで、配列はＰｕｂｌｉｃな変数として宣言せずに、プロパティとして公開し、配列の要素を変更できるようにした。

Ⅱ－２　行列データをどのようにして送信するか？

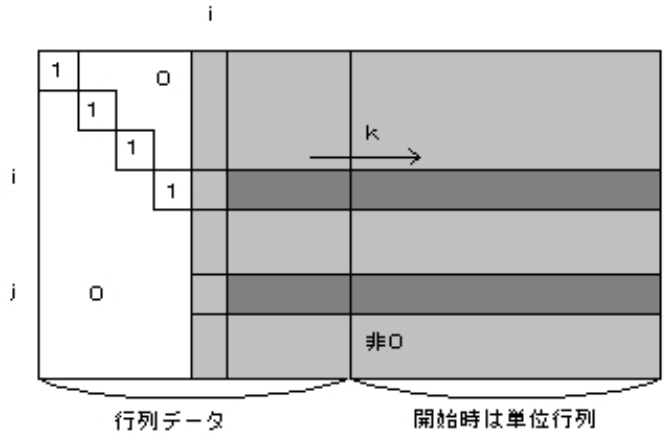
クライアントが計算を行うためには、行列データをサーバから受け、そのデータを配列に代入するというステップが必要である。そのために、計算開始の命令と共に行列データを送信する事にしたが、行列データをいかに分解し、また構成するかという事を考えなくてはならなかった。

そこで、配列に保持されている行列データを”,”を挟みながら結合し、クライアントへ送信、クライアントは受け取ったデータをＳｐｌｉｔ関数（分割文字を指定することにより、分割データを保持した配列を作成可能）により分割して行列データ用配列に代入するという方法をとることにし、非常にスムーズに処理を行う事が可能となった。

Ⅱ－３　計算終了をどのように判断するか？

すべてのクライアントが計算を終了した事を判断するために､共有するオブジェクトに､全てのクライアント数と、カウンタ用変数を用意しておき、クライアントからの計算結果を書き込む度に、カウンタを１つ加算して、記録してある全てのクライアント数と比較するようにした。この手順を繰り返せば、最終的には、最後のクライアントのデータを書き込んだ時に、カウンタが全クライアント数と一致するので、そのことを呼び出し元にＤＤＥで伝達する事によって計算終了を判断することができる。

Ⅲ：逆行列の計算をどのように分散するか？



逆行列の分散は乗算の分散とは異なり依存関係があるので、乗算の分散よりは非常に難

易度が高くなる。

どこに依存関係が生じるのかというと、図は（ｉ，ｉ）成分を１にすべく掃出しを行っている事を示す図なのだが、（ｉ，ｉ）成分の値というのは、ｉより前の列の掃出しの時に必ず変更されているので、ｉ列の掃出しと、それ以前の列の掃出しの順序が逆になってしまった場合、正しい計算結果を得ることができない。よって、ｉが進むループで同期を取らなくてはならない。その理由から、ｉを分散するのではなく、その内側のどこかで分散を行わなくてはならない。そこで選択肢は、ｋを分散するか、ｊを分散するかの２通りある。ここではどちらを分散してもよいが、２つを比べ分散がより容易なｊのループを分散させる事にした。

上記のように逆行列を求める事が分散可能であると分かった。しかし、乗算と違い、クライアントへ一度命令を出せば計算が完了するわけではなく、ｉが一つ進むたびにデータをサーバに返し、再び命令を行う。という手順を踏むため、より通信でのオーバーヘッドが増加することになると予想する事ができた。

Ⅲ－１　共有資源でのトラブルをどうやって防ぐか？

乗算の分散では、依存関係がないため、共有資源に書き込んだ計算結果を他のプロセスが読み込むという事がなかった。しかし、逆行列を求める事の分散では、書き込み中での、他のプロセスによる読み込みなどのトラブルが起きてしまう可能性がある。そこで、排他制御と呼ばれる手法を用いて、それを避けられるように改善を行った。

まず、共有資源であるオブジェクトの中に使用中であるか、使用中でないかのフラグを用意する。そして、プロセスが共有資源を利用する時にはそのフラグをＴｒｕｅにするようにしておく、そうする事によって、フラグがＴｒｕｅであれば待機、そうでなければ共有資源を利用できる、というような判断を可能とし、共有資源での衝突を防ぐ事ができる。

☆諸問題の解決

- ユーザーが逐次か分散で計算するかを選択できるようにするため、それぞれのモードを作成した。
- クライアントのアプリケーションは、試作２の計算部分だけを取り出し、サーバへの接続を行えるように作り変え、シンプルなＧｕｉとして作成した。

☆分散処理の最終的な流れ（乗算）

クライアントからの接続が完了
（この時に共有のオブジェクトをアウトプロセスのプログラムに渡しておく）
↓
計算式の中に乗算が存在した場合、共有オブジェクトに計算すべき行列を書き込み、
行数と台数から１台のクライアントに割り振る計算量を決定する
↓
アウトプロセスのプログラムを介して１台ずつに計算命令と行列データを送信する。
（呼び出し元は、全ての計算が完了した事が伝えられるまで空ループによって待機する）

↓

クライアントは受信したデータを配列に代入し、計算を行う

↓

クライアントから計算結果を受けたアウトプロセスのプログラムは排他制御を行いながら共有オブジェクトにデータを書き込む(他のプロセスが書き込みを行っていた場合は待機)

↓

全てのクライアントが計算結果を返した場合は、最後の書き込みの時に呼び出し元に計算終了を伝える

↓

呼び出し元は共有オブジェクトのデータを読み込み、次の計算に移る

☆分散処理の最終的な流れ（逆行列の計算）

※乗算と同じ部分は省略する

逆行列ボタンが押される。

↓

乗算と同じように計算命令を行う。

↓

全てのクライアントから計算結果が返ってきた事が呼び出し元に伝えられる。

↓

呼び出し元は上記の図における次のｉの計算のために、再び計算命令を行う。
（書き換えられた共有オブジェクトのデータを送信）

↓

最初に単位行列であった部分とそうでない部分との境までの列、全てが掃出せた場合
呼び出し元は計算結果を出力する

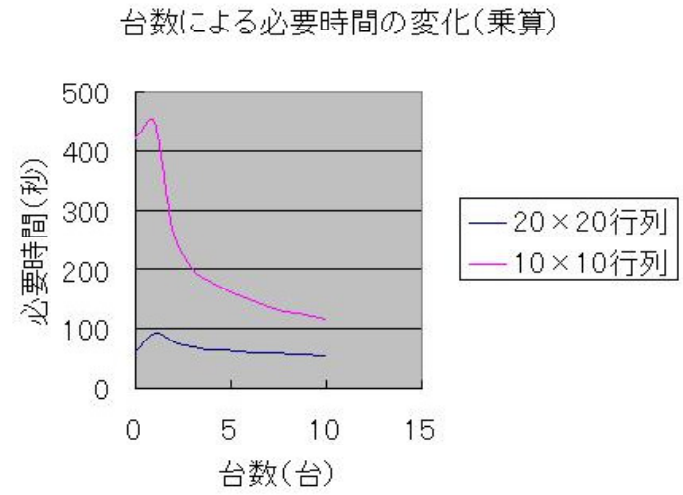
☆性能調査

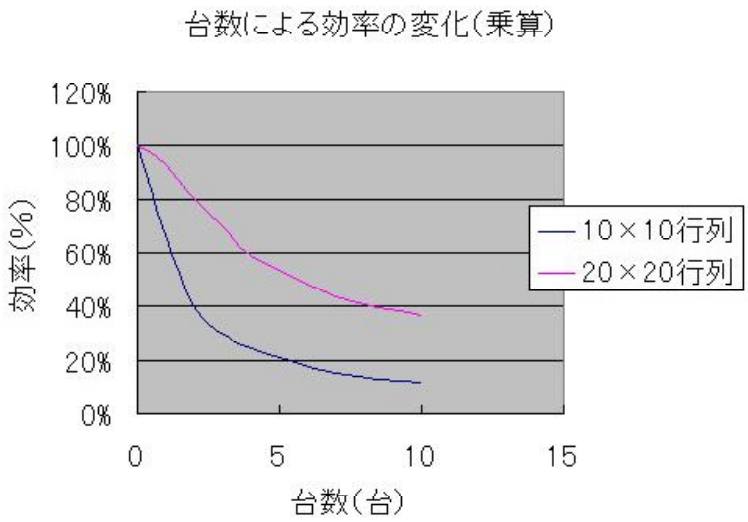
作成したソフトを実際にパソコン室で用い、性能を調査した。

※０台の時は、通信を介せず、逐次で計算を行った時を示す

※行う計算は”Ａ×Ｂ”の分散で、各要素は２０桁の乱数となっているとする。

Ｉ：乗算の分散について





グラフの数値から読み取れる事は、分散するための計算量自体があまり多くないと分散の効果はあまり得られないようであるという事だ。１０×１０行列の結果では１０台で分散した時の効率で１０%程度しか得ることができていない。

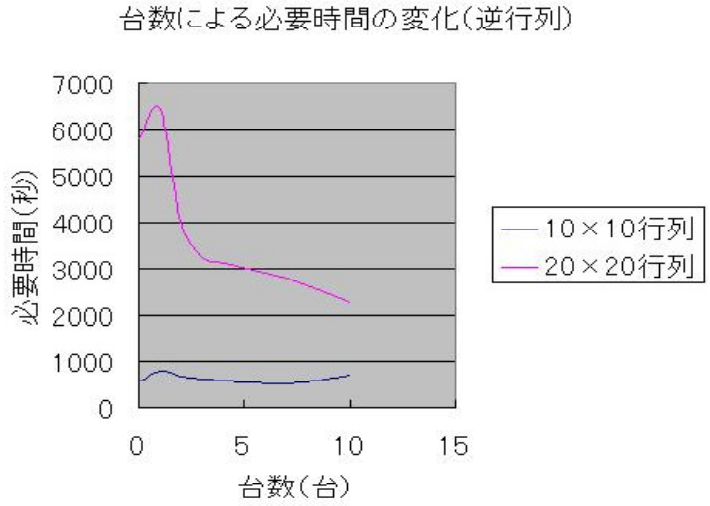
そこで、計算量が多ければ効率は上がるのか確かめるために２０×２０行列の分散に対しての調査も行った。その結果、やはり計算量が増えれば分散の効果が高く得られるという事をグラフが示す結果となった。

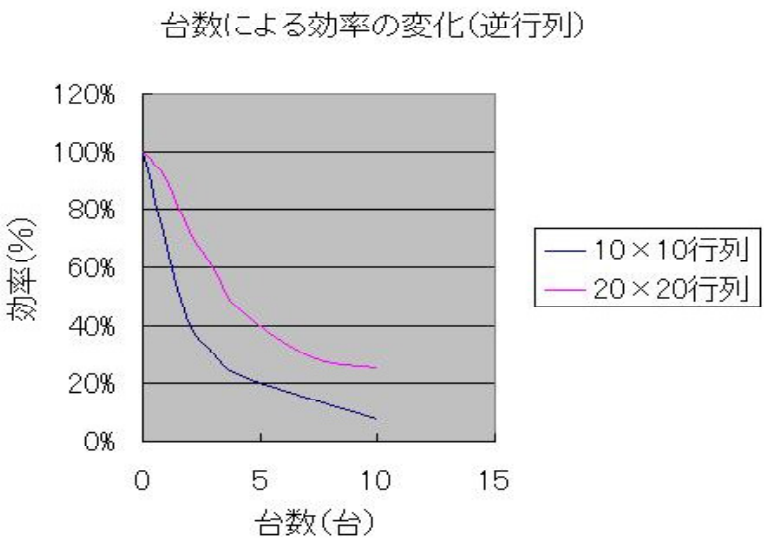
次に、グラフの形に注目してみると、計算量が増える事によって効率の低下も緩くなっている事がわかる。また、台数が増えるに従って効率が低下している事は確かだが、効率の低下の割合は徐々に減り、水平に近くなっている事も見て取れる。台数が増えていくと、

最後にはあるていどの効率に落ち着いてしまうのであろう。

Ⅱ：逆行列を求める事の分散について

※条件は調査１と同じとする



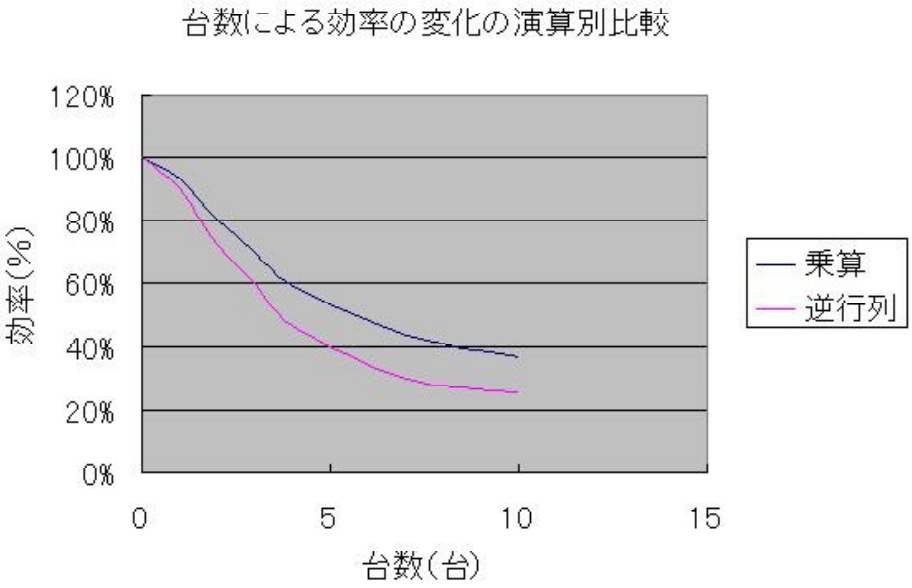


逆行列についての調査はさらに厳しい結果となった。１０×１０行列の必要時間のグラフにおいては、１０台に達するぐらいのころから、台数が増えているにもかかわらず、必要時間は増加してしまっている。また、効率の低下の割合も乗算の時と比べ高い。

しかしながら、計算量が増えるにしたがって、分散の効果が上がっている事は乗算の場合と同じで確かである。

Ⅲ：乗算と逆行列を求める場合の比較

※２０×２０行列・要素は２０桁の乱数、としての結果である



乗算と逆行列の分散の効率を比較すると、やはり逆行列の計算の方が通信を繰り返し、オーバーヘッドが増加してしまうため低効率となっているようである。しかし、グラフの形としてはあまり大差がない。演算の内容や手順はまったく異なるが、分散効率の変化を見る上ではあまり関係がないようだ。

☆実行画面

〇サーバ（計算機本体）

○計算用クライアント

第６項　分散型行列計算機製作を終えての考察

このプログラムの製作は多倍長のルーチンの製作から始めたという事と、資料がまったく存在しないという理由から非常に時間がかかった。その状況からも分かるが、長い製作を経て一つ分かった事は、やはり個人での分散のアプリケーションというのはまったく作られていないという事だ。インターネット上でそういったアプリケーションのサンプルや、情報などがないか探してみたが、ただの１つも存在を確認する事ができなかった。おそらく、プログラミングの高い技術が必要というわけではないが、そういったアイデア自体をあまり思いつかない。また、作成に手間がかかるという理由から、製作が行われないのであろうと私には思った。こういった状況を見ても、一般の人々への分散というアイデアへの関心の向上が必要と思われた。

しかし、こういった状況だからこそ、このプログラムの意義がある。こういったプログラムを大学や研究施設以外で作ったのはおそらく私が始めてであろう。（公開せずに作成していた場合は把握できない）

性能について考えると、性能としては非常に高速であるとは言えないが、それなりの性能を持っていると私は思う。また、これをもっと高速化して行くのであれば、データ送信をバイナリにする、クライアントへの行列データの送信を必要な要素だけにするなどの改善点が挙げられる。これらの点はプログラムを理解しにくくしてしまい、後でのプログラムを利用する場合などの事も考え、今回は採用は見送った。しかしながら、今後もこのプログラムの改善は続けていこうと思う。

第４節　素数発見プロジェクト２(ボランティアコンピューティング的方向性)

第１項　試作１「スクリーンセーバー型グリッドソフト１」

ここまでのソフトウェアは、グリッドの技術を用いたパソコンクラスタといったような性格のソフトウェアであった。ここからは方向性を転換し、もう１つの方向性であるＳｅｔｉ＠ｈｏｍｅのようなプロジェクトを、今までのプログラムの製作での技術を応用し、実現する事にした。

☆製作過程

まずはプログラムをどういった物にし、どういった機能が必要で、また、どういった条件があるのか考えた。

Ⅰ：プログラムの概要と必要な機能・

・ひとまず、パソコン室で行ってきた素数表作成というテーマとして継続する
・プログラムの形態としてはスクリーンセーバーであり、その起動後に、処理を行う
・ある程度大きな範囲（ユニット）を、起動するたびに少しずつ計算し、それが終わったならば、計算結果をサーバに送り返す
・サーバ側で、クライアントの情報を管理する
・起動後ビジュアル的に、ユーザーへプロジェクト参加の様子が伝わるようにする
・サーバー側の不備にも対応できるようにする

Ⅱ：条件

・ユーザーがコンピューターを再び使用しようとした時、なるべく短時間で、スクリーンセーバーが終了できるような構成とする
・スクリーンセーバー起動後であるので、処理能力はフルに利用してよい
・ユーザーのパソコンの安全性のためにレジストリは用いない
この他は、第２章の冒頭で述べた通りである。以上を踏まえて製作を進める事とする

Ⅲ：行う処理のおおまかな流れ（製作前）

初期設定を行う

↓

サーバへ接続し、計算すべき範囲を得る

↓

クライアントは与えられた範囲を､スクリーンセーバーが起動するたびに少しずつ計算し、保持しておく

↓

計算範囲を終えたらサーバへ計算結果を送信し、新しい計算範囲を受け取る

以上を繰り返す

☆製作の過程での問題点

Ⅰ：初期設定をどこで行えるようにするか？

このプログラムはスクリーンセーバーなので、フォームなどは存在しない。そのため、サーバのアドレスや、ニックネームの登録を行うためには、通常とは違う方法を用いなければならない。

そこで、まず、初回のスクリーンセーバーの起動時に設定を行えるようにするという方法を考えたが、スクリーンセーバーは利用しないから表示されるものであって、その時に設定を求めるというのは都合がよくない。また、初回しか設定できないとなると、ユーザーが変更したい時に設定が出来なくなってしまうので断念した。

次に、情報の管理をテキストファイルで行うようにする事で、ユーザーの書き換えによって変更を可能にする、という方法を考えた。しかし、テキストファイルでは情報があまりにも無防備に公開されてしまう。また、ユーザーが書く様式を間違えてしまうかもしれないという理由から、これも断念した。

最後に、スクリーンセーバーを設定するフォームには設定というボタンが存在する事に気づいた。調べてみると、設定ボタンを用いれば、いつでもスクリーンセーバーのプログラム中で用意しておいたイベントを利用する事が分かった。よって、この設定ボタンを用いて初期設定を行うこととした。

Ⅱ：いかにしてレジストリを用いずにプログラムを作成するか？

私がユーザの視点から考えた時、サーバへの接続の次にプロジェクトへ抱く不安は、パソコン中のデータの書き換えが行われてしまうことである。その中でも特にレジストリの

書き換えについては非常に敏感にならざるおえない。そのため、レジストリを用いる代わりに、ランダムアクセスのファイルを用意し、それに、構造体に保持しておいたプログラムの情報を保存しておくという方法をとった。

この方法をによって、そのファイルと計算データのファイルさえ存在すれば、クライアントプログラムがインストールされているどのパソコンでも計算を行え、プログラムのアップデートもその時点までの情報を損なわないで行うことができる、などの利点も生まれた。

Ⅲ：どのようにしてサーバー側でクライアントの情報を管理するのか？

本章第２節の試作６の内容のように、このプログラムでも接続処理を行う時に、アウトプロセスの呼び出し元とひとまず通信を行う。その部分を利用して、呼び出し元が最初のクライアントからの通信の時に情報を受け取り管理し､アウトプロセスのプログラムには､範囲を渡し、計算結果を保存するという作業を行わせるという分担とする。

そして、サーバー側で保持しておいた情報は、クライアント番号を割り振り、ファイルにレコードとして保持しておく。

Ⅳ：さまざまな情報をどのようにして送信するか？

ここでも、本章第３節の行列データの送信の時と同じように、区切り文字を設定し、Ｓｐｌｉｔ関数によって分割して用いるという手法を用いる事にする。それによって複数のデータを送信する必要が生じたり、また、追加で送信しなければならない情報がある場合でも、容易に変更を行う事ができる。

例　　神林！１００００！０　というデータを送信
神林＝ニックネーム　１００００＝計算した範囲　０＝始めての接続か？
というようなデータ送信の方法である

Ⅴ：クライアント側で少しずつ計算、完了したらサーバへ接続という事をどのように実現するか？

これを実現する事は､さまざまな状況についての処理を用意しなければならない事から、非常に苦労することとなった。

まず、起動が初回の場合には無条件にサーバへ接続を行うようにしなければならない。そこで、初期設定の時に、レジストリファイルの計算範囲の欄に”＃”を書き込むようにし、プログラム起動時にその文字が存在すればサーバへ接続するようにした。（計算範囲を与えられれば、自動的に計算範囲の欄は書き換えられる）

次に、すべての計算が終了した場合についてである。計算終了後すぐにサーバへ接続を行う事ができればばよいが、接続前にプログラムが終了してしまった場合、次回起動時に接続を行わなくてはならない。そのため、計算が終了しているかのフラグをファイルに記録しておき、それがＴｒｕｅの場合にはサーバへ接続を行うようにした。

他には、ユーザーがマウスなどを動かしてスクリーンセーバーが終了させられてしまう状況についても考えなければならなかった。ここではできるだけユーザーの意志にしたが

ってすぐに終了できるようにしたかったのだが、プログラムの処理の中では、終了されては困るような処理を行っている場合もある。そのため、マウスなどが動いても、終了処理を行ってよいというフラグがＴｒｕｅになっていない限りは終了することができないようにせざるおえなかった。

Ｖ－１　計算をどのように行うか？

計算はパソコン室での製作の時と同じように、Ｃ言語のプログラムをＳｈｅｌｌ関数で呼び出すという方法で行うのだが、ユーザーがプログラムを急に停止してしまう可能性を踏まえ、１回で計算する量を比較的少なくし、それを何度も呼び出すというようにした。

これにより、少しずつ計算した結果をファイルに書き込んでおき、すべてが終了したらそのファイルのデータを全てサーバへ送信するというように利用する事ができ、突然のプログラム終了の場合でも、少量の計算結果が無駄になるだけで済む。

Ⅵ：サーバ側でクライアントの接続をどのように処理するか？

まず、クライアントの接続が初回の場合は、新規でクライアントの情報をクライアントデータを保持する構造体の配列に追加し､すでにサーバから計算範囲を受けている場合は、アドレスとニックネームを参照し、一致するものがあれば、その情報に基づいて処理を行うようにした。しかし、以上の２つの場合に当てはまらない場合がある。それは、サーバかクライアントいずれかのトラブルによって、双方の情報が食い違っている場合である。

この問題は、クライアントの情報の改ざんなどによって起こる可能性もあるが、サーバが保持していた情報を誤ってクリアしてしまった場合や、クライアントが初期設定を行い直した場合も考えられる。この問題による対処を考えなければプロジェクトは正常に進行する事ができない。そこで、この問題を解決するため、初回の接続ではいが、情報が一致しない場合は、クライアントへ「情報が一致していないという警告」を送り、初回の接続として再接続を行わせる事とした（クライアントの情報はクリアされる)。

Ⅶ：サーバへの接続中に起こった問題にどう対処するか？

最初は、サーバへの接続時のエラー処理などは行っていなかったので、接続時にエラーが起こると、クライアントがスクリーンセーバーの状態で固まってしまうという事態が起きた。

これに対処するため、まずＰＩＮＧを用いて、アドレスが正しいか、コンピュータが起動されているか確認し、その後Ｃｏｎｎｎｅｃｔを呼ぶこととした。しかし、これではポートが開いていない、つまり、サーバのアプリケーションなどが起動していない時にはやはりエラーとなってしまう。そのため、Ｃｏｎｎｅｃｔメソッドに失敗のエラーに対しては例外処理を行い、一分後に再び接続を行うタイマーを起動させる事とし、５回接続を行っても正常に接続を介する事ができない場合は、強制終了するようにした。

Ⅷ：諸問題の解決

・サーバ側でクライアントの情報をビジュアル的に把握できるように、クライアントの情報をＭＳｆｌｅｘグリッドコントロールを用いて表形式で表示するようにした。

・クライアントでも、計算中、データ送信中など、プロジェクトに参加している事が伝わるような情報を、常に表示するようにした。
・完成したスクリーンセーバーを自宅のパソコン以外で動かしたところ、文字などが画面からはみ出してしまう事が分かった。これは、コントロールの配置などを、静的に配置してしまっている事から起きる問題であった。（自宅での製作のため盲点となっていた）
　そこで、すべてのコンピューターで正しく表示できるようにするために、すべてのコントロールを画面の大きさなどのの情報に合わせ動的に表示するよう改善を行った。
・２台のパソコン間で動作確認を行うと、データを正しく送信することができない場合が確認された。原因を調べると、まだ接続が完了していない状態で、データの送信を行ってしまっている事が原因であった。その問題を解決するため、接続が完了したことが確認されるまでループで待機し、接続が介された事を確認してからデータ送信を行うように改善した。
・ユーザーが計算状況を把握できるようにプログレスバーで計算がどれだけ完了しているのかを表示するようにした。

☆最終的なプログラムの流れ

サーバが動作しているようであれば、クライアントはサーバへ接続を行う（初回）
↓
サーバは、初回の接続か、すでに一回以上接続を行ったクライアントか判断を行う
↓
初回の接続であれば新規でデータを追加し、そうでなければ既存の情報を変更する
↓
サーバの指示で、アウトプロセスのプログラムの方へ接続を行う
↓
アウトプロセスのプログラムの方で新しい計算範囲を渡し、計算結果があれば保存する
↓
与えられた情報に従い、構造体の情報を変更し、Ｃ言語のＥＸＥを用いて計算を少しずつ行う
↓
計算結果は少しずつファイルにためていき、全て終わったならば、再びサーバへ接続を行う。（初回ではない）

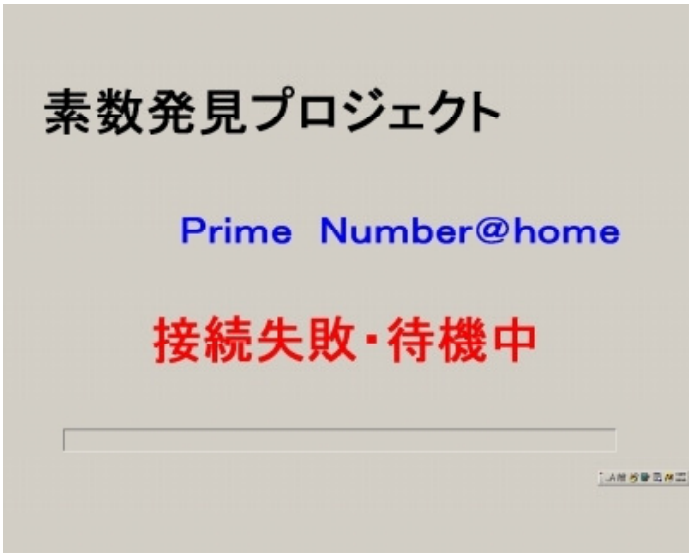
以上の手順を繰り返す

☆改善すべき点

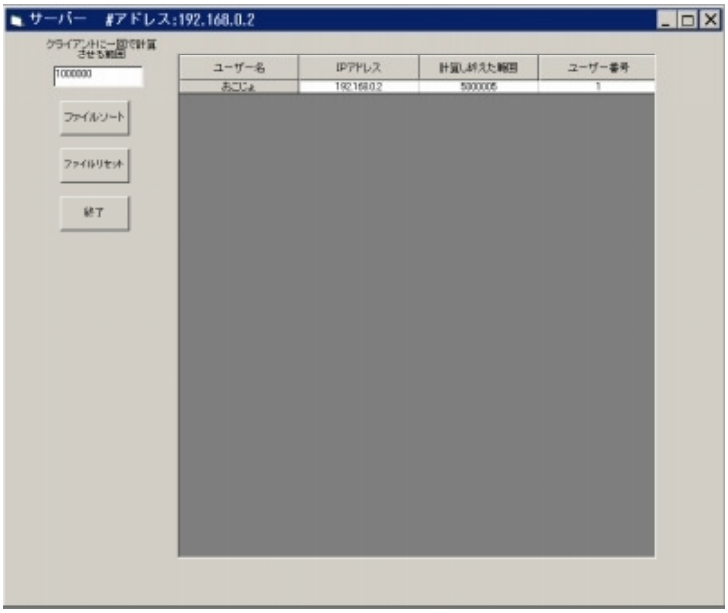
・もっとプロジェクトの情報がビジュアル的に伝わるようにする。
・素数発見のアルゴリズムを改良する。
・常駐して計算が行えるようにする。

☆実行画面

〇クライアントアプリケーション



〇サーバアプリケーション



第２項　「素数判定のアルゴリズムを改良する」

試作１でひとまずアプリケーションの雛型は完成したので、自分自身の興味もひとつの理由として、素数判定のアルゴリズムの改良を行う事にした。

※この項での計算時間は自宅のパソコンでのものである。

※ここまでのアルゴリズムでの計算時間は、１０００万までの素数列挙で**１５１．０４秒**かかっていた

☆製作の過程

Ⅰ：試行割り算法の改良

ここまでの、Ｎに対して、√Ｎまでの奇数でひたすら除算を行い、割り切る事ができなかったら、素数と判断するという方法を、試行割り算法と呼ぶ事にする。そこで、まず、この方法の改良を行った。

素数を表にしてみると分かることだが、２、３以外の素数はすべて、６ｎ＋１もしくは、６ｎ＋５の形をしている。この性質を利用して、試行の対象をその２つの形をした数だけにし、除算を行う数も、その形の数に２と３を加えた数だけにすれば高速化がはかれるはずだと考えた。

その改良を行った結果、計算時間は**７４．９８秒**まで縮める事ができた。

さらに高速化を目指すならば、割る数を、割る時点までに見つけた素数にするという方法があるが、ここではさらに早いアルゴリズムが存在するので次に移ることにした。

Ⅱ：エラトステネスの篩（ふるい）の利用

次に、素数判定のアルゴリズムとしてエラトステネスの篩（ふるい）というアルゴリズムを試すことにした。このアルゴリズムは非常に有名であるので、プログラミングを行う人間であれば大抵は知っているのではないであろうかと思う。その基本的な内容は

１：探索したい範囲と同じ大きさの配列を用意し、初期化しておく（素数であるか、ないかのフラグを記録する）↓

２，３，４，５，６，７，８，９，１０，１１，１２，１３，１４，１５，１６・・

２：まず、２の倍数をチェックしていく（その数は除く）↓

２，３，４，５，６，７，８，９，１０，１１，１２，１３，１４，１５，１６・・

３：次に２以上で、チェックされていない数を探す、すると、３が見つかる

４：同じように３の倍数を消していく（チェックされている場合はそのまま）

２，３，４，５，６，７，８，９，１０，１１，１２，１３，１４，１５，１６・・

以上がもっとも基本的なエラトステネスの篩のアルゴリズムであり、この手順を探索したい範囲の最後まで繰り返すと、チェックされていない数は素数であると分かる。

このアルゴリズムでプログラムを組んだ結果、計算時間は**３．５７秒**まで縮める事ができた。

結果から分かるように、このアルゴリズムは試行割り算法に比べ圧倒的な早さを誇る。その理由に、試行割り算法は除算をベースとしたアルゴリズムであるからという理由がある。コンピューターでは除算が加算や乗算に比べ高い処理能力を必要としてしまうため、除算を繰り返すのではあまり効率的ではないのである。一方、エラトステネスの篩は基本的に加算、また、１つの数についてループを行わず一括して求めていけるので高速なのである。

Ⅱ－１　高速化を目指す１

上記のエラトステネスの篩は基本的なアルゴリズムであるので、さらに工夫を行う事ができる。

上記の手順では”３の倍数を消していく”となっているが、エラトステネスの篩には、「素数ｐまでの倍数をチェックして、残った数のうち、ｐの２乗より小さいものは、すべて素数である」という性質がある。そのため上記の例では３^2 である９からの倍数をチェックすれば良い。

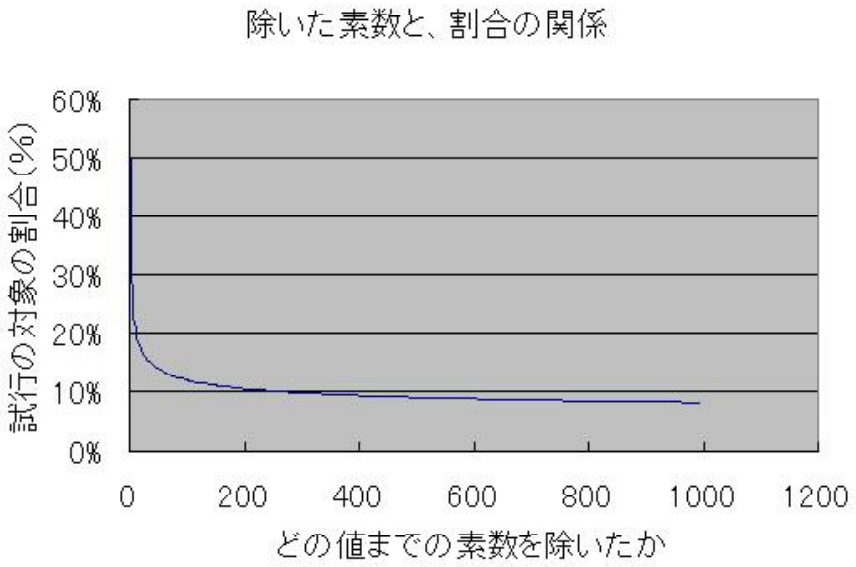
また、素数ｎにおいて、ｎ^2 は、ｎは奇素数であることよりｎの奇数倍である。そこに、ｎを加えた場合、偶数倍となり、偶数倍は２を因数に持つので、素数ではあり得ない。よってチェックしなくてもすでにチェックされているはずである。そのため、上記の例では３の２倍である６の倍数を消していけばよいこととなる。以上の２点を改良し、プログラムを組んだのだが、１０００万までの計算時間は、**３．５２秒**とほんのわずかしか縮める事ができなかった。しかし、今回ではほんのわずかの差しか現れなかったが、もっと探索範囲を広げれば差は明確に現れるはずである。

Ⅱ－２　高速化を目指す２

次に、試行割り算法の改良の時に用いた手法を、エラトステネスの篩においても試してみることにした。

試行割り算法の改良では６ｎ＋１　６ｎ＋５の上に２，３以上のすべての素数は存在していると書いたが、この考え方をさらに発展させると、２，３，５，７以外の素数は全て、２１０＋１、１１，１３，１７，１９，２３・・・・・というような事も分かる。このように別に考える素数を増やす事で、ａｘ＋ｎというような式を得、計算する前から計算する対象を減らす事ができる。そして、２，３，５，７まで除く事で試行の対象は全体の約

２２．８％まで減らす事ができる。



除いた素数と、残った数の割合は上のようになるのだが、７までの素数を除いただけでも、約４５ほどの式になってしまうので、素数を除いていく事で割合を減らしていくというのはプログラミング上大変な手間がかかる。よって、ここでは２１０ｎ＋ｘという形の式を用いてプログラミングし、その結果、**２．３秒**まで縮める事ができた。

Ⅲ　確率的な判定法

素数の判定法には今までの２つの方法のように素数であるか、ないかを確実に判定できるものの他に、確実ではないが、ほぼ確実に近く、そして速度の速い判定法が存在する。しかし、それらの判定法は数学の高い知識を必要とするので、既存のプログラムを参考にし、数学科の教官の協力のもと、製作を進めることとした。

Ⅲ－１　Ｆｅｒｍａｔ法

フェルマーの小定理と呼ばれる定理として以下が証明されている。

a と p が互いに素で, p が奇素数ならば, a^（p-1）= 1（mod p）が成り立つ.

Ｆｅｒｍａｔ法はこの定理の逆を利用して素数判定を行おうというものなのだが、残念ながら逆が必ずしも成り立つわけではない、それには疑素数、強疑素数などの要素が関係してくるのだが、数学的な説明は省略し、プログラムにするにはどうすればよいか挙げる。

１：　１以上、n 以下である自然数 a をランダムに選ぶ。

２：　a の n-1 乗を n で割る。

３：　割った結果、余りが１であれば、ｎは素数である可能性があると判断し、同じ手順を繰り返す、そうでなければ合成数であると判断する（合成数であるとの判断を誤ることは無い。つまり、素数を合成数と判断することはない）

以上を繰り返す事により、素数判定を行う。また、複数のａについて試行を行う事で確率を高めることができる。（Ｆｅｒｍａｔ法では決して判定できない数も存在する）

Ⅲ－２　Ｓｏｌｏｖａｙ－Ｓｔｒａｓｓｅｎｎ法

Ｆｅｒｍａｔ法で、合成数を素数と判断してしまう部分について改良を行ったアルゴリズムがこれである。ふたたび数学的な説明は省略するが、その手順は

１：　１以上、n 以下である自然数 a をランダムに選ぶ。

２：　a について場合分けを行う。

３Ａ：　1 以上、ｎ以下の自然数ｘで、ｘ＾２＝ａ（ｍｏｄ　ｎ）となるｘが存在する場合は、ａ＾（（ｎ－１）／２）の（ｍｏｄ　ｎ）の値をとる。このとき、ａ＾（（ｎ－１）／２）＝１（ｍｏｄ　ｎ）であるならば、素数である可能性があり、再び２からの手順を繰り返す。そうでなければ、ｎは合成数である

３Ｂ：　３Ａの条件を満たさない場合も、ａ＾（（ｎ－１）／２）の（ｍｏｄ　ｎ）の値をとる。ａ＾（（ｎ－１）／２）＝ｎ－１（ｍｏｄ　ｎ）であるならば、素数である可能性があり、そうでなければ、ｎは合成数である。

というものである。これについても複数のａについて試行を行う事で確率を高める事ができ。１回で試行するごとに誤る確立を約１/２にする事がきるので、ｎ回試行を行うことで、誤る確率を(1/2)^n 程度にする事ができる。

Ⅲ－３　Ｍｉｌｌｅｒ－Ｒａｂｉｎ法

このアルゴリズムもＳｏｌｏｖａｙ－Ｓｔｒａｓｓｅｎｎ法と同じく、Ｆｅｒｍａｔ法が改良されたものである。

１：　ｎ－１＝２^ｓ・ｔとなる自然数ｓ、奇数ｔをとる。

２：. １以上、n 以下である自然数 a をランダムに選ぶ。

３：　ａ^ｔ＝１（ｍｏｄ　ｎ）もしくは　０以上、ｓ以下のｉで、ａ^（ｔ・２＾ｉ）＝ｎ－１（ｍｏｄ　ｎ）が成り立つならば素数である可能性があり、他のａについても手順３を行う。どちらも成り立たないのであればｎ

は合成数。

これについても複数のａについて試行を行う事で確率を高める事ができ、Ｓｔｒａｓｓｅｎｎ法より効率がよく、誤る確率は試行の回数ｎにより（1/4)^ｎとすることができる。

以上の確率的な方法とエラトステネスの篩で、素数の判定（列挙ではない）を行うプログラムを作成し比較した。その結果、１０００万程度の素数の判定に、エラトステネスの篩では**２秒**ほどかかったが、確率的な方法は多倍長でなくても扱える限界の大きさの素数に対しても０．００秒と計測する事ができないほどの高速性を見せた。しかしながら、確率的な手法はどんなに確率が高くても、依然、確率的である事に変わりないので、私のこだわりの中では利用を許す事はできず､これらのアルゴリズムの利用は見送ることとした。

※確率的な方法の試行の回数は１０回で調査した。

Ⅳ　高速化によって生まれた問題点

予想以上の高速化によって、新しい問題点が生まれた。それは、あまりにも高速に素数を発見する事ができるため、通信でのオーバーヘッドや、サーバで保持するための容量などを無視する事ができなくなったのだ。

３０秒程度で１億までの素数を列挙する事ができる状況では、発見した素数を保持しておくだけで数十～数百メガバイト（一億までの素数をバイナリで保持したところ３０メガバイトほどになった）という容量を必要としてしまう。そして、当然のように、そのような巨大なデータをクライアントに送信させるということは事実上不可能である。

また、もう一つ問題点がある。試行割り算法で、発見した素数によって割っていく、また、エラトステネスの篩でふるっていくには、先の素数判定には、それ以前の計算データが必要となる。つまり、依存関係が生じるのである。

この問題は、最初から計算を行う、また、以前の計算データをサーバから渡すという方法で回避することができないわけではないが、最初から計算をやり直すというのはせっかく分散して計算を行っても他のクライアントがすでに行った計算をやり直すという事に他ならず、データを渡すという方法も桁数が上がってしまうと実質不可能である。この問題は、複数のクライアントで分散して計算を行うという環境の上では致命的な問題となってしまうのである。

以上の問題から、計算する題材の方向転換をせざるおえなくなった。

今までは、それほどの考えなしに素数表を作るなどと考えていたが、プロジェクト上の条件を考えれば当初から無理な公算が高かったと気づくべきであった。そこで、素数表を作るのではなく、巨大な素数の判定を行うことにテーマを変更した。

Ⅴ　巨大な素数の判定のためのアルゴリズム

巨大な素数の判定にあたり、発見する素数も今までの素数と違い、特別な種類の素数の発見を行うことにした。

素数には、いくつかの種類があり、有名なものに、メルセンヌ素数とフェルマー素数がある。メルセンヌ素数は**２＾ｐ－１（ｐは素数）**という形で表されるものであり、２進数で表した場合１１１１１・・・・という１の羅列になるという性質を持つ。現在では３８個発見されていることとなっている。

次にフェルマー素数だが、これは**２＾（２＾ｎ）+1**という形で表される、こちらは現在５個が見つかっているが、その他のものはまだ確認されていない。

この２つの内、メルセンヌ素数にはルカス法という効率的なアルゴリズムが存在し、比較的巨大な素数であっても高速に判定を行う事ができるという理由から、メルセンヌ素数の発見を行うことにした。（フェルマー数の探索は飛躍的に試行の対象が巨大化してしまうので、ここでは採用を見送った）

Ｌｕｃａｓの定理

メルセンヌ数　２^ｐ－１（ｐは素数）を Ｍｐとし、
数列　Ａｎを、Ａ１＝４，Ａｎ＋１＝Ａｎ^２－２　により定義する。
このとき、Ｍｐ（ｎ は３以上の素数）が素数であるための必要十分条件は、
Ａｎ－１がＭｐで割り切れることである。

この定理はフェルマーの小定理と違い、逆が成り立つ。そいて、この定理を発展させた素数判定法がＬｕｃａｓ法である。

Ｌｕｃａｓ法

メルセンヌ数　２^ｐ－１（ｐは素数）を Ｍｐとし、
数列　Ｓｎを、Ａ０＝４，Ｓｎ＋１＝（Ａｎ^２－１をＭｐを割った剰余）と定義する。
このとき、Ｓｐ－２＝０であればＭｐは素数、そうでなければ素数ではない。

Ｌｕｃａｓの定理での数列Ａは急激に増加するため、判定するｐまで Ａ を計算してから剰余をとる事が困難なので、Ｌｕｃａｓ法では毎回、剰余をとった形で数列を進めるように改良されている。以上を理解したうえで、既存のプログラムを参考に、Ｌｕｃａｓ法によるメルセンヌ素数判定プログラムを作成した。そして、その結果、メルセンヌ素数**Ｍ1279**（３８６桁）を、**３３．２８秒で**判定する事ができた。

第３項　試作２「スクリーンセーバー型グリッドソフト２」

アルゴリズムの改良を経て、プロジェクトの方向性は巨大メルセンヌ素数の判定を行うというものへ変更されることとなった。そこで、その変更を反映させ、また、まだ不十分である利便性、安全性、ビジュアル性などを改良することとした。

☆製作の過程

Ｉ：計算を少しずつ行う事をいかに実現するか？

以前にもこれと同じような問題があったが、今回は少し性質が異なる。前回の場合は複数の素数判定を行うため、プログラムを複数回起動する事を前提として良かった。そのため、少しずつファイルに貯めていけば、プログラムが終了されたとしても計算を行っていた少しの部分が無駄になるだけで済んだ。しかし、今回は１つの素数の判定を行うため、その手法を用いる事はできない。

そこで、プログラムの構造をよく考えてみると、ループのネスト構造となっている。その部分を上手く利用すれば、計算を途中から始める事が可能なのではないかと試行錯誤した結果、一番外側のループが一巡するたびに変数のデータをファイルに保存しておくことにより、次回起動時にデータを読み込み、変数に代入し直す事で、上記の事が可能になると分かった。

他の要素への配慮も含めプログラムを組んだ結果、下のような流れとなった。

プログラム起動時、前回の計算データが存在すれば読み込み、そこから計算を開始する
そうでなければ、指定された通りに、最初から計算を開始する
↓
一番外側のループを一巡するたびに、変数の内容をバイナリで記録しておく
↓
判定が完了したら、計算データを削除する

しかし、この手順では毎回書き込みを行わなくてはならないので、その部分でのオーバーヘッドが生まれてしまう。これを解決するため、プログラム本体から終了の指示が出た時のみ、すべての変数の内容を保存しておき、毎回保存しておくのはループのカウンタなどの少量のデータのみにするよう改良を行った。

Ⅱ：プログラム本体から、計算状況や計算終了をどうやって知るか？

プログラム本体から計算終了や計算状況を知るため、今までは、発見した素数が書き込まれるテキストファイルを監視する事によって実現していたことを、Ⅰでの計算データのファイルを監視することによって実現する事にした。それによって、ループがどこまで進んでいるのかも把握できので､プログレスバーによって進行状況を表示することもできる。そして、ループのカウンタの値の部分に本来であればなる事のない、－１や－２という数が書き込まれる事によって判定結果と計算終了を伝達する事とした。

Ⅲ：さらにビジュアル的に、ユーザーにプロジェクト参加を意識させるためにはどうすればよいか？

繰り返しになるが、ユーザーがクライアントのアプリケーションを利用することで、楽しみや関心を覚えるようでなければ、プロジェクトは成立しない。そこで、まずユーザーがどんな情報を得る事でプロジェクト参加を意識する事ができるか考えた。その結果、

・現在どんな値を計算しているのか
・今までに何回ワークをこなしたのか

・すべての参加している者の中でどれだけ自分は貢献しているのか（順位）

・現在どれくらいの時間、クライアントのアプリケーションを動かしているのか

といったものを挙げた。

実際に、これらの中で３つは採用し、４つ目はサーバー側で把握できるようにすることとした。結果、これらが表示される事で味気のないクライアントのアプリケーションがよりグリッドのアプリケーションらしくなったように思われる。

Ⅳ：サーバ側でクライアントのどのような情報を必要とするか？

サーバー側で、どんなクライアントがプロジェクトに参加しているのか？、また、プロジェクトの進行状況はどういった状態であるのかをサーバ側で把握したいと管理者は当然思うものであると私は思う。（とにかく、自分はそうである)、そこでそういった面を改善することにした。

まず、クライアントの処理能力がどれ程なのかを把握するため、インターネットで提供されている、コンピューター情報を得るＤＬＬなどを利用し、ＣＰＵのクロック、種類、物理メモリの量をサーバに送信させることとした（拒否する事も可能である)。その他、プロジェクトへの要望などがユーザーにあった場合、それをフィードバックするために、初期設定の時にコメントを記入するＢＯＸを用意し、思いついた時すぐにそれを伝えられるようにした。

次に、プロジェクトがメルセンヌ素数の判定となったため、どの数が判定中で、どの数が素数と判明したかを記録・表示しなくてはならない。そのために、クライアントの情報用ファイルとは別のファイルを１つ用意し、探索結果を記録しておいて、新しく数が割り振られたり、返ってきたりするたびに、表示内容を切り替えながら提示するようにした。

Ⅴ：データの改ざんを防ぐにはどうすればよいか？

これまでは、データの送信やファイルでの保存などでも、特に処理を行ってから送信・保存を行うという事はして来なかった。しかし、バイナリやランダムアクセスのファイルとして保存していても､バイナリエディタなどを用いることによって改ざんが可能であり、送信データなども、何らかの方法で改ざんが行われてしまうかもしれない。そこで、暗号化を行う必要性があることを認識し、どのように暗号化を行うか決めなくてはならなくなった。

暗号化にもたくさんの種類があり、極端に言えば、ＤＥＳやＲＳＡなどを利用する方法もある。しかし、それでは暗号・復号化に時間がかかってしまい、非常に高い技術を必要としてしまう。そこで、ここでは安易な思いつきによる改ざんが防げる程度の簡単な暗号化を行う事にした。

ここで用いた方法は、データをバイナリデータとして抽出してから倫理演算であるＸｏｒ（排他的論理和）結合を行う方法である。

例（要素一つに対して）

２５５　Ｘｏｒ　１００＝１５５　→　１５５　Ｘｏｒ　１００＝２５５

というように、Ｘｏｒ結合で用いた値さえ同じであれば復号化する事が可能である

上記のような事を、１度行っただけではさすがに暗号の信頼性が低くすぎるので、配列に代入されたバイナリデータの、添え字が奇数である部分と、偶数である部分での結合する値を変えたり、Ｘｏｒ結合をさらに何度か行う事によって、信頼度を高めた。
※結合する値は事前にサーバとクライアントで同じ値を保持している。
※暗号化後はファイル名に”＋”という文字が結合される。

Ｖ－１　文字列の暗号化での問題

ファイルの暗号化ではファイルをバイナリデータとして読み込めばよいだけなので、簡単にバイナリデータの形にする事ができるのだが、ただの文字列をバイナリデータに抽出する方法はＶｉｓｕａｌ　Ｂａｓｉｃには用意されていなかった。そこで、その方法を模索した結果､ＭｏｖｅＭｅｍｏｒｙという外部ＡＰＩを利用する事で可能なことが分かり、そのＡｐｉを利用することで文字列の暗号化が可能となった。

文字列を暗号化し、バイナリデータから文字列に戻すことによって「**?*90ｸｲ'　"ｹ８ﾖ'　：・・7/5>ｫ#ﾙ/・ｵく・ﾐ'>/・ｾｭ**」というような解読不可能な文字列を作り出すことできるのだが、復号化を行うと元に戻らない部分が出てしまうという問題が起こった。特定の文字は復号化ができないようなのである。

この問題はＶＢｕｎｉｃｏｄｅなどの仕様が原因のように思われるが、つまり、暗号化の後に文字列にしてしまうと復号化ができなくなってしまうのである。そこで暗号化後、文字列に戻す事を断念し、１５６，３５，１２５，２４・・・というようにバイナリデータを文字列として結合し、暗号として利用する事とした。

Ⅵ　ユーザーの不安を取り除くにはどうすればよいか？

プロジェクトを成功させるためには、ユーザーの不安を無くす事も非常に重要である。特に、サーバへの接続や、常駐するという面から、ユーザーは不安を抱きやすい。そういった心配を招かないようにするために、初期設定で、

・サーバへの接続を自動で行わせないようにする
・毎回自動で計算を開始しないようにする
・サーバへパソコンのスペックなどを送信しないようにする

というような選択を可能とした。

また、ヘルプのテキストファイルなどをしっかりと用意する事なども重要である。

Ⅶ　諸問題の解決

・Ｓｈｅｌｌ関数でプログラムを呼ぶ時に、プロセスＩＤを保持するようにし、終了処理の時に起動しているか確認する事ができるようにすることで、計算データの暗号化がタイミングのずれから失敗する問題をなくした。
・接続時のエラー処理が足りず、接続を介した後でエラーが起きてしまった場合、クライアントが固まってしまうという問題があった。そのため、データ送信後６０秒たっても応答がない場合、強制終了するようにした。

・以前のように大量のデータをファイルに保存するという処理が無くなったため、アウトプロセスのプログラムを呼ぶ事をやめ、サーバのプログラム本体だけですべての処理を行うように改良した。
・メルセンヌ素数での指数は素数である必要があるので、クライアントへ送信するための素数データがサーバには必要となる。
　最初は、毎回エラトステネスの篩などで生成しようかと考えたが、そのために処理能力を割きたくない、また、その時に生成する必要性がないので、事前にＣ言語のプログラムで作成しておいたバイナリデータの素数表を利用することとした。
・Ｃ言語のプログラムを、今までのように強制終了せずに、「ＥｎｄＴｅｌｌｅｒ」というファイルを生成することで、プログラム終了の指示を伝え、計算データを保存させてから終了するようにした。
・サーバとクライアントの保持している情報が一致しなかった場合、クライアントにデータを返し、もう一度接続を行わせるようにしていたが、その方法は効率的ではなく、また、通信の部分でのエラーを招きやすいので、１度の通信で、初回として計算を始める指示を出すように改良した。
・クライアントに割り振った値が確実に返ってくる事は保証できないため、判定結果が戻ってこない場合も想定しなければならなかった。そこで、念のため、２台のクライアントに同じ値を割り振ることとした。

第４項　試作３「タスクトレイ常駐型グリッドソフト１」

スクリーンセーバー型のグリッドソフトには１つ致命的な問題点がある。それは、プロジェクトへの貢献のために、パソコンの電源を長時間入れたままにして、スクリーンセーバーを動かしてくれるような、ハードユーザであればスクリーンセーバー型で問題はないのだが、そうではなく、特にパソコンを長時間利用するわけではない、本当に一般的なユーザーであれば、スクリーンセーバーを起動するまでもないまま１回のパソコンの利用を終えてしまう場合が多いかもしれないからである。

この事はこの研究のテーマである「一般的な人々」という部分を、クライアントを利用する側に対しても適用し考えるのならば、そのままにしておくことはできない。また、スクリーンセーバー型は余剰の処理能力利用という面から考えても、少しピントがずれた選択であると言わざるおえない。

そこで、ワープロとして使っていても、インターネットをしていても、どんな時にでも余剰の処理能力を利用できるように、タスクトレイ常駐型として作成し直す事とした。

☆製作の過程での問題点

Ⅰ　タスクトレイ常駐のプログラムを作成する

タスクトレイ常駐型のプログラムを作った経験がないため、まずは、情報を集めるところから始めなくてはならなかった。そして、インターネットで探した結果、以下の手順でタスクトレイ常駐のプログラムが作成可能である事が分かった

Ｓｈｅｌｌ＿ＮｏｔｉｆｙｉｃｏｎＡｐｉを用いて、タスクトレイにアイコンを表示する

↓

フォームを非表示にする

↓

アイコンの操作をした場合のイベントをプログラムで受けられるように工夫する事によって、適当な処理を行う事ができる（ここではフォームを表示し、タスクトレイからアイコンを取り除く処理を行う）

↓

最小化ボタンなどでサイズが変更された場合、再び、フォームを非表示にし、タスクトレイにアイコンを表示する

Ⅱ　余剰の処理能力を利用するにはどうすればよいか？

余剰の処理能力を利用するということは、ユーザーが利用しているプログラムなどの処理に負荷をかけてはいけないということである。

そこで、まずＣＰＵの使用率などから、余った処理能力を算出し、ループを調整するなどの方法を考えたが、ＣＰＵの使用率は、このプログラムの使用している分なども含んでしまうので、それを監視しながらの処理というのはあまり当てにならない。また、こちらの方が重要だが、ループの調節による処理能力の調節には機種依存の要素が多く、技術的にも非常に困難であるという理由から断念せざるおえなかった。

次に、考えた方法が、プログラムの優先度を下げるという方法である。Ｗｉｎｄｏｗｓのようにマルチタスクのｏｓでは優先度というものを基準に処理能力を割り振っているので優先度を最低にする事で、他のプログラムが必要とする処理能力に負担をかける事なくプログラムを動かす事ができる。この方法が比較的容易で、効果も信頼できることからこの方法を採用することとした。

優先度を下げる事は、ＯＳに関わる事なので、外部ＡｐｉであるＳｅｔＰｒｉｏｒｉｔｙＣｌａｓｓＡＰＩを利用し、優先度クラスをＩＤＬＥ＿ＰＲＩＯＲＩＴＹ＿ＣＬＡＳＳにする事で実現できた。

Ⅲ　諸問題の解決

・プログラムが処理を行っている間に、タスクトレイのアイコンをクリックしても表示されないという問題が起こった。プログラム側では、そういった問題が起こらないようにするため、一定の頻度でＯＳへ処理を戻していたので、起こるはずのない問題であった。そこで、フォームでＯＳからのイベントを受けるようにしていた所を、専用のＰｉｃｕｔｕｒｅＢｏｘを用意し、そちらでイベントを受けるようにしたところ、正常に動作するようになった。おそらく、フォームがホールドされており、イベントを受ける事ができなかったのであろうと思う。

・Ｓｈｅｌｌ関数によって呼び出されるプログラムの優先度も下げなければならないと、方法を探したが、Ｓｈｅｌｌ関数によるプログラムは呼び出し元と同じになるので特に処理を行う必要はなかった。

・優先度は基本的に最低としているが、ユーザーの意思によって、変更できるようにした。しかし、Ｓｈｅｌｌ関数によって呼ばれたプログラムはすでに一度呼ばれてしまった後なので、個別に変更する必要があった。そこで、Ｓｈｅｌｌ関数の時に保持しておいたプロセスＩＤを利用して、そちらの優先度も下げるようにした。

第５項　「プログラムの公開を目指す」

このプログラムのテストを行うため、校内の教師用ＬＡＮの利用を求める嘆願書を提出し、返答を待ったが、残念な事に利用の許可が降りることはなかった。

そのため、プログラムのテストを行うために、知人の協力の元で行うにしても、ＣＤ－Ｒなどで一枚一枚配布するというわけにはいかず、またメールでの添付という形でも容量が大きすぎるので、インターネットでの公開を行う事にした。

Ⅰ：ソフトのダウンロードをどのように行えるようにするか？」

まず最初に、Ｖｅｃｔｅｒなどのフリーウェア公開サイトで公開する事を考えたが、公開のための審査などもあり、これから改良を行っていく事を考えると、この選択は適当ではなかった。

次に考えた方法が、Ｙａｈｏｏブリーフケースなどの、無料ＦＴＰサーバのサービスを利用する方法である。しかし、知人以外の人間への公開を見すえて考えた場合、この方法も適当ではないと判断した。

結局、最終的には公開のために専用ホームページを作成する事にした。（一般に公開しない段階では、ホームページを作らず、ファイルをサーバに置き、そのアドレスを知人にだけ教える事で、予期した人間にしかダウンロードさせない事ができる）

したがって、プロバイダへ無料ホームページ用のスペースの登録（１０Ｍ）を行い、ＦＴＰサーバへの転送ソフトを用意する事で、知人へのプログラム公開が可能となった。

Ⅱ：サーバをどのようにして運用可能にするか？

まず、サーバをどこに置くか決めなくてはならない。理想を言えば、校内でサーバのアプリケーションを動かしたいと考えたのだが、校内のネットワークはサーバ室のサーバによって、外部のネットワークからの通信は固く遮断されており、一部の解放を要請したが、困難であるという回答であった。

そのため、サーバは自宅のパソコンで運用する事になった。

ここまでのテストはすべて、校内のＬＡＮや自宅のＬＡＮ内で行ったため、ＩＰアドレスなどはプライベートアドレスがあればよかった。しかし、インターネット上でのテストにあたり、グローバルＩＰアドレスで利用できるようにしなくてはならない。しかし、自宅のパソコンは常時接続のＩＳＤＮでダイアルアップである。そのため固定ＩＰを持つ事はできない。

その問題を解決するのが、ダイナミックＤＮＳというものである。ダイナミックＤＮＳはネットワークに接続する度に変わってしまうＩＰアドレスを、事前に登録しておいたド

メインに毎回割り当てる事によって、あたかも固定ＩＰとドメインを持っているような状況を作ることができるサービスである。そして、このサービスによって、クライアントアプリケーションを利用する側は一つのドメインを把握していればサーバと接続する事ができる。

そこで私は「家サーバープロジェクト」というサイトのサービスに登録を行い、”ｍｙｈｏｍｅｇｒｉｄ．ｍｙｈｏｍｅ．ｃｘ”のドメインを入手した。これによって、自宅のパソコンでもインターネットからの要求を受ける事ができる状況になった。また、加えて”ＤＩＣＥ”の導入も行った。”ＤＩＣＥ”はＩＰアドレスが変更されるたびに、自動的にサイトにアクセスし、ドメインに割り当てるＩＰアドレスを変更するというダイナミックＤＮＳ用のユーティリティソフトである。これによって、変更のたびに手動でサイトにアクセスし変更するという手間をかけず、半自動でドメインを維持する事ができるようになった。

Ⅱ－１　サーバを立てるにあたってのセキュリティの向上

グローバルなアドレスを公開して、サーバを立てるという事は、インターネット上の誰からでも自宅のコンピューターへアクセスする事が可能であるという事を意味している。そのため、何の対策も行わないと非常に危険である。

自宅ではルーターを用いずにネットワークに接続しているため、ルーターによるファイヤーウォール機能を期待する事ができない。そこで、フリーで入手する事ができ、高機能なファイヤーウォールソフトである”Ｚｏｎｅ　Ａｒａｒｍ”を導入した。また、コンピューター　ウィルスに対しての対策として”Ａｖｇ”も導入し、Ｗｉｎｄｏｗｓでの利用していないネットワークサービスなども全て停止させた。

この程度ではネットワークセキュリティはまだ万全とは言えないが、ひとまず以上でセキュリティ対策を行った事とした。

Ⅲ：サーバへの接続が行えないという問題の解決

以上のⅠ～Ⅱの問題を解決した事で、自宅のサーバは外部からの通信を受ける事ができるはずなのだが、知人がテストを行った結果、”接続失敗・待機中”になってしまうという問題が確認された。

まず、Ｐｉｎｇでサーバへ確認を行う部分で、ファイヤーウォールソフトによってｐｉｎｇが失敗してしまっているという問題があった。そこでＰｉｎｇを用いないで、サーバのアドレスが間違っている場合なども、Ｃｏｎｎｅｃｔメソッドの例外処理で一括で取り扱う事にし、Ｐｉｎｇを用いる部分は取り除いた。しかし、それだけではこの問題は解決できなかった。

そこで、アドレスをドメインではなく、グローバルＩＰを直接入力するようにして接続させると、正常に動作する事を確認する事ができた。その事から、クライアントでのドメインの名前解決が正しく行われていないか、Ｗｉｎｓｏｃｋコントロールではドメインの指定ができないのかの、２つの原因を予想したのだが、どちらの場合でも解決法は一つである。ＲｅｍｏｔｅＨｏｓｔのアドレスの指定で、事前にドメインから割り出しておいたＩＰアドレスを指定すればよいのである。

ドメインからＩＰアドレスを割り出すことは、ＧｅｔｈｏｓｔｂｙｎａｍｅＡＰＩを利用する事で比較的用意に実現できた。

以上の改善を行い、ドメインを入力しても正しく接続する事ができるようになった。

Ⅳ：ＷｉｎｄｏｗｓＸＰやＷｉｎｄｏｗｓ２０００でインストールする事ができない

上記のＯＳを利用しているパソコンはエラーによりインストールが正しく行えないという状況が確認された。これは以前からＶｉｓｕａｌ　Ｂａｓｉｃのエラーとして知られており、ユーザーがログインを行う時にひらがなや漢字などのダブルバイト文字を使用している場合に起こる問題である。

この問題はユーザーがログイン用のユーザーネームを変更することで回避が可能なのだが、ユーザーに解決を求めるのではなく、プログラム側で解決しておくべきと考え、改善を行った。

このエラーの原因はＶｉｓｕａｌ　Ｂａｓｉｃ付属のＳｅｔｕｐ１．ｅｘｅの欠陥が原因である。とマイクロソフト社が修正方法とともに公開している。それに従って、Ｓｅｔｕｐ１．ｅｘｅの欠陥部の改善を行った結果、正しくインストールされる事が確認された。

Ⅴ：ホームページでアプリケーションを公開する

複数の知人の協力によって、インターネット上での動作が確認されたので、ホームページ上での公開に踏み切る事にした。そのために、まず、ホームページの製作を行ったのだが、製作の上で留意した点は、

・必要最低限の情報だけを伝えるようにし、すぐにすべての内容を確認できるようにする。

・ユーザーが知りたい、また、疑問に思うような情報への回答を用意する。

以上の２点である。

第６項　試作４「タスクトレイ常駐型グリッドソフト２」

ホームページでの公開を行ったが、数日たっても１人の参加者も現れなかった。その変わり何通かの意見のメールが届いた。

それらの内容は基本的にはすべて同じ１つの事を挙げていた。要約すると、

「ＧＩＭＰＳというメルセンヌ素数発見のプロジェクトがあるのに、こちらに参加する意味はない」という事である。

この結果は、ある程度予測のできた事ではあったが、やはり、ユーザーの意欲というものを重視しなければならないと感じさせられた。

そこで上記の意見を元に、新しい計算テーマへの変更を行うこととした。

☆製作の過程での問題

Ⅰ：新しい計算テーマに何を選択するか？

素数以外に興味のあるテーマでは円周率や、暗号解読などが挙げられるが、分散が事実

上不可能であったり、必要とするプログラムがあまりにも現在とは異なるようなテーマは好ましくない。また、他のプロジェクトですでに行われていて、こちらのプロジェクトに参加する意味がないような場合も当然選択してはならない。

そこで、現在でも正しいと予想はされているが、未解決問題で、証明も、反例も発見されていない、コラッツ予想というものがある。そして、この予想の反例探索をメルセンヌ素数候補に対して行う事を考えた。

コラッツ予想

任意の自然数Nに対して、以下の手順を繰り返す。

Nが奇数であれば　N＝３×N＋１

Nが偶数であれば　N＝N/２

上の手順を繰り返した場合、どんなNでも最終的に１となり、１→４→２→１のループとなる

コラッツ予想のプロジェクトは海外のプロジェクトでも２つほどあるが、１から順にすべての数を確かめていくという方式のため、たかだか１７桁程度の数までしか探索を行われていない。そのため、メルセンヌ素数候補に対しての探索によって先に反例を見つける事ができる可能性は依然残されていると言える。そのため、単純にメルセンヌ素数を探索しようとした時のように、すでに計算されてしまった範囲の探索のために存在意義があまり無いという問題は生じないはずである。

Ⅱ：計算プログラムを作成する

今回のアルゴリズムは非常に単純なため、演算速度の影響が直接出てしまう。しかし、自作の多倍長ルーチンは、数値を文字列として保持しておく方式のため、あまり高速な演算を期待する事はできない。また、シフト演算なども利用する事ができないので、残念ながら、ここでは、提供されているルーチンを利用する事にした。

Ⅱ－１　反例をどのように判断するか？

コラッツ予想の反例の形式としては以下の２つが挙げられる

・１→４→２→１以外のループに陥ってしまう

・１に向かっていかないで、無限大に発散してしまう

この２つの形式のうち、１つ目はコンピューターによる計算で証明可能なのだが、２つ目は証明する事ができない、どこまで値が大きくなったら発散と判断するかが明確ではないからである（反例かもしれないという判断を下す事は元の数との比率などから可能かもしれない)。つまり、いくら値が大きくなっても、どこで１に向かって行くか分からないので反例であると断定はできないのである。よって、もし反例が存在したとして、その形式が２つ目の形式だった場合には数学的な証明を待つしかない。そのため、非常に残念だが、２つ目の形式の反例に対しての対策のために処理能力を消費することは避け、もう一方の形式の反例探索に力を入れることとした。

そこで、１からのループではないループに陥ってしまうという形式の反例をどのように判断するかを考えた。この形式の反例を証明するためには、計算の仮定で同じ値を通過した事を確認すればよいのである。つまり、Ｎが２回以上同じ値をとった事を確認すればよいのである。そのために、最初は通過したすべての数をファイルに保持しておき（配列では多倍長のため、すぐにメモリが不足してしまう)、１回手順を行うたびに現在のＮと比較するという方法をとった。しかし、時間に比例して比較する値が非常に高い割合で増加して行き、その比較部分のオーバーヘッドのため、プログラムの速度が致命的に遅くなってしまった。

では、もっと効率の良い比較方法はないのかと考えた時、すべて保存する必要はないのだという事に気付いた。

例　　**判明←X←X←X←X**

・・・・X←X←X←X←X←X←X←X←X←X←X←X←X←X←X←・・・・・

↑

（コラッツ算の試行の結果一度通過した値に戻る）

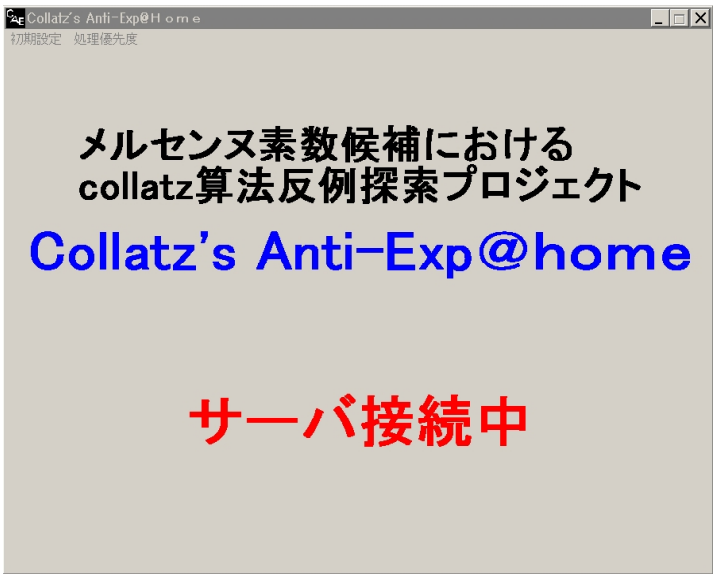
※太字は２回目の通過を示す　※Xはその値を記録している事を示す

例のようにコラッツ算の手順を繰り返した結果、一度通過した値に戻ったとしても、一定の間隔で値を記録しておけば、やがて記録しておいた値を通過する時に２巡目、つまり反例であると判断する事ができる。なので、記録する頻度を減らすため、計算回数の桁が上がる度に記録する事とした（１０回，１００回，１０００回で記録というような方法)。

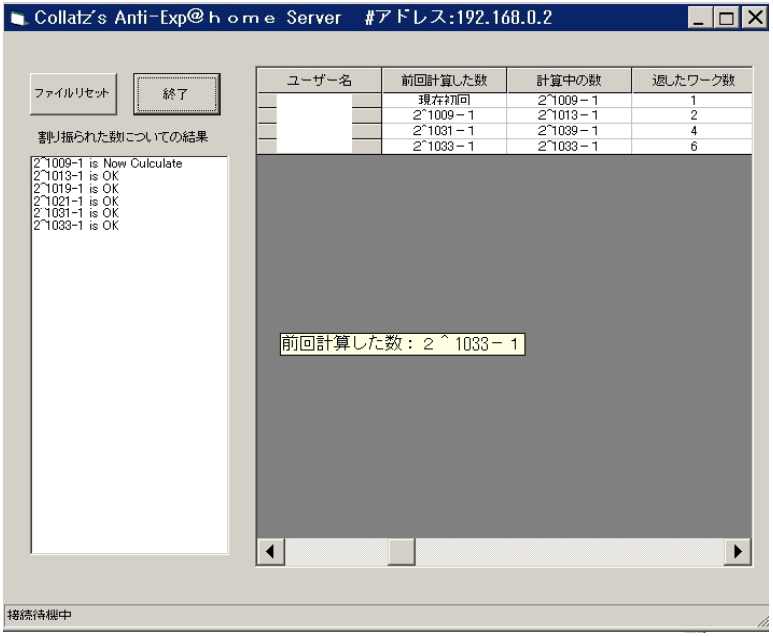
以前、自宅で探索した結果と、上記のプログラムをパソコン室のパソコン１０台で動かした結果を合わせ、２＾１００００までの候補の中に反例を見出すことはできなかった。よって、プロジェクトでの探索は２＾１００００から先の候補に対して行う事となった。

☆実行画面

〇クライアントアプリケーション

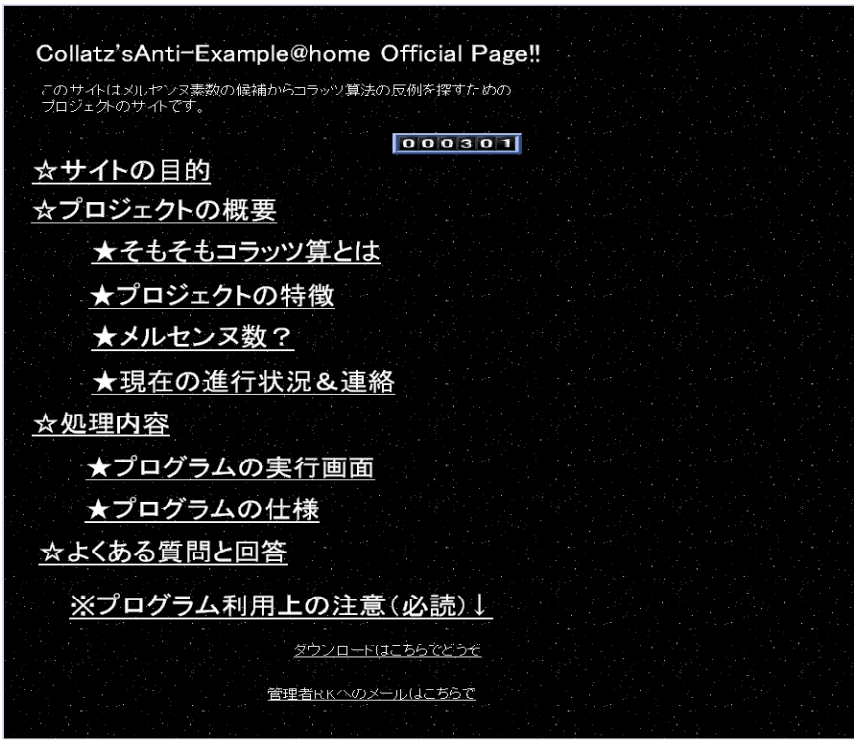


○サーバアプリケーション



※実際の参加者の情報が含まれるため、ユーザー名はふせることとする

○専用ホームページ（http://www12.ocn.ne.jp/~mygrid/colats_001.htm）



第７項　「プロジェクト遂行の歩み」

コラッツ算法の反例探索のアプリケーションを公開したところ、早速、参加者が現れ、探索を行う事ができた。この事は、初めて不特定の人物から余剰の処理能力の提供を受けるということが実現できたという事を意味する。

とにかく、プロジェクトはユーザーに受け入れられ、開始する事ができたのである。しかし、プロジェクトが開始できた事で満足してしまうのではなく、向上を目指して行かなくてはならない。

☆プロジェクト向上のための問題解決

Ｉ：プロジェクトの知名度の向上

まずは、大勢の人にこのプロジェクトの事を知ってもらわなくてはならない。そのために

・検索エンジンへの登録
・ホームページ宣伝ページでの宣伝書き込み
・グリッドを扱うサイトへの申告

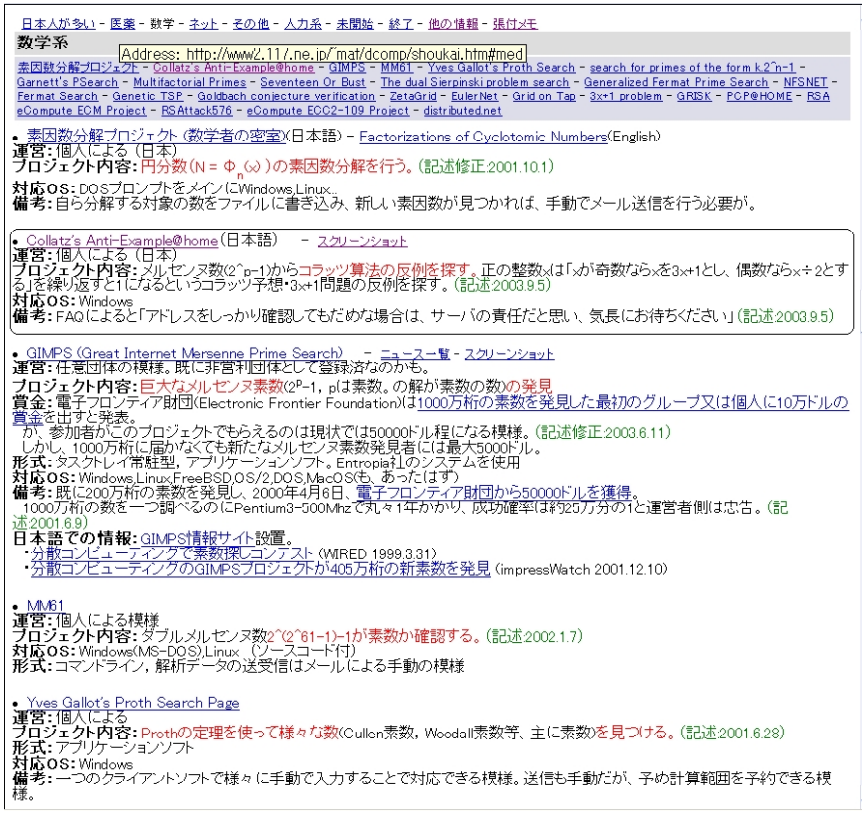
などの方法をとった。

その結果、参加者の増加はそれほど認めることはできなかったが、少しずつサイトアクセスの量がふえていった。

また、Ｍａｔ氏（さまざまなグリッドプロジェクトの紹介を行っているサイトの管理を行っている）に個人的に掲載の要請をしたところ、快諾を得る事ができた。↓

「各種分散コンピューティングプロジェクトの紹介」

http://www2.117.ne.jp/~mat/dcomp/shoukai.htm



Ｍａｔ氏のサイトを経由し、下のサイトにも紹介された。

「ＤＯＵＢＬＥ　ＳＬＡＳＨ／／」

http://www6.plala.or.jp/kozai/

上記の掲載の要請時、Ｍａｔ氏から次のような激励を受ける事ができたので、Ｍａｔ氏の許可のもと、改変なしで転載する。

「P2P となりますと、幅広く、私の守備範囲外となる部分も多いのですが、CPU パワーをメインにした、自動型 P2P アプリケーションの配布は初めてだと思います。

遂に待ち望んだ、日本オリジナルの、一般参加型で、自動型の、分散コンピューティングシステムが、神林さんの手により作られたんですね。感慨深いです。」

この言葉を受け、よりプロジェクトの意義を再認識する事ができたとともに、プロジェクトについての責任の重さを感じることとなった。

Ⅱ：ユーザーの意見のフィードバックと、プログラムの向上

プロジェクトの向上のためには、ユーザーの意見に耳を傾ける事も非常に重要な事である。

そこで、ホームページなどへの要望がいくつかあったので挙げる。

- **クライアントアプリケーションにバージョンをつけて区別して欲しい**
- **プロジェクトの進行状況をホームページで掲載するようにして欲しい**
- **毎回、ランタイムのインストールをするのでは、効率が悪いので、バージョンアッ**

プ用のパッケージを用意して欲しい

以上の意見を受け、それらをホームページに反映した

また、クレームとして

・インターネットに再接続を行ってから、サーバへ接続を行うと、今まで計算してきたワーク数などがリセットされ、初回扱いになってしまう

というものがあった。

これを受け、原因を追求したところ、サーバでのクライアントの認証に、誤まってＩＰアドレスを利用してしまっていため、インターネットに再接続した事によるグローバルＩＰアドレス変更を、ＩＰアドレスとニックネームの不一致と判断し、初回接続とするような命令を出してしまっていたのだと分かり、改善を行った。

第８項　プロジェクト開始を迎えての考察

行列計算プログラムの考察で述べた事の繰り返しとなってしまうのだが、やはり、まったく情報が存在しなかったという点が重要である。行列計算のプログラムでは、ＭＰＩなどを用いた分散処理の情報などであれば存在したのだが、こちらの情報は、大学などの研究などを含めても皆無であった。おそらく、日本国内の個人では、大学や研究施設を含めても、Ｐ２Ｐコンピューティングのプログラムは初めての試みであろう、また、実際に公開を行って、プロジェクトを遂行しているという所まで限定するならほぼ確実に断言できる。そういった点ではこの研究の意義は非常にあったと言える。（データ送信が手動などのプロジェクトは存在する）

当然ながら、この研究は、プロジェクト開始によって終わるものではなく、引き続きプロジェクトを続けていくことにより、研究内容は発展していく。つまり、研究はこの段階で初めて始ったとも言える。そして、現在の、情報がまったくないという状況を変えるため、また、グリッド技術の一般への普及のため、技術公開を行っていかなくてはならない。

そうなれば、技術そのものが難解なわけではないのであるから、一般の、プログラミングを行う人々でも簡単にこのようなプロジェクトを始める事ができるようになる。そのための一歩となることができれば、非常に喜ばしいことである。

最後に、現在、プロジェクトへの参加者はたったの５人程度であるが、現在の参加者のためにも、これから参加人数を増やす事はもちろんの事、改善を行っていかなくてはならない。

第３章　次世代のグリッド技術についての考察

現在のボランティアコンピューティングの状況を考えた時、Ｓｅｔｉ@ｈｏｍｅやＧＩＭＰＳなどように、すでに大勢の人々が参加しており、非常に知名度も高いという、圧倒的に大規模なプロジェクトがいくつか存在する。それらに対して、日本国内のプロジェク

トはまったく歯が立たないというのが現状である。そのため、日本国内の人間ですら大部分が日本国内のプロジェクトではなくそういったプロジェクトに参加してしまっている。

国内の計算資源が海外のプロジェクトによって利用されてしまっているこの状況を、ＩＴ国家を目指す日本は認めていてよいのであろうか、そこで、その状況を打破したいと考え行動を起こすことを決意した。

そのための第一段階が、この研究中のボランティアコンピューティング的方向性の部分である。

私はこの研究の中で実際に日本国内の個人として始めての自動型グリッドプロジェクトを立ち上げる事ができた。しかし、このプロジェクトでは立ち上げる事で限界であり、私の目指す目標を達成するには程遠いと言うしかない。

そこで、これからのプロジェクトの改善を考えるにあたって、大規模なプロジェクトがなぜそれほどの知名度を得たのかを考えた。その結果、それらのプロジェクトは魅力的なテーマを持ち、ユーザーにとって参加の意欲を与える力があるからこそ、そういった地位を築くことができたのだという結論に至った。

そこで、そういった大規模なプロジェクトに対抗するにために私は下のような要素を挙げる。

Ⅰ：それらのプロジェクトを越える独創的な計算テーマ
Ⅱ：ユーザにとって魅力となるサービスの提供
Ⅲ：社会貢献の性格を強く持つ計算テーマ
Ⅳ：根本的な視点を変えたプロジェクト

上の４つの要素の中で、**Ⅱ**は基本的に企業などにとっての手法であると考えられる、よって、私の研究で強調する部分にはなり得ない。しかし、他の３つの要素については十分検討するに値する。

まず、「Ⅰ：独創的な計算テーマ」と「Ⅱ：社会貢献の性格を強く持つ計算テーマ」、についてだが、結論を先に述べてしまうと、これからの研究において、その部分を考えるのは私自信である必要はないと私は考えている。

Ｓｅｔｉ＠ｈｏｍｅやＧＩＭＰＳにおいても同様であるが、実際にプロジェクトのためのプログラムやネットワーク基盤の構築を行っている人間と、計算するテーマについての詳細を考える人間は同じであるはずがない。どちらもその分野のエキスパートでなければ、あれほどの高度で大規模なプロジェクトを行う事は実現できないはずだからだ。そのため、私は入学後すぐに他の学類も含め大学内の人間に当たり、どんな計算テーマが必要であるか調査し、協力してもらえる者を探して新しいプロジェクトを立ち上げる。そういった点でも総合大学である筑波大学は私のやりたい事を実現するためには最高の場所なのだ。

しかしながら、協力者を得て良い計算テーマとそれについての知識を得たとしても、計算テーマが良いだけでは大規模なプロジェクトに対抗する事はできないであろう。あまりに知名度の差が圧倒的である事に加え、テーマの良し悪しの差もそれほど付く事は期待できないからだ。そこには、新しい視点が必要となる。つまり、Ⅳの”**根本的な視点を変えたプロジェクト**”を考えなくてはならないという事である。

そこでどういったように視点を変えるかだが、私はどういった計算資源を利用するかという点で視点を変える。つまり利用する計算資源はパソコンだけでよいのであろうか、という点についての視点変更である。

事実、私も資料１の研究ではパソコンを計算資源として利用しているが、これから先を見すえて考えていく上で､他の計算資源の利用を考える事が重要であると私は考えている。その中で私が挙げるのが、携帯電話の計算資源としての利用であるのだ。そしてそれにはいくつかの理由がある。

まず携帯電話であれば、パソコンでの参加のように、肩ひじをはらずとも、手のひら感覚のボランティアとして多くの人に参加してもらえるはずであろうと考えるからである。

次に、最近の携帯電話の著しい処理能力の向上がある。デジタルカメラ並みのカメラを搭載したものや、３Ｄ描画が行えるものなどが登場し、さながら小さなパソコンとすら言えるかもしれない。加えて、現在の携帯電話の普及率は非常に高く、日本国民の２人に１人は持っているという状態にまでなっている。

そして最大の理由は、携帯電話への定額制パケット通信の導入のきざしである。

パソコンの世界でボランティアコンピューティングが成立したのは、常時・定額接続のブロードバンド回線の普及が大きな理由である事はまず間違いない。そして、その事を携帯電話においても当てはまるとするならば、携帯電話での定額制の導入にも非常に大きな意味があるはずである。実際にＰＨＳではあるが、ＤＤＩなどではすでにメール使い放題などのサービスが開始されており、ＤＯＣＯＭＯなどもその後を追いかけ導入の意思を表明している。これらを見る限り、その他のメーカーでも定額使い放題のサービスは次第に増えてくるはずである。

以上のように、携帯電話利用のアイデアの追い風となる要素をいくつか挙げたが、依然解決しなければならない問題点も多い。

１つは、厳しいアプリケーションの機能制限である。例えば、Ｄｏｃｏｍｏのｉアプリは当初セキュリティのため､携帯内の個人情報にまったくアクセスする事ができなかった。しかし、５０５シリーズ登場によってトラステッドｉアプリという仕様が利用できるようになり（現在ではまだ認可の行われた業者に限られている)、電話帳などの情報を利用する事も可能となってきている。このように、アプリケーションへの制限はユーザーのニーズにより次第に緩和されていくので、プロジェクトに必要なアプリケーションによる自発的な通信が可能になるのにそう時間はかからないであろう。

次に、処理能力の問題が挙げられる。しかし、携帯電話でのボランティアコンピューティングは当然低い処理能力を集める事を想定して行われるであろうから、私はこの事を問題視する必要はないと考える。

実は盲点となりやすいのだが、重要な問題に携帯電話のバッテリーの問題がある。プロジェクトのために、継続的に携帯電話の処理能力を利用するとなると、電力の供給を外部から受けていないので、携帯電話の電池はすぐに消費されてしまう。この問題が露呈しユーザーが携帯電話を利用しようとする時に使えないというような状況が発生してしまえば、ユーザはプロジェクトへの参加を中止してしまうであろう。しかしながら、この問題がすぐに解決されるような見込みを見出す事は現状ではできない。よって、ハード的な改善を待たずして、この問題を解決する方法を考えてみた。

上の問題を、バッテリーから長時間、継続的に電力を得る事ができるように改善が行われる場合（バッテリーの改善）以外で解決するには、計算を行う時に外部から電力の供給を受けるしか解決方法はない。そこで、私が考えたこれをもっとも簡単に実現できる時間帯が、自宅などで充電を行っている時間である。この時間であれば、多少の充電時間の延長は必要となるかもしれないが、プロジェクトのために電力を消費しても問題はないはずである。そして、こういった処理を行うためには、充電開始・終了を感知するイベントが必要となる。（私の知る限りでは、現在はまだ用意されていないようである）

もう一つ次世代のグリッドに関して私が考えた事が、近い将来パソコンがモニターと最低限の通信機器しか必要としない時代が必ずやって来るだろうという予想である。この事が何を意味しているのかというと、ユーザは一つの巨大な仮想コンピューターにログインする事によって、パソコンを利用するというような時代が必ず来るという事である。

私が考える形態では、記憶媒体さえユーザーそれぞれ別で扱うようにすれば、利用していない計算資源は他のユーザーが利用できるようになるので、それぞれが自宅にパソコンを持つ形態と比べ、計算資源を圧倒的に効率的に利用する事ができるはずである。（現在のボランティアコンピューティングでは、社会全対の家庭用パソコンの処理能力の１％すら利用していないと言われている）。そして、その仮想コンピューターが現実に完成する事となったとしたら、おそらく１台のコンピュータではなく、多数のコンピュータが結合されたネットワークの形態をとるはずである。そのネットワークによる仮想コンピューターを実現する技術が他ならぬグリッドなのだ。

まず、この形態を発想したきっかけだが、高校での”作業フォルダ”についての勘違いがきっかけであった。

私が所属する高校では、生徒がそれぞれアカウントを持っており、そのアカウントさえ覚えていれば、パソコン室のどのパソコンを利用しても、自分が保存しておいたファイルを保持している”作業フォルダ”を利用する事ができる。これはよく考えれば、校内のサーバに一括して置いてある生徒それぞれのデータを呼び出しているだけなのだが、私には最初それが不思議に思え、それぞれの生徒用にパソコンが用意されているのと同じ事なのか、と勘違いしてしまった。しかし、この勘違いを発展させ皆が同一の巨大コンピュータを利用していたらどうなるのだろうかと考え、それにグリッドが結びついた事によって、この形態に行きつくこととなった。

そこで、この形態について今までに得てきた知識を活用し、どういったような変化が起こり、どういった長所・短所が生まれるのかを自分なりに考えてみた。

まず実現により予想される変化だが、私はソフトウェアを購入するという概念そのものが無くなってしまうのではないかという点を挙げる。

ソフトウェアを配布する者が、共有資源である記憶媒体の上にソフトウェアをインストールしておき、その領域のアクセス権を販売・提供する事により、ユーザは余計な手間がかかる事なく利用する事ができる。（同じ領域を複数のユーザが利用するか、それぞれ別の領域を用意するかという違いは生じるかもしれない）この事は、ソフトウェアの販売という概念が、すべてサービスの提供に置き換えられてしまう事を意味する。

もう一つ挙げたいのは、どんな場所でもアカウントさえ持っていれば、自分のパソコン

を呼び出せるので、ノートパソコンなどを持ち運ぶという事が無くなるであろうという点である（公衆電話ではなく、公衆パソコンが街中に設置されるかもしれない)。

次に、長所について述べる。やはり一番の長所としては計算資源を非常に有効に利用できる事が挙げられる。待機状態のパソコンが存在しなくなるのだから、この事は当然である（共有の処理能力を分割し、それぞれのユーザに割り振っておくというような方式で利用された場合は除く、しかし全ての共有の処理能力を利用している人数で等分して利用できるというような場合ならばこの長所が生まれる)。

短所として挙げるべき点としては、ネットワークへの通信経路が存在しない場合、パソコンという存在自体が成立しなくなってしまうという点が挙げられる。この点を考えると、ネットワークが地球上のどのような場所でも利用可能にならない限りは、全てのパソコンがこの形態になるのではなく、現在のような通常のパソコンと新しい形態のパソコンが両立するという状態が続く事になるだろう。

最後にこの形態の実現において問題となるであろう事象を考える。

まず第一にセキュリティ面が挙げられる。速度や効率といった問題は当然重要な要素だが、世界中の人々がコンピュータ資源を共有するのであるから、セキュアなネットワークを構築する技術が確立されなければユーザは安心して利用する事はできない。

また、利点を最大限に利用しようとすればユーザが自分のアカウントによってパソコンを呼び出せる方式になるであろうと考えるのだが、どこからでもそのアカウントがあればパソコンが呼び出せてしまうとなると、そのアカウントの認証の方法は文字列の羅列という方法では、いずれ漏洩が生じてしまう事は明らかである。そのため、指紋や眼球による認証などのように、本人しか絶対に利用できないような対策が行われなければならない。

他の問題としては、高速な通信ネットワークの実現が不可欠だという点が挙げられる。

現在、光通信の常時・定額のサービスも一般に利用され始めているが、たとえ光通信ですべてのネットワークが構築されたとしても、パソコン内部のデータ伝達速度には遠く及ばない。このような状況では実現されたとしても、現在のパソコンの速度には及ばず、ユーザは依然として現在の形態のパソコンを利用し続けてしまう事になるであろう。しかしながら、この部分は情報の分野を超えてしまうので、他の分野の人々との協力を回りながら解決していかなくてはならない。

第４章　今までの研究を終えて

論文の各部分でそれぞれ考察を述べたので、ここでそれらを踏まえ、研究を振り返る。

☆考察

「一般の人々でも利用可能なグリッド技術の実現」についての研究を行って来たが、アルゴリズムの工夫や、プログラミングの技術についてはまだまだ未熟な部分も多く、改善の余地がある部分は多い。

例　メルセンヌ素数発見のための数列にも実は依存関係があり、探索のために、クライアントへ判明している最後の数列の値を渡しておかなくては、すべてのクライアントが同

じ素数を探索し直すのと同じであったという改善点や、巨大数の乗算などにおけるＦＦＴの利用、大きな乗法に対しての平方乗法の利用　など

しかし、実際に、難解な技術は極力利用せずに、また、金銭などを必要とする事もなく、グリッド技術を利用するプログラムを製作する事ができた。そして一人の高校生である私がそれをやってのけたという事は、一般の人々でも利用可能である事を意味するはずだ。つまり、研究の結論はこうである。

大学や研究機関などに属さない、一般の人々でもグリッド技術は利用可能である

この事を、本研究によって証明する事ができたと私は思う。また、研究の過程で、私の長年の夢であった、日本の個人では初めての自動型Ｐ２Ｐプロジェクトを開始する事もできた。これについての意義は大きいと思われる。そして、これについては、プロジェクト遂行と共に、引き続き、調査を行っていく。

次に、性能評価という面について考える、そこでは調査結果が決して高い値ばかりを示したわけではなかった事に注目したい。私が示した、グリッド技術が利用可能である事は、技術として利用可能なのであって、それが真に恩恵を受けるという意味で利用可能とするには、綿密に、ネットワークの構築や、プログラムの作成を行い、さらに高度な研究を続けていかなくてはならない。

最後に、研究結果がそのまま実際に産業的・技術的に利用可能かどうかという事は別として、大学での研究の基礎となるグリッドのプロジェクトについてのノウハウを身につける事ができたはずだと私は考える。この経験を生かし、また、さらに大学で数学的な知識や数値計算についての知識を高める事によって、現在の研究を続け高めて行くつもりである。そして、次の段階として

「携帯電話を利用したボランティア　コンピューティング」

「新しいパソコンの利用形態」

の実現を目指して行きたい。

○主要参考文献

・「並列処理コンピュータ」、小畑　正貴著、東京電機大学出版局、２００１年
・「超並列コンピュータ入門」、野口　正一監修、オーム社、１９９２年
・「数学とっておきの１２話」、片山　孝治著、岩波ジュニア新書、２００２年
・「素数の不思議」、堀場　芳数著、講談社、１９９４年
・「図解雑学　フェルマーの最終定理」、富永　裕久著、１９９９年
・「パソコンで挑む円周率」、大野　栄一著、講談社、１９９１年
・「高等学校数学科用　数学Ｃ」、藤田　宏著、東京書籍、２００２年
・「コラッツの問題」、梅田敏夫、愛知教育大学、２００２年
・「素数生成アルゴリズムの調査・開発　調査報告書」
情報処理振興事業協会、２００３年

○参考ウェブサイト

"各種分散コンピューティングプロジェクトの紹介"
http://www2.117.ne.jp/~mat/dcomp/shoukai.htm

"グリッド研究センター"
http://www.gtrc.aist.go.jp/jp/

"グリッド協議会"
http://www.jpgrid.org/ Grid Consortium Janpan,

"GIMPS"
http://www.mersenne.org/works.htm

"家サーバー・プロジェクト"
http://www.ieserver.net/index.html

"On the 3x+1 problem"
http://personal.computrain.nl/eric/wondrous/ ,

"Visual Basic Library"
http://www5d.biglobe.ne.jp/~tomoya03/index.html

"Visual Basic Q & A 掲示板"
http://homepage1.nifty.com/MADIA/index2.html

"VB レスキュ-（花ちゃん）"
http://www.bcap.co.jp/hanafusa/index.html

"VB で使用できる DLL の作り方"
http://www.angel.ne.jp/~mike/vb_dll/index.html

"月刊プログラム！ Visual Basic For Excel Vol_18"
http://www5b.biglobe.ne.jp/~kouta_y/news/newsvba/vba18/newsvba18.htm

"VB Tips"
http://sumire.sakura.ne.jp/~satoyam/vbtips/index.html

"ふぢのＶＢ小技集"
http://www.donnk.com/Fudi/VBtips/VBtips02.html ,

"ＯＣＮホームページ"
http://iba.er.cyberwing.co.jp/hserver/SITE
"多項式時間素数判定アルゴリズムについて AKS アルゴリズム　PRIMES is in P"
http://www.h4.dion.ne.jp/~a00/ms_project_jp.html
"素数のページ"
http://www.crt.or.jp/~kokochi/sosu.htm#moku
"The Prime Glossary"
http://primes.utm.edu/glossary/page.php?sort=WheelFactorization
"暗号と整数の世界－べき乗剰余演算とフェルマーの小定理"
http://aitech.ac.jp/~koikelab/webp/cipher/cipher_03.html
"暗号と整数の世界－擬素数と素数判定"
http://aitech.ac.jp/~koikelab/webp/cipher/cipher_07.html
"アセンブラを使わない Ｃ＋＋による多倍長計算ライブラリ"
http://hp.vector.co.jp/authors/VA018507/japanese.html
"FFT と AGM による円周率計算プログラム"
http://momonga.t.u-tokyo.ac.jp/~ooura/pi_fft-j.html
"多倍長演算ライブラリ"
http://www5.airnet.ne.jp/tomy/cpro/longman.htm
"南中野パソコン教室のパソ楽のページー VB.NET はじめる－多倍長計算"
http://www.tokyo-pax.co.jp/pasovb5.htm
"点と線の部屋－多倍長整数クラスの設計"
http://plaza16.mbn.or.jp/~graph_puzzle/2no1.htm
"精度拡張クラス"
http://kilin.u-shizuoka-ken.ac.jp/software/lb/lb.html
"確率的な素数判定方法"
http://www.geocities.co.jp/SiliconValley-PaloAlto/4639/page_PrimeNumber.htm
"Lucas の判定法"
http://www.ese.yamanashi.ac.jp/~morisawa/sc2001/Lucas.html
"巨大素数と計算機－ Mersenne（メルセンヌ）素数”
http://mailsrv.nara-edu.ac.jp/~asait/c_program/sample/mersenne.htm
"素数の判定（Lucas テスト）"
http://alfin.mine.utsunomiya-u.ac.jp/~niy/algo/l/lucas.html
"猫の知ったかぶり－ Shell 関数で起動したウインドウの終了を待つ"
http://www.bekkoame.ne.jp/~ilgg/VBMain/VB/Process/WaitShellPGCloset.html
"Lookin' for － Visual Basic 系？－ VB でタスクトレイに常駐するソフトを作る"
http://hp.vector.co.jp/authors/VA021282/vbtech/tasktray.html
"FTP Exchange"
http://www.toyota.ne.jp/~kawauso/web/
"ConditionGreen －自宅サーバー構築"
http://ae101kai.zive.net/myhomeserver.html

"竜の情報館ー Net Security"

http://ryulife.com/net/za/basicinfo.html

"WindowsXP での Path or File Not Found 障害対策"

http://pcweb.mycom.co.jp/news/2003/04/18/20.html