Vol.12, No.1 2013

全国共同利用版

広報

新汎用コンピュータシステム運用開始

「新しい汎用コンピュータシステムの概要」河原達也、森信介、赤坂浩一●「ホスティング・ホームページサービスの改定について」赤尾健介、赤坂浩一、針木剛、秋田祐哉、森信介、河原達也【解説】「CrayPatによる性能解析」武田大輔●「GPUを用いたFEMアプリケーションの高速化」大島聡史【研究紹介】「交通計画分野の高速処理演算需要」山﨑浩気

ISSN 1347-3581

京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

Vol. 12, No.1 号の発刊にあたって

京都大学学術情報メディアセンター
岩下　武史

今号では、2012年末にリプレースされました新汎用コンピュータシステムに関する記事を中心に編集させて頂きました。汎用コンピュータシステムの主な使命は、大学に求められている研究・教育活動や研究成果の公開による社会貢献を支える情報基盤を提供することにあります。近年では、これらの情報基盤を安定的かつ省電力で運用すると共に、情報基盤で扱われるデータの安全性を高いレベルで維持する堅牢なシステムの構築が求められています。一方、大学の情報基盤を構築するための予算には限りがあり、無尽蔵というわけでありません。そこで、大学の情報基盤やそれに伴うサービスとして必須なものが何であり、限られた予算で提供し得る最大限のサービスレベルはどの程度かを見極めた上で、ハードウェアの機能の取捨選択や提供サービスを規定する必要があります。こうした情報基盤に関するサービスや機能の取捨選択は他機関においても多かれ、少なかれ存在するものと考えられます。本号で紹介する汎用コンピュータシステム導入にかかわる「京都大学の選択」が、他の大学や機関における情報基盤構築の参考例となれば幸いです。

本号では、まず新汎用コンピュータシステムの仕様策定に携わった河原氏、森氏、赤坂氏から、導入システムの概要について寄稿を頂きました。今回のシステムの特徴はプライベートクラウド型のシステムを中心に据えながら、学外のクラウドサービスも活用している点です。具体的には落札業者である富士通社の館林データセンターをデータや機能のバックアップ先として活用し、本学の計画停電時や災害等の緊急時においても、最低限のサービスが可能となるようにシステムが設計・構築されています。次に、赤尾氏、赤坂氏、針木氏、秋田氏、森氏、河原氏より、同システムを用いたサービスに関する解説を頂きました。サービスの中心となるVMホスティングサービス、ホームページサービス、ストリーミングサービスの概要と利用法が述べられています。

また、本号では昨年の5月にリプレースされたスパコンシステムに関する記事も3件掲載しています。スパコンシステムについては、納入ベンダが変更となったため、コンパイラの利用法やジョブの実行・管理法についても変更がありました。しかしながら、前システムと変わらない高い稼働率でご利用頂いています。システムも2年目に入り、今後はシミュレーションプログラムのチューニングや京大センターとしては初の導入となったGPUの利用が進むと期待しています。そこで、これらの利用を支援する記事を寄稿いただきました。納入ベンダであるクレイの武田氏からは性能改善ツールであるCray Patの使用法と性能改善に関する解説を頂きました。Cray XE6上でのチューニングにご活用いただければと考えます。また、大島氏からは、GPUを活用したアプリケーション事例を紹介いただきました。GPUに関する基本的な解説に加えて、具体的なシミュレーション事例として、有限要素解析におけるGPUの利用に関して述べられています。GPU利用に関する参考となれば幸いです。また、山崎氏からは交通シミュレーションに関する事例紹介を頂きました。実データに基づいたシミュレーションを行い、社会システムと連携する事例となっており、シミュレーション技術の新しい展開として皆様の参考になれば幸いです。

本センターのシステムを皆様の研究、教育にご活用いただけるようセンター教職員も尽力をしていきますので、今後ともご利用、ご支援の程、よろしくお願いします。

新しい汎用コンピュータシステムの概要

大学の情報基盤を支えるクラウド型システムにむけて

河原達也[1]　森信介[1]　赤坂浩一[2]

[1]京都大学学術情報メディアセンター　[2]京都大学情報部

平成 24 年 12 月末にリプレースされた新しい汎用コンピュータシステムについて紹介する。本システムは、大学のコンテンツ・データベースを扱う多種・多数のサーバ環境を仮想マシン（VM）の形で提供するとともに、ホームページサービスや電子メールサービスなどの情報基盤を担うものである。これらのサーバの多くはミッションクリティカルで、24 時間 365 日の安定運用が要求されるとともに、省電力性能そして災害対応も重要となっている。今回のリプレースに際しては、このような点を考慮して仕様を策定した。その結果、前回のシステムに比べて、２倍以上の計算機性能（ＣＰＵコア数・メモリ容量・ディスク容量）を、より少ない消費電力で実現するとともに、最新鋭のデータセンター（富士通・館林システムセンター）にバックアップシステムを構成するに至った。

1　システム更新の背景

汎用コンピュータシステムは、学術情報コンテンツや研究資源アーカイブなどの学術研究・教育等に関する多様な情報の蓄積・管理・交換・発信などを担う情報環境基盤として位置づけられている。これは、インターネットの広帯域化により大容量・多様なコンテンツが交換・発信できるようになったこと、さらに国立大学法人化後、研究成果や教育内容等を広く学内外に公開することが求められるようになったこととも適合する。

このような流れ・ニーズがますます加速していく中で、前システムの性能・容量の限界がみられたので、平成 24 年 12 月末にシステムのリプレースを行った。本システムが担うサービスや学術データの重要性、ならびに東日本大震災後における各地の経験もふまえて、サーバのバックアップ体制も含めた信頼性・堅牢性ならびに省電力性を重視した。

2　新システムの仕様の概要

今回導入したシステムの概要は以下の通りである。個々のシステムについて次章以降に説明する。

- 汎用サーバシステム
  ノード数 128 以上、ＣＰＵコア数 2048 以上で構成されるクラスタ型計算機
- ストレージシステム
  物理容量 450TB 以上のディスクアレイ装置
- コンテンツサービスシステム
  上記サーバの仮想マシン（VM）として構成
  - ホスティング用サーバ
  - ホームページサーバ
  - データベースサーバ
  - ファイル共有サーバ
  - 映像配信サーバ
- 電子メールサービスシステム
- ネットワークシステム
- バックアップシステム
  データを定期的に保全するとともに、本学の停電時・災害時などにも基幹的なサービスが継続できるようにする。

3　汎用サーバシステム

以下の要求仕様からなる。

- ノード数 128 以上、ＣＰＵコア数 2048 以上（前回の２倍）で構成されるクラスタ型計算機
- ノード当りプロセッサ数２以上、プロセッサ当りのＣＰＵコア数８以上（前回の２倍）
- プロセッサは Intel 64 アーキテクチャ
- ノード当りの主記憶メモリは、128GB 以上（前回の８倍）
- ノード当たりの消費電力が 270W 以下（前回を下回る）
- サーバハードウェア全体の消費電力が 36kW 以下（前回を下回る）
- 仮想マシン（ＶＭ）ソフトウェアとして、VMWare 48 式と KVM 80 式
- 仮想マシン管理ソフトウェアによる一元管理

富士通製の PRIMERGY CX250 （ＣＰＵ: Xeon E-2650L(1.8GHz;8 コア)×2、メモリ:128GB, SSD:100GB）が 128 ノード分導入され、うち 48 ノードには VMware vSphere 5 Enterprise、80 ノードには Red Hat Enterprise Linux (KVM)が導入された。

4　ストレージシステム

以下の要求仕様からなる。

- 物理容量合計 450TB 以上（前回の約２倍）のディスクアレイ装置
- うち 150TB 以上を 15000rpm 以上のＳＡＳまたはＦＣディスクドライブで構成（前回はない）
- RAID レベル 0+1, 5, 6 のいずれかでディスクアレイを構成
- SPC Benchmark-1 の IOPS が 60,000 以上、もしくは SPECsfs2008_nfs.v3 が 60,000 以上
- ストレージシステム全体の消費電力が 14kW 以下（前回を下回る）
- バックアップシステムにデータ（最大 20TB）をバックアップする機能

富士通製の ETERNUS NR1000 F3240 が２クラスタで導入された。これには、ＳＡＳ(600GB) ディスク 252 個（計 147TB）とＳＡＴＡ(3TB)ディスク 108 個（計　324TB）が含まれ、インタフェースとしては FibreChannel-8Gbps が ８ 口、10GBASE が２口ある。

5　コンテンツサービスシステム

以下のサーバが、汎用サーバシステム及びストレージシステム上の仮想マシン（ＶＭ）として構成される。

5.1 ホスティング用サーバ

「VMホスティングサービス」として、仮想マシン（ＶＭ）をサーバ単位で利用者にＰａａＳ型で提供するとともに、ＷｅｂなどのＳａａＳ型のサービスを提供するものである。

Red Hat Enterprise Linux 6 を 190 式、Microsoft Windows Server 2008 を 10 式用意している。

5.2 ホームページサーバ

「ホームページサービス」として、仮想マシン（ＶＭ）上のＷｅｂサーバソフトウェアでバーチャルホスト機能を利用したＷｅｂホスティングサービスの提供を行う。これに加えて、利用者のＷｅｂコンテンツを管理するファイルサーバや、Ｗｅｂホスティングと連動した利用者ドメイン毎のメールホスティングサーバから構成される。

5.3 映像配信サーバ

「ストリーミングサービス」として、ホスティング用サーバやホームページサーバと連携して、映像・音声コンテンツのストリーム配信サービスの提供を行う。

Real Networks Helix Server Unlimited Mobile Edition が導入された。

5.4 データベースサーバ

ホスティング用サーバ及びホームページサーバで利用されるデータベースサービスの提供を行う。

MySQLサーバ3式とOracleアプライアンスサーバ1式から構成される。

5.5 ファイル共有サーバ

学内外で業務・連絡用のデータの保存・管理・受け渡しに利用するものである。

Proself Enterprise Edition 2式及びFreeNAS 40式が導入された。

6 電子メールサービスシステム

本学のすべての教職員が用いる電子メールシステム（「京都大学教職員用メールシステム」KUMail）である。今回のリプレースでは、バックアップシステムと同じ設備内で運用される。これにより、24時間・365日対応の迅速な保守が可能になった。

以下に主な仕様を挙げる。

- 単一ドメインで15,000ユーザアカウント以上をサポート
- 20TB以上のメールスプール用ディスク領域
- 各ユーザアカウントのスプールの上限を、1GB以上20GB以下の範囲で任意に設定・変更可能
- Webメールとしての閲覧・送信に加えて、POP3とIMAP4をサポートし、転送設定を行える

ディープソフト社のMailSuite Academyが導入された。

上記に加えて、本学で管理するサブドメインをホスティングするメールホスティングシステム、メールの添付ファイルを分離するメールプロキシシステムも導入されている。

7 ネットワークシステム

本システムの要素間及び学内ＬＡＮと接続を行うとともに、負荷分散・監視・インターネット中継（ネットワークプロキシ）・VPN接続などのサービスを行う。

具体的には、以下から構成される。

- 汎用サーバスイッチ Catalyst4510R+E
- 負荷分散装置　IPCOM EX2500LB　4式
- 汎用サーバ監視装置 McAfee Network Security Sensor M-2850
- HTTPプロキシサーバ
- SOCKSプロキシサーバ
- NATサーバ　13式
- PPTPサーバ　13式
- SSTPサーバ　2式
- OpenVPNサーバ　2式
- SSHポート転送サーバ
- SSLリバースプロキシサーバ
- ログ管理サーバ

8 バックアップシステム

データを定期的に保全するとともに、本学の停電時・災害時などにも基幹的なサービスが継続できるようにするためのものである。

本システムを受注した富士通の館林システムセンターに構築されている。最寄りのSINETのデータセンターと1Gbpsの専用線で接続されている。

ディスクアレイ装置を有し、4のストレージシステムと連携して、5のコンテンツサービスシステムのデータ（最大20TB）のバックアップを定期的かつ自動的にとることができる。

また、サーバコンピュータを有し、5のコンテンツサービスシステムのサーバ（仮想マシン最大48式）のバックアップを収容することができる。これらは、本学の計画停電時、または災害等の緊急時に、代替サーバとして機能するように設計・構築を行っている。

9 おわりに

以上、今回のシステム更新により、システム性能の向上（前回のほぼ2倍）と一層の省電力化を両立することができた。

さらに、本システムに構築されるサーバのミッションクリティカル性と蓄積されるデータの重要性を鑑みて、民間の最新鋭のデータセンターを活用したバックアップシステムを構築し、24時間365日の保守体制、高信頼性・高可用性を実現した。これは、全国的にも先進的な取り組みであると考えている。

このように本システムは、大学における様々なデータ指向のサーバに対するクラウド型の情報基盤を提供するとともに、本システム自身も民間のクラウド型サーバを活用した二重のクラウド型システムとなっている（図１）。

今後も様々な利用者の意見をうかがいながら、最適なシステム・サービスの設計・構築につとめていきたい。

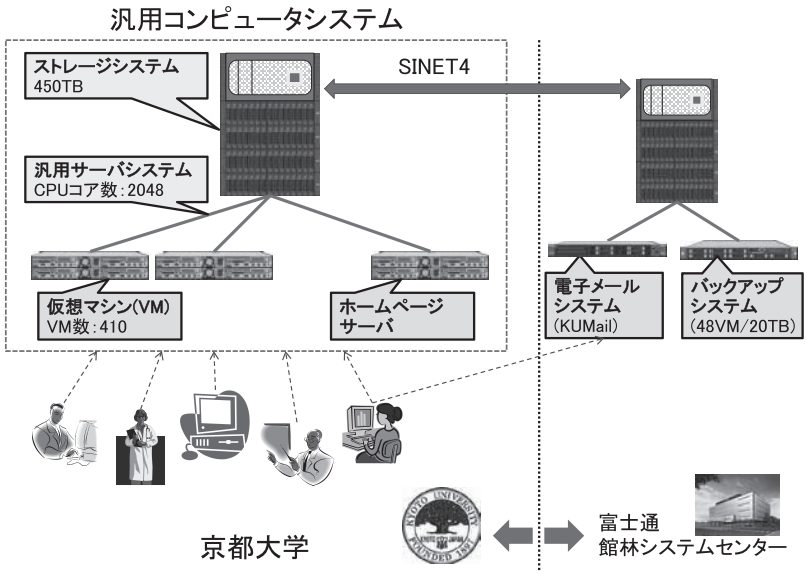
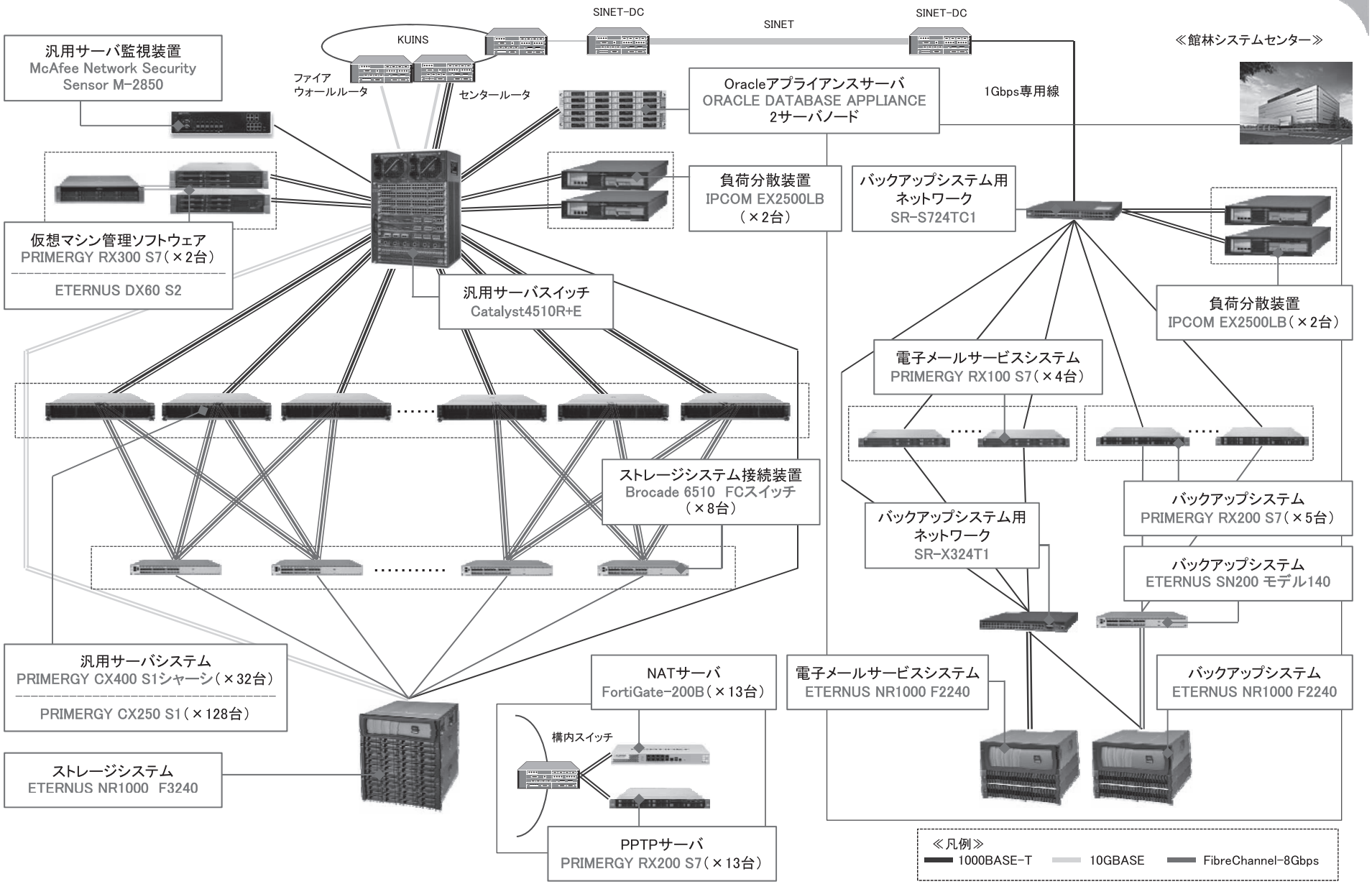


図１　システムの概要

汎用コンピュータシステム全体構成図



ホスティング・ホームページサービスの改定について

改定のポイントとサービスの利用方法

赤尾 健介† 赤坂 浩一† 針木 剛† 秋田 祐哉‡ 森 信介‡ 河原 達也‡

†京都大学情報部 ‡京都大学学術情報メディアセンター

1 はじめに

学術情報メディアセンターでは2012年12月に汎用コンピュータシステムをリプレースした．それに伴い，2013年4月1日より，情報環境機構（情報環境支援グループ学術情報基盤担当）で提供しているホスティング・ホームページサービスの内容と負担金金額を改定した．本稿では，改定された内容と，改定後のサービス利用方法について説明する．

2 サービスの概要

ホスティング・ホームページサービスでは，Web を活用した情報発信を行うための3種類のサービスを提供している．具体的には，占有のサーバ環境を提供する VM ホスティングサービス，Web サイトの公開環境を提供するホームページサービス，マルチメディアコンテンツの公開環境を提供するストリーミングサービスである．サービスのラインナップと概要を図1に示す．

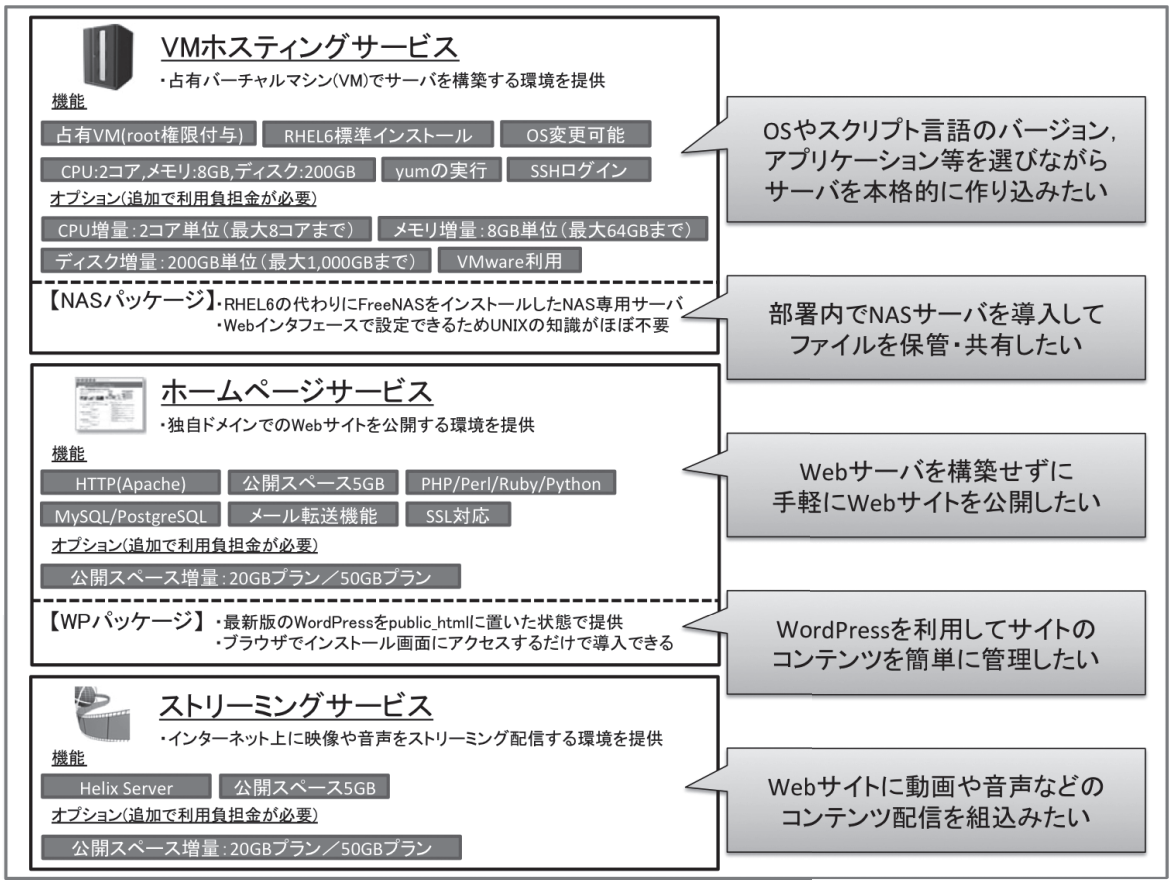


図1．サービスのラインナップと概要

3　新しくなったサービス

2009 年より 4 年間サービスしてきたホスティング・ホームページサービスは，今回の改定により，サービスラインナップやサービス内容の見直しと利用負担金の値下げを実施した．

3.1 サービス提供内容の変更点

ここでは，2013 年 3 月までのサービスからの変更点をサービス種別ごとに説明する．

3.1.1 VM ホスティングサービス

VM ホスティングサービスは，搭載するディスクや CPU，メモリの計算機資源について，初期スペックならびに増設可能な上限値を増やした．これは，サービス開始当初の 4 年前に比べ，世の中の標準的なサーバのスペックが向上しており，それらと同等の環境を提供するための改定である．また，これまで CPU とメモリの増設はセットで行う必要があったが，今回から個別に増設が可能になった．なお，ディスクの増設は 200GB 単位となった．さらに，事務組織などを中心に，NAS によるファイル共有を業務で活用したいという需要があり，VM サーバを部署内のファイル共有専用サーバとして利用できる NAS パッケージの提供も始めた．この NAS パッケージは FreeNAS というファイル共有専用の OS を搭載することで，複雑な Linux の知識がなくても，Web インタフェースを用いて，簡単にファイル共有サーバを利用できるものである．

3.1.2 ホームページサービス

ホームページサービスは標準の公開スペースを 5GB に変更し，20GB または 50GB への拡張が可能になった．これまでホームページサービスでは，9 割以上の利用者が 5GB 未満のコンテンツしか公開しておらず，提供している容量が持てあまされていた．これをふまえ，標準の公開スペースの容量を小さくする代わりに利用負担金を大幅に値下げした．なお，これまで通りの 20GB やそれ以上の容量へのニーズには，別途追加で利用負担金を支払うことで容量を拡張できるプランを用意し対応する．また，ここ数年のホームページサービスでは，CMS の利用者が増えており，その中でも特に WordPress の利用が多くなっている．そこで，事前にデータベースの設定がされた WordPress をコンテンツ公開スペースに設置し，利用開始後にすぐにインストールを実行可能な「WordPress パッケージ」を用意した．

3.1.3 ストリーミングサービス

ストリーミングサービスは，これまでオプションサービスという扱いであったが、今回の改定でホスティング・ホームページサービスの単体サービスに格上げした．また，ホームページサービスと同様に初期の公開スペース容量を 5GB とし，20GB または 50GB への拡張が可能である．ストリーミングを行うサーバも新しくなり，Helix Server 14.3 の環境を利用できる．

3.1.4 提供サービスの一部終了

これまで提供してきた「個人向けホームページサービス」と「メール転送サービス」ならびに「Oracle(オプションサービス)」については，利用数がわずかであったので，サービス終了となり，2013 年 4 月以降は提供されない．なお，メール転送に関しては，単体で提供されるサービスが終了するだけであり，「ホームページサービス」のメール転送機能については，2013 年 4 月以降もこれまで通り利用できる．

3.2 利用負担金の改定

今回のサービス改定で利用負担金はこれまでに比べ大幅に値下げとなった．改定後の VM ホスティングサービスの利用負担金ならびに CPU やメモリ，ディスクを増量する際に加算される利用負担金は表 1 の通りである．改定後のホームページサービスの利用負担金とコンテンツ公開スペースを 20GB または 50GB に拡張する場合に加算される利用負担金は表 2 の通りである．ストリーミングサービスの利用負担金とコンテンツ公開スペースを 20GB または 50GB に拡張する場合に加算される利用負担金は表 3 の通りである．

表１　VM ホスティングサービス利用負担金

区分	利用負担額	単位
VM ホスティングサービス	72,000 円/年	1 仮想マシンにつき
CPU 増量	18,000 円/年	2 コアにつき(最大 8 コアまで)
メモリ増量	18,000 円/年	8GB につき(最大 64GB まで)
ディスク増量	18,000 円/年	200GB につき(最大 1,000GB まで)

表２　ホームページサービス利用負担金

区分	利用負担金	単位
ホームページサービス	6,000 円/年	1 ドメイン名につき
公開スペース 20GB プラン	3,000 円/年	
公開スペース 50GB プラン	9,000 円/年	

表３　ストリーミングサービス利用負担金

区分	利用負担金	単位
ストリーミングサービス	6,000 円/年	1 申請につき
公開スペース 20GB プラン	3,000 円/年	
公開スペース 50GB プラン	9,000 円/年	

3.3 利用初年度の利用負担金減額

本サービスの利用負担金は年度単位で課金される．新たにサービスを利用する場合，利用初年度の利用負担金は申請された月に応じて，月割りに減額される．なお，年度途中でサービスを終了する場合でも，残りの月数に応じた利用負担金の減額は行われない．

4　各サービスの詳細

次に新しくなったホスティング・ホームページサービスの詳細について説明する．

4.1 VM ホスティングサービス

VM ホスティングサービスは占有バーチャルマシン (VM) による独自ドメインの計算機環境 (サーバ) を提供する. 本サービスを利用することで、独自のサーバの構築・運用が可能である．また，利用者の要望に応じて提供機能の拡張や OS の変更など, サーバ環境のカスタマイズにも対応する．

4.1.1 サービスの特徴とメリット

ハードウェアを購入・管理する必要がなく，サーバ運用に必要な費用と労力を軽減できる．サーバへは SSH を用いることで，ネットワークを通じてログインし操作することが可能であり，root 権限が付与された占有サーバであるため，実機を購入するのと同様に自由にサーバを構築できる．特に，ホームページサービスでは，利用できる PHP・CGI のバージョンや設定のパラメータが決まっているが，VM ホスティングサービスでは利用者自身でサーバを構築するため，バージョンの選択や細かい設定変更が可能となる．さらに，占有サーバ内で仮想ホスト機能を用いて，複数のドメインを活用したサイトの展開も可能である．

4.1.2 VM サーバのスペック

VM ホスティングサービスで提供される VM サーバは，標準で RHEL6 がインストールされている．また，NAS パッケージで申請されたサーバの OS は FreeNAS8 である．その他の OS についても，当サービスで使用している仮想化環境の KVM でサポートされていれば導入が可能である．なお, 有償の OS で RHEL 以外を利用する場合は, ライセンスを申請者自身で用意する必要がある．

VM サーバは標準で CPU:2 コア, メモリ:8GB, ディスク:200GB となっている．これらは追加の利用負担金を支払うことでそれぞれの計算機資源を増量可能である．

4.1.3 利用例

VM ホスティングサービスの計算機環境は次のようなケースで利用されている.

- 部局や学科・専攻などで大規模な Web サイトやスプール付きの電子メールを運用
- ホームページサービスで動作しない CMS や高度な Web アプリを活用したサイトを構築
- 部署内でのファイル共有サーバの運用
- 学会や学術イベントなどの運営サーバ

4.1.4 利用の対象と利用の範囲

VM ホスティングサービスの利用の対象は,「原則として京都大学の教員が一員となっている学術研究・教育等の組織・プロジェクト，及び京都大学の部局，学科・専攻，研究室等（kyoto-u.ac.jp 以下のサブドメインを有する組織）. 当該部局・組織の代表者または広報担当者（京都大学の教職員）が本サービスの申請者となること.」と定められている.

また，利用の範囲は学術研究・教育等に関する情報発信・広報に限られる.

4.1.5 サービスを利用する上での注意点

VM ホスティングサービスは，サーバを利用者自身が運用しなければならないため，利用にはサーバ管理と情報セキュリティについての十分なスキルが必要になる．自身でサーバを確実に管理できない場合は，専門業者へ管理を委託するなど，万全の対策をした上で利用をお願いしたい.

4.2 ホームページサービス

ホームページサービスは，共有 Web サーバの仮想ホスト機能を用いた独自ドメイン名でのホームページ公開とメール転送の環境を提供する．本サービスを利用すれば，サーバを用意することなくホームページや PHP・CGI を利用した Web アプリケーションの公開などができる.

4.2.1 サービスの特徴とメリット

ホームページサービスを利用すれば, Web サイト公開する上で，新たにサーバを購入して構築する必要がなくなる．また．サイト公開後もサーバの維持管理やセキュリティ対策などに労力・費用を必要としない．サイトコンテンツの更新でファイルをアップロードできる権限は，複数の教職員や学生ならびに ID 発行を受けた委託業者に付与できる．なお，KUINS-II 利用負担金は，情報環境機構が利用者に代わって負担する形で提供しているため，別途支払う必要はない.

4.2.2 Web 公開環境のスペック

ホームページサービスの Web サーバは RHEL6 上で動作する Apache を用いて提供される．公開スペースは標準で 5GB となっており，追加の利用負担金を支払うことで 20GB または 50GB に拡張できる．本サービスは SSH などを用いたサーバへのログインを認めておらず，コンテンツは FTPES に対応した FTP クライアントを用いてアップロードを行う.

データベース環境は MySQL と PostgreSQL が用意されている．各データベースのバージョンは表 4 の通りである．データベースも同様にサーバへの直接ログインを認めていないため，これらは公開される PHP や CGI のスクリプトからの操作または，提供される Web インタフェースの phpMyAdmin と phpPgAdmin のみで操作が可能である. ホームページサービスで使用できる PHP 及び各種 CGI の言語のバージョンは表 5 の通りである.

表 4　データベースのバージョン

（2013 年 4 月 1 日現在）

言語	バージョン
MySQL	5.0.77
Postgre SQL	8.1.18

表 5　利用できるスプリクト言語

（2013 年 4 月 1 日現在）

言語	バージョン	実行パス
PHP	5.3.3	/usr/bin/php-cgi
Perl	5.10.1	/usr/bin/perl
Python	2.6.6	/usr/bin/python
Ruby	1.8.7	/usr/bin/ruby

ホームページサービスでは，この他に Web サイトを公開されるドメインでメールを受信して転送するメール転送機能を提供している．利用者は転送アドレスを .forward+ ファイルならびに Mailman で設定が可能である.

4.2.3 利用例

ホームページサービスでの Web サイト公開は次のようなケースでの情報発信に利用されている.

- 部局や学科・専攻・研究室
- コンソーシアムや COE などのプロジェクト
- 国際会議や学会などのイベント

4.2.4 利用の対象と範囲

ホームページサービスの利用の対象は,「原則として京都大学の教員が一員となっている学術研究・教育等の組織・プロジェクト，及び京都大学の部局，学科・専攻，研究室等（kyoto-u.ac.jp 以下のサブドメインを有する組織）を対象とし，京都大学の教職員がその代表者または広報責任者であり，本サービスの申請者となること.」と定められている.

また，利用の範囲は学術研究・教育等に関する情報発信・広報に限定される.

4.3 ストリーミングサービス

ストリーミングサービスは，映像や音声などのメディアコンテンツを学内外にストリーミング配信する環境を提供する.

4.3.1 サービスの特徴とメリット

ストリーミングサービスを利用すれば，インターネット上にコンテンツをストリーミング配信する際に専用の配信サーバを用意する必要がなくなる．なお，ストリーミングサービスは，配信するコンテンツごとにグローバル配信と学内限定配信を分けることが可能である.

4.3.2 ストリーミング環境のスペック

ストリーミングサービスの配信サーバは Helix Server14 を用いて提供される. 公開スペースは標準で 5GB となっており，追加の利用負担金を支払うことで 20GB または 50GB に拡張できる. 配信するコンテンツは FTPES に対応した FTP クライアントを用いてアップロードできる.

Helix Server は RealAudio，RealVideo，QuickTime，MPEG4 など様々なファイル形式で記録されたマルチメディアコンテンツの配信に対応している.

4.3.3 利用例

ストリーミングサービスは，ホームページサービスなどで公開している Web サイトのアクセス者向けに，映像や音声などのコンテンツを配信するケースで利用されている.

5 サービスの申請について

5.1 申請方法

本サービスの利用申請は下記のアドレスの申請負担金システムによる Web オンライン申請で行う．なお，申請負担金システムにログインするには京都大学の SPS-ID が必要となる.

- https://sabs.iimc.kyoto-u.ac.jp/

申請負担金システムの利用方法は下記に PDF ファイルで公開されている.

- http://sabs.iimc.kyoto-u.ac.jp/manual/top

サービス申請から利用開始までの具体的な手順は図２の通りである.

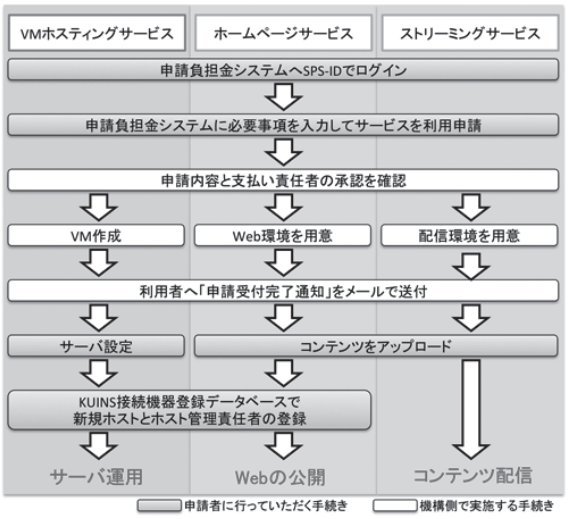


図２　サービス申請から利用開始までの流れ

5.2 仮想ホスト名の指定

ストリーミングサービス以外のサービスでは，アカウントごとに独自の仮想ホスト名での利用を基本としており，申請時に指定する．本サービスで利用できる仮想ホスト名は，原則的に

- 仮想ホスト名.所属の部局のサブドメイン.kyoto-u.ac.jp

となる．仮想ホスト名は任意の文字列を使用できるが，部局内で了承を得る必要がある．(利用される仮想ホスト名は kyoto-u.ac.jp に続く部局のサ

ブドメイン名の配下に置くことを原則としているが，それ以外を希望する場合は，汎用コンピュータシステム運用委員会で承認されれば使用できる）．

5.3 大型計算機システム利用者番号

ホスティング・ホームページサービスの各サービスを申請すれば，z59 で始まる本サービス専用の大型計算機システム利用者番号が発行される．この利用者番号は本サービス利用者のアカウント管理のためのものであり，これを用いてスーパーコンピュータなどのサービスは利用できない．スーパーコンピュータを利用する場合は，スーパーコンピュータ用に利用者番号（利用負担金が必要）を別途取得する必要がある．

6　問い合わせ

サービスに関する問い合わせは，情報環境機構サイトの問い合わせフォームまたは電子メールにて受け付けている．利用中のサービスに関する質問や設定変更は，z59 で始まる大型計算機システム利用者番号を必ず記載した上で．申請者より行っていただきたい．

また，本サービスの担当者から利用者へ連絡する際は，基本的に電子メールを使用する（緊急の場合は電話の可能性もあり）．メールを送付する宛先は，サービスの連絡担当者用メールアドレスとしてファイルサーバ上にアップロードいただいている転送設定の「.forward+」ファイルに記載のアドレスである．したがって，利用者や利用者のアドレスに変更があった際は，必ずファイルの修正をお願いしたい．なお．サービスの利用に関する重要な連絡はこれに加えて申請負担金システムに登録されているメールアドレスにも送られる．

7　おわりに

本稿では，汎用コンピュータシステムのリプレースにあわせて改訂されたホスティング・ホームページサービスに関して，新しくなったサービスの内容改定のポイントと改定後のサービス利用方法について説明した．

なお，本サービスの詳細・利用サポート情報・最新情報については，学術情報基盤サービスの Web ページにて案内している．

- http://www.iimc.kyoto-u.ac.jp/services/whs/

また，本サービスの利用申請受付完了時に，本サービスを利用する上で必要な事項が記載された利用マニュアルが添付されるので，これらの情報も併せて確認いただきたい．（サイトやマニュアルは，サービスの仕様の変更やよくある問い合わせ内容を踏まえて適宜改訂し，マニュアルに関しては改定の際に利用者のメーリングリストで最新版が送付される）．

CrayPat による性能解析

武田 大輔

クレイ・ジャパン・インク

1 はじめに

本稿では、スーパーコンピュータシステム A として運用されている Cray XE6 上で利用可能な性能解析ツール CrayPat の使い方をご紹介します。性能解析ツールとは、ユーザプログラムの実行時間がプログラム中のどこでたくさん消費されているかを調べたり、またキャッシュミス等の実行性能に影響を及ぼすイベントがどれほど発生するかを調べたりするソフトウェアで、プログラムの高速化を図る上で欠かせないツールです。

2 サンプリングとトレーシング

CrayPat の主要機能は、ユーザプログラム中の関数・サブルーチンに対して性能解析用のコードを埋め込み、プログラム実行中に関数・サブルーチン単位の実行時間やイベント発生回数を計測することです。これを「トレーシング (tracing)」と呼んでいます。これは非常に強力な解析方法ですが、実行時間が短く多数回呼び出されている関数・サブルーチンまで対象にしてこれを行ってしまうと、オーバーヘッドの影響が大きくなり、プログラムの実行時間が極端に長くなってしまうことがあります。そこで CrayPat では、「サンプリング (sampling)」と呼ばれる軽量な方法によって簡易なプロファイルを採り、その結果を基にトレーシングの対象とする関数・サブルーチンを決定するという二段階の手順を踏むのが一般的になっています。トレーシング対象の選択は自動的に行われるため、この方法は「Automatic Profile Analysis (APA)」と呼ばれます。以下では、APA に基づいてサンプリング及びトレーシングを行う基本的な手順についてご説明します。この手順の概要は以下の通りです。

1. モジュールのロード
2. プログラムのビルド
3. pat_build (サンプリング)
4. プログラム実行 (サンプリング)
5. pat_report による結果生成
6. pat_build (トレーシング)
7. プログラム実行 (トレーシング)
8. pat_report による結果生成

3 解析手順ひとめぐり

3.1 モジュールのロード

最初に、使用するコンパイラに対応した PrgEnv と perftools のモジュールをロードしておきます。本稿ではコンパイラとしてデフォルトでロードされている Cray Compiling Environment (CCE) を使用しますので、PrgEnv はそのままとし、perftools のみをロードします。

```
$ module load perftools
```

3.2 プログラムのビルド

この状態で解析対象となるプログラムのビルドを行います。性能解析のために必要なコンパイラオプションやライブラリ等は perftools のモジュールによって自動的に設定されますので、他は特に設定を変えることなく、通常通りの手順 (make 等) でビルドを行います。注意点として、後にオブジェクトファイル (*.o ファイル) が必要とな

りますので、これらを削除せず残しておきます。なお、-c オプションを使用しないでコンパイルをしていて、オブジェクトファイルが生成されない場合には、CrayPat が自動的にオブジェクトファイルを保存します。

以下、プログラム（実行形式ファイル）の名前は myprogram とし、このファイルはカレントディレクトリにあるものとします。

3.3 pat_build とサンプリング

プログラムのビルドが完了したら、次にサンプリングのための準備を行います。pat_build コマンドに –O apa オプションを指定することで、APA のためのサンプリング用に準備された (instrumented) 実行形式ファイルを生成します。

```
$ pat_build -O apa ./myprogram
```

この操作によって、myprogram+pat のように元のプログラム名に +pat が追加された実行形式ファイルが生成され、これを実行することによってサンプリングが行われます。実行は、通常通りの手順でサブシステム A の計算ノードで実行します。計算ノードでプログラムを実行するためには aprun コマンドを使用する必要があることを忘れないようご注意ください。以下にジョブ実行の簡単な例を示します。サブシステム A におけるジョブ実行手順の詳細は 参考文献 [2] をご参照ください。

```
$ cat go.csh
#!/bin/tcsh
#QSUB -A p=64:t=1:c=1:m=1G
#QSUB -q キュー名
aprun -n $LSB_PROCS ./myprogram
$ qsub go.csh
```

pat_build を行う際の注意点をいくつか挙げておきます。まず、既に注意したように、pat_build を実行するにはオブジェクトファイルが必要です。これがないと、リスト 1 のようなエラーが発生してしまいます。また、元のプログラムのビルド時に perftools のモジュールをロードしていないと、やはり pat_build コマンドが リスト 2 のようなエラーで失敗してしまいます。これらのエラーが発生する場合には、元のプログラムのビルドが適切でなかった可能性がありますのでご確認ください。

リスト 1　pat_build のエラー (1) オブジェクトファイルが見つからない場合

```
$ pat_build -O apa ./myprogram

ERROR: Can not access the file 'bt.o' [No such file or directory].
ERROR: Can not access the file 'initialize.o' [No such file or directory].
(以下略)
```

リスト 2　pat_build のエラー (2) プログラムのビルド時に perftools がロードされていなかった場合

```
$ make
$ module load perftools
$ pat_build -O apa myprogram
ERROR: Missing required ELF section '.note.link' from the program 'myprogram'. Load the correct
'perftools' module and rebuild the program.
```

3.4 サンプリング結果

さて、サンプリングの実行が完了すると、拡張子 xf のファイルが出力されているはずです。これがサンプリング結果を保持するファイルとなっていますが、さらにこれを pat_report コマンドで処理することによって、人間可読なテキスト出力を含むいくつかの結果ファイルに整理できます。

```
$ pat_report myprogram+pat+21581-115s.xf > sampling.txt
```
(実際には 1 行)

xf ファイルの名前は実行ごとに変化しますので、適宜変更してください。この操作によって、以下の 3 つの出力を得ることができます。

A) サンプリング結果のテキスト出力
B) ap2 ファイル
C) apa ファイル

上記 A) のテキスト出力には、一般に「プロファイル」と呼ばれる実行時間分布や、メモリ使用量等の情報が出力され、これだけでも有用な情報を得ることができます。リスト 3 にプロファイルの例を示します。ここで “Samp” はサンプル数で、デフォルトでは 10 ミリ秒間隔で IP サンプリングを行っています。サンプル数は必ずしも正確な実行時間を表すわけではありませんが、プログラムの各部分の実行時間の目安として使用することができます。”Imb. Samp” は、該当する処理に対するサンプル数の全プロセス・スレッドにわたった平均値と最大値との差を表しており、ロードバランスの良し悪しの指標として使用できます。ヘッダ部分の右端に “Group Function PE=HIDE” とあるのは、この表がまず USER や MPI 等のグループでまとめられており、さらに各グループについてサンプル数が多い関数・サブルーチンから順に表示されていること、また各 PE (MPI プロセスのこと) のサンプル数は表示せずに平均値だけを出力していることを、それぞれ表しています。後述するように、これらの整理の方法は容易にカスタマイズすることができます。

リスト 3　サンプリングによるプロファイル例

```
Table 1:  Profile by Function

 Samp%  |  Samp  | Imb.  |  Imb.  |Group
        |        | Samp  | Samp%  | Function
        |        |       |        |  PE=HIDE

 100.0% | 2988.3 |    — |     — |Total
|-------------------------------------------------------
|  88.0% | 2629.3 |    — |     — |USER
||------------------------------------------------------
||  29.3% |  875.3 |  56.7 |   6.2% |binvcrhs_
||   8.5% |  254.5 |  35.5 |  12.4% |z_solve_cell_
||   8.2% |  245.8 |  52.2 |  17.8% |y_solve_cell_
||   8.1% |  241.0 |  35.0 |  12.9% |matmul_sub_
||   7.2% |  214.7 |  30.3 |  12.6% |x_solve_cell_
||   7.1% |  210.9 |  21.1 |   9.3% |compute_rhs_
||   5.0% |  149.6 |  24.4 |  14.2% |matvec_sub_
||   3.8% |  112.1 |  13.9 |  11.2% |x_backsubstitute_
||   3.4% |  101.9 |  35.1 |  26.0% |z_backsubstitute_
||   3.0% |   89.8 |  32.2 |  26.8% |y_backsubstitute_
||   1.3% |   40.2 |  20.8 |  34.6% |add_
||=======================================================
|  11.9% |  356.4 |    — |     — |ETC
||------------------------------------------------------
||  10.1% |  302.1 |  75.9 |  20.4% |
||   1.7% |   50.5 |  30.5 |  38.2% |__cray_dcopy_AVX
||=======================================================
|   0.1% |    2.6 |    — |     — |MPI
|=======================================================
```

B) の ap2 ファイルは、主に CrayPat の GUI である Apprentice2 によってプロファイル結果を表示するために使用されるファイルですが、これを改めて pat_report で処理してテキストレポートを生成することも可能です。pat_report コマンドの注意点として、xf ファイルを処理する際には、サンプリングで得られたアドレスと関数・サブルーチンやソース行を対応付けるために実行形式ファイルが必要となる点が挙げられます。通常は xf ファイル中に実行形式ファイルの場所が保存されていて自動探索されますが、ファイル名やパスが変更されていて自動で見つけられない場合は、-i オプションを指定して明示的に実行形式ファイル (myprogram+pat に相当するもの) の場所を指示する必要があります。一方、ap2 ファイルにはアドレスとソースプログラムの対応づけ情報が保存されているため、ap2 ファイルを解析する際には実行形式ファイルは不要です。この理由から、CrayPat によるサンプリング・トレーシングが終了したら、すぐに

pat_report を実行して xf ファイルを ap2 ファイルに変換しておくと便利です。

C) の apa ファイルは、次のトレーシングの際に使用するものです。

3.5 トレーシング

トレーシングを行うには、再度 pat_build を用いて、トレーシング用の実行形式ファイルを準備します。この際に、先ほどのサンプリングの際は –O apa というオプションを使用しましたが、今回は –O オプションで前節 C) の apa ファイルを指定します。

```
$ pat_build -O myprogram+pat+21581-115s.apa
INFO: Trace intercept routine created for the 5972 byte function 'binvcrhs_'.
INFO: Trace intercept routine created for the 22431 byte function 'z_solve_cell_'.
INFO: Trace intercept routine created for the 1693 byte function 'matmul_sub_'.
INFO: Trace intercept routine created for the 22315 byte function 'y_solve_cell_'.
INFO: Trace intercept routine created for the 19962 byte function 'x_solve_cell_'.
```

xf ファイルと同様、apa ファイルの名称も実行ごとに変化しますので適宜変更してください。上記のメッセージから、いくつかの関数・サブルーチンに対してトレーシング用のコードが生成されたことがわかります。この時に生成されるトレーシング用実行形式ファイルは myprgram+apa のように +apa が追加されたものになっています。これを実行するとトレーシングが行われます。トレーシングが完了すると、再び xf ファイルが生成されますので、これを pat_report で処理してレポートを作成します。

```
$ pat_report myprogram+apa+21328-7t.xf > tracing.txt
```
(実際には 1 行)

今回は、テキストレポートと ap2 ファイルが生成されますが、apa ファイルは生成されません。テキストレポートには、サンプリング時と同様の形式のプロファイルの他、関数・サブルーチン毎の詳細な解析結果が出力されます。例えば、リスト 4 にはパフォーマンスカウンタによって計測した演算回数等が報告されています。

リスト 4　トレーシングの出力例（一部抜粋）

```
USER / z_solve_cell_
------------------------------------------------------------------------
Time%                                                  25.9%
Time                                                72.179138 secs
Imb. Time                                            2.715097 secs
Imb. Time%                                              3.7%
Calls                                  22.362 /sec      1608.0 calls
PAPI_L1_DCM                            19.238M/sec   1383357458 misses
PAPI_TLB_DM                             0.015M/sec      1113752 misses
PAPI_L1_DCA                           240.618M/sec  17302452752 refs
PAPI_FP_OPS                            56.063M/sec   4031416800 ops
DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE:
  ALL                                  20.186M/sec   1451547830 fills
DATA_CACHE_REFILLS_FROM_NORTHBRIDGE     0.475M/sec     34140187 fills
Average Time per Call                                0.044888 secs
```

4　応用的な使い方

前節までで CrayPat の基本的な使用方法はご理解いただけたかと思います。これだけでもたくさんの情報を得ることができますが、本節ではもう少し CrayPat の動作をカスタマイズし、必要な情報を的確に得る方法についてご紹介します。

4.1 いろいろなレポート生成

前節では pat_report がデフォルトで生成するレポートについて見ましたが、実際には pat_report はたくさんの種類のレポートを生成することができます。非常に多数のオプションがあるため詳細は man pat_report をご参照いただくとして、ここではいくつかよく使われる例を挙げたいと思います。

まず、前節で見たプロファイルでは PE=HIDE となっていて、関数・サブルーチン毎のサンプル数や実行時間はすべての PE（MPI プロセス）にわたる平均値となっていました。これを、スレッド毎や PE 毎の値を出力させるためには、-b オプションに thread（th と省略可）や pe を指定します。

リスト 5　プロファイルに PE 毎のサンプル数を表示させた例（一部抜粋）

```
Table 1:

 Samp%  |  Samp   | Imb.   |  Imb.   |Group
        |         | Samp   | Samp%   | Function
        |         |        |         |  PE

 100.0% | 2988.3 |     — |      — |Total
|-------------------------------------------------
|  88.0% | 2629.3 |     — |      — |USER
||------------------------------------------------
||  29.3% |  875.3 |  56.7 |    6.2% |binvcrhs_
|||-----------------------------------------------
3||  31.2% |  932.0 |     — |      — |pe.42
3||  31.1% |  929.0 |     — |      — |pe.40
3||  31.0% |  927.0 |     — |      — |pe.14
3||  30.8% |  919.0 |     — |      — |pe.5
3||  30.6% |  914.0 |     — |      — |pe.26
3||  30.5% |  910.0 |     — |      — |pe.60
3||  30.4% |  908.0 |     — |      — |pe.36
3||  30.4% |  907.0 |     — |      — |pe.19
（以下略）
```

```
$ pat_report -b gr,fu,pe
  myprogram+ pat+21581-115s.ap2 >
  sampling.txt
```

(実際には 1 行)

ここで、pe と一緒に指定している gr や fu は Group や Function を出力させるためのものです。この出力例を リスト 5 に示します。

もう一つの例として、コールツリーの生成をご紹介します。コールツリーとは、関数・サブルーチンの呼び出し系列を表すもので、どの関数・サブルーチンがどこから呼び出されているのかを調べるために使用します。pat_report でコールツリーを生成するためには、-O calltree (ct と省略可) または –O caller (ca と省略可) を指定します。両者の違いは、calltree が呼び出しを呼び出す側から順に「上から下へ」辿るのに対し、caller では呼び出される側から順に「下から上へ」辿ります。また、-O ct+src や –O ca+src を指定すると、行番号の情報も追加されます。

リスト 6　コールグラフの例（一部抜粋）

```
Table 1:  Function Calltree View

 Samp%  |  Samp   |Calltree
        |         | PE=HIDE

 100.0% | 2988.3 |Total
|-------------------------------------------
| 100.0% | 2988.2 |mpbt_
|  99.4% | 2970.7 | adi_
|||-----------------------------------------
3||  29.8% |  889.0 |z_solve_
||||---------------------------------------
4|||  22.4% |  669.5 |z_solve_cell_
|||||-------------------------------------
5||||   9.6% |  287.2 |binvcrhs_
5||||   8.5% |  254.5 |z_solve_cell_(exclusive)
5||||   2.6% |   78.5 |matmul_sub_
5||||   1.6% |   48.8 |matvec_sub_
|||||=====================================
4|||   3.4% |  101.9 |z_backsubstitute_
4|||   2.9% |   86.9 |
||||=======================================
```

例として、リスト 3 のサンプリング結果について次のコマンドでコールツリーを生成した結果をリスト 6 に示します。

```
$ pat_report -O calltree
  myprogram +pat+21581-115s.ap2 >
  calltree.txt
(実際には 1 行)
```

既にお気づきの方もいらっしゃるかと思いますが、前節のサンプリングとトレーシングの結果を比較してみると、リスト 3 ではサブルーチン z_solve_cell_ のサンプル数は全体の 8.5% しかなかったのに、リスト 4 の実行時間の割合では 25.9% も占めていることになっています。これは、サンプリングのプロファイルでは各サブルーチン自身の中で発生したサンプル数だけを表示しているのに対して、トレーシングのプロファイルではそれらのサブルーチンの中から (直接または間接に) 呼び出されるサブルーチンの実行時間もすべて含めた値を表示しているため、このような差が生じています。この状況は、リスト 6 のコールツリーを見るとはっきりと理解できます。つまり、z_solve_cell_ の中からは binvcrhs_ や matmul_sub_ 等が呼び出されているので、トレーシングで計測された実行時間にはこれらの実行時間も含まれています。一方、サンプリングで z_solve_cell_ のサンプル数として計上されているのは、z_solve_cell_(exclusive) の部分だけになります。

4.2 apa ファイルのカスタマイズ

前節では APA 任せでトレーシングを実行しましたが、これは簡単にカスタマイズすることができます。apa ファイルは単なるテキストファイルとなっており、テキストファイルで開いてみると、リスト 7 のような pat_build コマンドに対するオプションが列記されています。これを編集することでトレーシングをカスタマイズできます。例えば、-T オプションはトレーシングする関数・サブルーチンを指定するものですので、APA がトレーシング対象として取り上げなかったものを追加したい場合、あるいは APA はトレーシング対象とした関数・サブルーチンをトレーシングしたくない場合等は、この部分を編集するだけで容易に調整が可能です。その他の pat_build オプションについては man pat_build をご参照ください。

リスト 7　apa ファイルの内容 (一部抜粋)

```
  -w  # Enable tracing of user-defined
functions.
      # Note: -u should NOT be specified as an
additional option.

# 29.29% 5972  bytes
         -T binvcrhs_

# 8.52% 22431  bytes
         -T z_solve_cell_

# 8.23% 22315  bytes
         -T y_solve_cell_
```

4.3 パフォーマンスカウンタ

既に見たように、トレーシングの際にはパフォーマンスカウンタを使用したイベントサンプリングが行われます。実際にどのイベントを計測するかということは、トレーシング実行時に環境変数 PAT_RT_HWPC を設定することで指定できます。よく使われるイベントの組み合わせは予めグループ化されて定義されており、各グループに番号が付けられていますので、その番号を指定するだけでイベントグループを指定することができます。イベントグループについては man hwpc を参照してください。サブシステム A (CPU は AMD Interlagos) では、グループ 23 がデフォルトとなっています。PAPI になじみのある方は、以下の例のようにして PAPI イベント名や PAPI が認識可能なネイティブイベント名を直接 PAT_RT_HWPC に指定することもできます。

```
$ setenv PAT_RT_HWPC
PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_FP_OPS
(実際は 1 行)
$ aprun -n $LSB_PROCS ./myprogram+apa
```

4.4 CrayPat API

前節のトレーシングでは、関数・サブルーチン単位で計測コードを埋め込んでいましたが、プログラムによっては計測対象としたい部分が必ずしも関数・サブルーチン単位になっているとは限らないこともあるかと思います。このような場合は、ソースプログラムに若干の修正を加えることによって、計測対象としたい部分を詳細に指定することも可能です。詳細は man pat_build の“APPLICATION PROGRAM INTERFACE (API)” の節をご参照ください。

4.5 Apprentice2

Apprentice2 は CrayPat の GUI であり、X Window System が表示できる環境であれば容易に使用することができます。サブシステム A で X Window System を利用する方法は参考文献 [3] をご参照ください。この設定を行った後に、perftools モジュールを読み込み app2 コマンドを実行すると、お使いの端末上で Apprentice2 のウィンドウが開きます。

```
$ module load perftools
$ app2
```

紙数の都合から GUI の使用方法を詳細に解説することはできませんが、典型的なユーザインターフェイスを備えていますので、あまり戸惑うことなく直観的に使用できると思います。基本的な使用方法は、File メニューから ap2 ファイルを指定してオープンし、ツールバーから表示したいレポート形式を選択します。一例として、リスト 6 でテキストとして生成したコールグラフを GUI で表示した例を 図 1 に示します。

なお、man app2 にも記載されています通り、表示するレポート形式によってはデフォルトよりも詳細な情報を保存する必要があるため、環境変数 PAT_RT_SUMMARY を 0 に設定しなければなりませんが、これによって xf ファイルや ap2 ファイルが巨大になる可能性があります。これらの出力ファイルが大きくなりすぎないようにするため、繰り返し回数や時間ステップ数を小さくして実行時間を短くしたり、または CrayPat API を使用して解析対象となる部分を制限する等の工夫をされることをお勧めいたします。

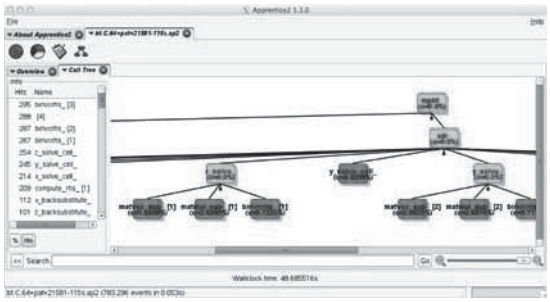


図 1　Apprentice2 によるコールグラフ

5　まとめ

本稿では、CrayPat の使用方法を駆け足でご説明しました。特に第 4 節は限られた内容になってしまいましたが、ここでご紹介した以外の機能もぜひお試しいただければと思います。プロファイル解析は性能最適化の第一歩であり欠かせないものですので、本稿を足がかりに CrayPat とサブシステム A をご活用いただけますと幸いです。

6　参考文献・リンク

[1] CrayPat 使用方法
man intro_craypat, pat_build, pat_report, pat_help をご参照ください。(サブシステム A にログインし module load perftools の後にご利用ください。)

[2] サブシステム A でのジョブ実行方法
- http://web.kudpc.kyoto-u.ac.jp/manual/ja/run
- http://web.kudpc.kyoto-u.ac.jp/manual/ja/run/batchjob/systema

[3] X Window System (Exceed onDemand) の使用方法
- http://web.kudpc.kyoto-u.ac.jp/manual/ja/install/eod
- http://web.kudpc.kyoto-u.ac.jp/manual/ja/login/eod

GPUを用いたFEMアプリケーションの高速化

大島 聡史 *

*東京大学 情報基盤センター

1　はじめに

本稿では筆者らがGPU(Graphics Processing Unit)を用いて行っている研究の一部を紹介させていただく．

GPU はその名の通り画像処理向けに発展してきたハードウェアである．高度な画像処理を高速に行うべく発展してきた GPU は，近年では 2 章で紹介するように既存の CPU とは特性の異なる並列計算ハードウェアとなっている．GPU は CPU と比べて高い理論演算性能や理論メモリ性能を備えているため，HPC 分野をはじめとして様々な領域で活用され，注目を集めるようになってきた．現在では GPU に対応した高性能な科学技術計算アプリケーションが多く開発され利用されている．もちろん，GPU 本来の仕事である画像処理能力を活かした高度なインタラクティブアプリケーションなどへの活用も行われている．

大量の GPU を搭載したスーパーコンピュータシステム，いわゆる「GPU スパコン」における超大規模な GPU の活用も進められている．例えばスパコンの性能を順位付けしている Top500 の最新ランキング (2013 年 6 月版) の No.2 には米国 DOE/Oak Ridge National Laboratory の Titan という GPU スパコンがランクインしている．Titan は Cray 社製の GPU スパコンシステム XK-7 を計算ノードとして用いているが，これは京都大学にて 2012 年 5 月に稼働を開始したスパコン「Camphor」を構成する XE-6 の GPU 搭載モデルである XK-6 の後継機である．GPU のように既存の CPU とは異なる高速演算ユニット (アクセラレータ) を搭載した不均質構成の (ヘテロジニアスな) スパコンは近年のスパコンにおける最大のトレンドの 1 つと言える．

筆者は 2004 年から GPU を用いた研究を行っている．現在は主に疎行列ベクトル積 (sparse matrix vector multiplication, SpMV) の高速化や，有限要素法 (Finite Element Method, FEM) アプリケーションの高速化 (主に CG 法の高速化を行っており，SpMV の高速化が重要な意味を持つ) などに取り組んでいる．そこで本稿では GPU を用いた FEM アプリケーションの高速化について，考え方と実装内容，そして実装の結果として得られた性能について紹介する．

2　GPUコンピューティング

2.1　GPUとGPUコンピューティング

GPU はいわゆるビデオチップやビデオカードとして提供されてきたハードウェアである．現在では薄型ノート PC やスリム形状の PC の普及とあわせて CPU と GPU が 1 つのパッケージに封入された統合型の GPU も多く用いられているが，HPC 向けの GPU など高性能な GPU は PCI-Express 接続の拡張カードとして提供されている．2000 年頃までの GPU は主に高解像度で高品質なカラー画像をディスプレイに出力するために発展してきた．一方 2000 年頃以降の GPU は高度な陰影処理や照明処理を伴う 3D 描画処理への要求に応えるために性能や機能を向上させてきた．現在の GPU は高い浮動小数点理論演算性能とメモリバンド幅を持つ並列計算ハードウェアへと発展しており，その高い性能は画像処理のみならず様々な用途に活用されるようになってきた．このような活動は GPU を用いた汎用演算 (General-purpose computation) という意味から GPGPU(General-purpose computation using GPUs) もしくは GPU を描画処理=レンダリングではなく演算処理=コンピューティングに使うため GPU コンピューティングと呼ばれている．著者が取り組んでいるような数値計算問題

への活用も GPU コンピューティングの代表的な例の 1 つである．

GPU をアプリケーション高速化に活用するためには，対応するプログラミング言語を用いて自らプログラムを記述する，もしくは GPU を活用することができるライブラリなどを利用する必要がある．本稿で述べる FEM アプリケーションの高速化においては GPU コンピューティングのための GPU プログラミング環境 CUDA(CUDA C) を用いているため，本章では CUDA の概要と最適化手法について簡潔に述べる．なお GPU の発展の経緯や CUDA 以前の GPU 活用等については拙著の記事 [1] 等も参考にしていただきたい．

2.2 CUDA 最適化プログラミング

CUDA は GPU ベンダーである NVIDIA 社の提唱する並列計算プラットフォームとプログラミングモデルの総称である．また C/C++言語を一部拡張した言語仕様とプログラミング言語処理系 (コンパイラ・ライブラリ) が提供されており，CUDA C と呼ばれている．ただし実際には CUDA 対応 GPU のアーキテクチャや CUDA C のことをまとめて CUDA と呼ぶことが多いため，本稿でも特に必要のない限りこれらをまとめて区別せずに CUDA と呼ぶことにする．また今後特に断りのない限り，GPU と表記した場合には CUDA に基づく GPU のことを意味する．

CUDA プログラム (CUDA C プログラム) の全体的な構成 (ソースコード) を図 1 に示す．このプログラムでは

1. GPU 上のメモリを確保
2. CPU から GPU へ計算元データを転送
3. GPU 上で演算 (配列計算) を実行
4. GPU から CPU へ演算結果データを転送

という基本的な一連の動作を行っている．CUDA プログラムは一見すると cuda で始まる API 関数を多用した通常の C/C++に見える．しかし GPU 上で実行される内容が__global__という指示子の付加された関数 (以下 GPU カーネルと呼ぶ) として分離されていることや，GPU カーネル呼び出し時に幾つかのパラメラを与えている (並列度などを与えている) こと，GPU カーネル内では ID を生成して計算対象のデータを指定していることといった CUDA プログラムの特徴的な記述が含まれている．またこの例では使用していないが変数 (メモリ) についても独自の指示子を用いて指定できる情報が存在する．CUDA プログラムの最適化においては，必要に応じて CPU-GPU 間のデータ転送に関する最適化と GPU 上で行われる演算 (GPU カーネル) の最適化を行う必要がある．本稿では特に GPU カーネルの最適化を中心に扱う．

```
__global__ void arrayadd(float *fOut, float *fInA, float *fInB){
  int id = threadIdx.x + blockIdx.x*blockDim.x;
  fOut[id] = fInA[id] + fInB[id];
}
int main(int argc, char** argv){
  float h_InA[N], *h_InB[N], *h_Out[N];
  for(i=0; i<N; i++){ h_InA[i] = ...; h_InB[i] = ...; }
  float *d_InA, *d_InB, *d_Out;
  cudaMalloc((void**)&d_InA, Nof(float)*N);
  cudaMalloc((void**)&d_InB, Nof(float)*N);
  cudaMalloc((void**)&d_Out, Nof(float)*N);
  cudaMemcpy(d_InA, h_InA, Nof(float)*N, cudaMemcpyHostToDevice);
  cudaMemcpy(d_InB, h_InB, Nof(float)*N, cudaMemcpyHostToDevice);
  arrayadd<<< 128, 128 >>>(d_Out, d_InA, d_InB);
  cudaMemcpy(h_Out, d_Out, Nof(float)*N, cudaMemcpyDeviceToHost);
  for(i=0; i<N; i++)printf(" %.2f", h_Out[i]); printf("¥n");
  cudaFree(d_InA); cudaFree(d_InB); cudaFree(d_Out);
  return 0;
}
```

GPU カーネル

1

2

3

4

図 1: CUDA プログラムの例 (ローカル変数の宣言など一部の記述は省略)

表 1 に幾つかの CPU と GPU の性能諸元を示す．CPU と GPU の特に目立った違いとしては計算コアの数があげられる．一般的な CPU に搭載されている計算コアの数はたかだか 10 程度であるのに対して GPU に搭載されている計算コアの数は 100 以上，特に最新世代の GPU では 1000 を越えている．一方で計算コアのクロック周波数は CPU の方が数倍高速である．また計算コアのそなえるキャッシュの階層数や容量にも違いがあり，現在の CPU は高い性能を得るために階層の深い大容量のキャッシュを搭載している一方，GPU に備えられたキャッシュは階層が浅く，容量も多くない．(ちなみに旧来の GPU にはキャッシュが搭載されていなかった．) このように，CPU が高い動作周波数，大量のキャッシュ，高度な分岐予測器などによって高い性能を得るアーキテクチャである一方で，GPU は低速な計算コアを大量に利用して合計の演算性能を稼ぐアーキテクチャであるという違いがある．

ところで，CPU の計算コアと GPU の計算コアには性能諸元の数値だけではわからない大きな違いがあり，最適化戦略に大きな影響を与えている．CPU に搭載された計算コアはそれぞれが個別に演算ユニットと命令発行ユニットを持っており，同時に別々の

表 1: CPU と GPU の性能比較

	Xeon W3520 (Nehalem)	Xeon E5-2687W (SandyBridge)	Tesla C2050 (Fermi)	Tesla K20c (Kepler)
コア数	4 (8 スレッド)	8 (16 スレッド)	448 (32 コア ×14 ユニット)	2496 (192 コア ×13 ユニット)
動作周波数	2.67 GHz	3.1 GHz	1.15 GHz	706 MHz
メモリ種別	DDR3	DDR3	GDDR5	GDDR5
倍精度浮動小数点理論演算性能	42.72 GFlops	198.4 GFlops	515.2 GFlops	1.17 TFlops
メモリバンド幅	25.6 GB/s	51.2 GB/s	144 GB/s	208 GB/s
キャッシュ構成	L1 64KB/core L2 256KB/core L3 8MB/shared	L1 64KB/core L2 256KB/core L3 20MB/shared	L1 64KB/unit L2 768KB/unit	L1 64KB/unit L2 1536KB/unit

※ GPU の L1,L2 キャッシュはユニット単位の共有メモリ．L1 キャッシュは共有メモリ (SharedMemory) と共用であり分割割合を数パターンから指定可能．

処理を行うことができる．一方 GPU については計算コアごとに命令発行ユニットを持っているわけではなく，常に一定数の計算コアが同じ演算を行っている．例えば TeslaC2050 は合計 448 の計算コアを搭載しているが，物理的には 32 の計算コアごとにグループが作られており，この 32 計算コアが 1 つの命令発行ユニットによって制御されている (これを 1WARP と呼ぶ)．イメージとしては，SIMD 計算を行う演算器を計算コアとしてカウントしていると考えると良い．WARP 内の計算コアが別の命令を実行しなくてはならない場合にはマスク処理を用いて処理が分岐させられるため，命令実行のフローとしては全てのフローを経由することになり性能低下要因となる．そのため CUDA プログラムの最適化においては WARP 内で分岐が行われないようにすることが大きな意味を持つ．

CUDA プログラムにおける GPU カーネルの実行の際には 2 段階の並列度を指定する．1 つはグリッドと呼ばれる単位であり，GPU 上の 32 コアや 192 コアのグループと対応づけて考える値である．もう 1 つはスレッドブロックと呼ばれる単位であり，こちらはグループ内の各コアと対応づけて考える必要がある．既存の CPU において OpenMP や MPI を使用する際にスレッド数やプロセス数を指定するのに似ているものの，最適な値の考え方には違いがある．CPU における OpenMP や MPI の場合はスレッド数やプロセス数を搭載されているコア数以下におさえるのが一般的であり，より多くの値を指定すると複数のスレッドやプロセスがコアを奪い合って性能低下を招いてしまう．一方 CUDA の場合には物理的なコア数よりもずっと大きな並列度を用いることが推奨されている．これは GPU のコンテキストスイッチ切り替えコストが CPU と比べて非常に小さいことに関連している．GPU においては計算コアがメモリアクセスによりストールした場合に別のコンテキストに切り替えて計算コアを動かし続けることで全体の演算性能を向上させる．そのため物理コアよりも多くのインスタンスを生成することが推奨されているのである．

スレッドブロックの数 (並列度) は性能に大きな影響を与えることが多い．最適な値の指標は GPU の世代等によっても異なるが，一般的には WARP の倍数である値，例えば 128 などが適している．ただし GPU カーネル内で必要となる資源 (レジスタや共有メモリ) の量とも関連するため，プログラム内容によって最適な値は異なる．また後述するように同一スレッドブロック内の計算コア同士でのみ共有されるメモリが存在するため，共有メモリの使い方とあわせて並列度を設定する必要がある．一方のグリッドについてはあまり明確な指標は存在しておらず，またスレッドブロックと比べると性能に与える影響は一般的に小さめである．そのため，計算対象の持つデータ並列度やアルゴリズム上の並列度，スレッドブロックの並列度，そして GPU に搭載されているマルチプロセッサの数などを勘案し，全計算ユニットが稼働するのに十分な値を指定すれば良い．

並列度と同等以上に性能へ影響をあたえるものとしてメモリの使い分けがあげられる．GPU には複数

の特徴の異なるメモリが搭載されているため，用途に応じて適切なメモリを使う必要がある．

特に利用しやすいメモリとしてはGlobalMemoryとRegisterがあげられる．

GlobalMemoryはGPU全体で共有されるメモリであることとCPU・GPUそれぞれから読み書きできることから，計算対象データや計算結果の格納・受け渡しなどに汎用的に用いられる．このメモリはレイテンシは大きめであるがバンド幅は高いという特徴を持っている．また「コアレスなメモリアクセス」ができるか否かにより大きく性能が変わるため最適化対象として重要なメモリである．コアレスなメモリアクセスの詳細な成立条件については割愛するが，基本的にはIDの近いスレッドが連続ないし近接するメモリアドレスを同時にアクセスする場合にメモリアクセスがまとめられて高いメモリ性能を得ることができる．

SharedMemoryは同一スレッドブロック内でのみ共有されるメモリである．容量こそ限定的ではあるものの，レイテンシ・バンド幅ともに高速であるため積極的に活用することが望ましい．典型的かつ効果的なSharedMemoryの使い方の例としては，再利用性や局所性のあるデータを用いて計算を行う場合に，GlobalMemoryからコアレスに読み出したデータをSharedMemoryに蓄積し，各コアによる計算をSharedMemoryとRegisterの中で行い，計算結果をSharedMemoryからGlobalMemoryに書き戻す，という使い方があげられる．この方法であればGlobalMemoryとSharedMemoryそれぞれの特徴と性能を活かして良い性能が得られることが期待できる．次章で述べるSpMVの高速化においてもこの方法を用いている．

この他にもCUDAプログラムの性能を高めるための技術は多く存在している．その中には，CPU-GPU間のデータ通信の最適化や複数GPU使用時のGPU-GPU間通信の高速化といった演算ではなく通信の最適化に関するものもあげられる．ただし，GPUの世代やホスト側の通信チップ構成などによって利用できない・効果が得られない手法もある．こうした点がGPUのアーキテクチャ更新が早いことと相まって最適化の難しさを高めてしまっていることは否めない．

その他のCUDAに関する情報や最適化手法についてはNVIDIA社の開発者向けwebサイト[2]やプログラミングガイド等を参照していただきたい．

3　GPUを用いたFEMアプリケーションの高速化

本章ではGPUを用いた有限要素法(Finite Element Method, FEM)アプリケーションの高速化について述べる．はじめに対象アプリケーションであるFEMアプリケーションの概要について述べたうえで，GPUを用いた最適化の戦略と実装方法や性能について示す．

3.1　対象アプリケーション

本稿の対象アプリケーションである有限要素法は，偏微分方程式の数値解法の一つであり，連続体力学が対象とする熱伝導解析，構造解析，流体解析，及びそれらの連成問題など，幅広い分野へ適用されている．FEMを用いた計算の手順は以下の通りであり，本研究では主に「FEM本体」を対象としている．

1. プリプロセス処理
    1. メッシュ生成
2. FEM本体
    1. データ入力
    2. 行列コネクティビティ生成
    3. 係数行列生成
    4. 境界条件処理
    5. 線形方程式ソルバー(疎行列ソルバー)
3. ポストプロセス
    1. データ処理
    2. 可視化処理

FEMでは，解析領域全体を節点(離散点)で構成される要素とよばれる微小領域に分割して扱う．支配方程式に対して要素単位で重み付き残差法を適用することで要素剛性マトリクスを計算し，それを全節点数と自由度の積の次元を持つ全体行列へ足し込むことで全体剛性行列を計算する．したがって，全体剛性行列は差分法と同様に疎である．要素剛性マ

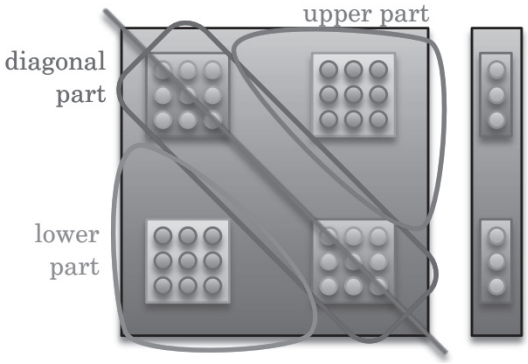


図 2: 行列格納イメージ

トリクスの全体化には各要素を構成する節点の情報 (コネクティビティ) が用いられ，全体剛性行列生成後，境界条件処理を施すことで，最終的な係数行列が求められる．最後に疎な全体剛性行列を係数行列とする連立一次方程式を解くことで，離散方程式の変数であった速度，温度，変位等が解として求められる．

FEM 本体の計算は，係数行列の生成と線形方程式ソルバによる連立一次方程式の求解に大部分が費やされ，特に線形方程式ソルバ部分が多くの時間を占める．線形方程式ソルバとしては，係数行列が疎であることから反復解法が用いられ，また行列の定値対称性から共役勾配法 (Conjugate Gradient Method, CG 法) の適用が一般的であり，前処理と合わせて適用されることが多い．

本稿では GeoFEM プロジェクト [3] で開発された並列有限要素法アプリケーションを元に整備した性能評価のためのベンチマークプログラム群の 1 つである三次元弾性静解析プログラム [4] を元に実装を行っている．本プログラムの行列格納形式は CRS(Compressed Row Storage) 形式であるが，プログラムは CPU 上での性能を上げるために、計数行列に対して 3x3 のブロック化を行い、さらに対角ブロック・上三角ブロック・下三角ブロックにわけて保持している (図 2)．

図 3 に対象プログラムの全てを CPU によって計算した場合の実行時間内訳を示す．実験には表 2 に示す環境を用いており，対象問題の大きさは 80x80x80 の単純な立方体形状である．図 3 から，1CPU による逐次実行についても，OpenMP を用いた並列実行についても，疎行列ソルバー (CG 法) の実行時間の

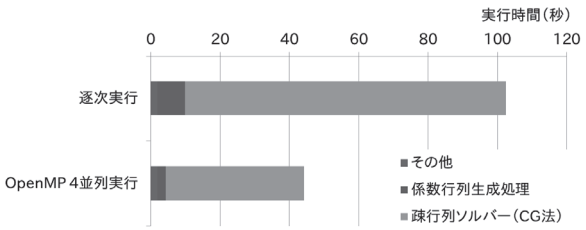


図 3: 実行時間内訳 (CPU)

割合が非常に大きいことが確認できる．実行時間の割合を見てみると，逐次実行でも OpenMP を用いた並列実行でもおよそ 90%が疎行列ソルバーに費やされている．また疎行列ソルバーに続いて時間がかかっているのは係数行列生成処理であり，逐次実行で 7.8%，OpenMP を用いた並列実行で 5.4%程度の時間が費やされている．そこで本研究ではこれら実行時間割合の大きな部位に対して GPU による高速化を行った．

3.2 GPU を用いた実装

3.2.1 CG 法の高速化

CG 法の計算内容を図 4 に示す．CG 法を構成する計算の多くは行列やベクトルに関するものであり，特に実行時間が長く大きな実行時間割合を占める必要な処理は疎行列ベクトル積 (SpMV，図 4 における $q^{(i)} = [A]p^{(i)}$) である．高度な前処理を用いた場合には前処理も大きな割合を占めることになるが，今回は非常に簡単な前処理である対角項の抽出を用いているため大きな割合を占めることはない．

GPU はベクトルや行列に関する処理に適したハードウェアである．連続する要素を GPU 上の多数のスレッドで処理することで GPU の持つ高い演算性能やメモリ性能を活用して良好な性能を得ることができる．特にベクトル長がある程度大きい問題に対しては GPU 上に大量に搭載された計算コアをフル稼働させて高速に計算を行うことができる．ただし今回のように疎行列を扱う問題の場合には間接参照やランダムメモリアクセスが必要となるため高いピーク性能比を得ることは難しい．単純な CRS 格納形式の疎行列ベクトル積においては，例えば行列の非ゼロ要素を行単位で WARP に割り当てて計算し，各

表 2: 実験環境

CPU	Intel Xeon W3520 (Nehalem, 4 cores, 2.67 GHz)
メインメモリ	PC3-10600 (DDR3-1333) 12 GB
GPU	NVIDIA Tesla C2050 (Fermi, 448 SPs, 1.15 GHz)
GPU メモリ	DDR5 3 GB
CPU-GPU 間の接続	PCI-Express x16 (Gen 2)
OS	CentOS 6.3 (kernel 2.6.32)
コンパイラなど	gcc4.4 (4.4.6 20120305 (Red Hat 4.4.6-4) (GCC)) CUDA 4.2 (release 4.2, V0.2.1221) main compiler option: -O3 -arch=sm_20

```
Compute r (0) = b-[A]x (0)
for i= 1, 2, …
    solve [M]z (i-1) = r (i-1)
    ρ i-1 = r (i-1)  z (i-1)
    if i=1
        p (1) = z (0)
    else
        β i-1 = ρ i-1 / ρ i-2
        p (i) = z (i-1)  + β i-1 z (i)
    endif
    q (i) = [A]p (i)
    α i = ρ i-1 /p (i) q (i)
    x (i) = x (i-1)  + α i p (i)
    r (i) = r (i-1)  - α i q (i)
    check convergence |r|
end
```

図 4: CG 法の計算内容

行内の足しあわせを CUDA アーキテクチャに最適化された高速なリダクション演算によって処理すれば良好な性能が得られる．しかしながら，本研究で扱っている疎行列は高速化のために 3x3 ブロック化が行われている．そのため，単純な CRS 形式疎行列向けの SpMV ではなく 3x3 ブロック化された行列向けにより高速な実装を行う余地がある．CPU では単純な CRS 形式行列の SpMV よりも 3x3 ブロック化された行列の SpMV の方が良い性能が得られていたのに対して GPU ではどのような性能となるのかも評価の意味がある．

2 章で述べたように，GPU を用いて高速な計算を行ううえでは十分な並列度とコアレスなメモリアクセスを得ること，そして可能であれば SharedMemory の活用などによって GlobalMemory へのアクセスそのものを削減することが重要である．並列度については対象行列の行数が十分に多ければ特に困難な問題とはならないことが予想できる．一方のコアレスなメモリアクセスについては工夫が必要である．例えば 3x3 ブロックを 1 つのスレッドに担当させて計算させることは可能であるが，この場合は同一 WARP に属する連続するスレッドが 9 要素ごとのメモリをアクセスすることになるため良い性能が期待できない．逆に要素数分すなわち 9 スレッドに 1 つの 3x3 ブロックを割り当てた場合，3x3 ブロックと対応するベクトルの積 (小さな密行列ベクトル積) を計算する際にはスレッド数が多すぎて無駄が大きい．また計算ごとに毎回 GlobalMemory へアクセスしていては非効率であるため SharedMemory の活用が必要である．良い性能を得るためには同一 WARP に属する少数のスレッドが協調して同一の 3x3 ブロックを処理する必要があると考えられる．

そこで今回は 3 スレッドで 1 つの 3x3 ブロックを担

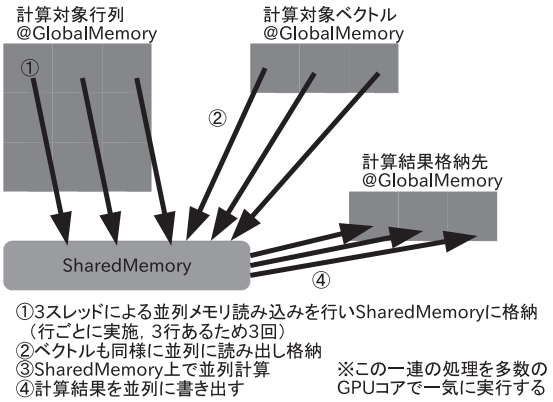


図 5: 3x3 ブロックに対する SpMV

当するという実装を採用した．まずはじめに 3 スレッドで 3x3 ブロック 1 つ分すなわち 9 要素を取得し，SharedMemory にコピーする．次に SharedMemory を活用して 3x3 ブロックとベクトルの積を計算する．最後に結果を GlobalMemory へ書き戻して終了とする．もちろんこれでは WARP を構成する 32 スレッドがフル活用できないため，30 スレッドを用いて 3x3 ブロックをまとめて 10 個処理するようにした．この実装であれば全ての GlobalMemory へのアクセスが連続になるわけではないが，ある程度の局所性と並列度が保たれるため良好な性能が期待できる．

表 3 に SpMV 1 回あたりの実行時間を示す．参考情報として 3x3 ブロック化と対角・上三角・下三角の分離を行わずに同じ内容の SpMV を実行した際の性能 (3x3 ブロック化無し) も示している．この実行結果から，3x3 ブロック化は CPU でも GPU でも性能向上に寄与できていること，GPU による 3x3 ブロック SpMV が最も良い性能を得られていることがわかる．ここで CPU と GPU の性能 (3x3 ブロック化有り同士) を比較してみると，3.2 倍の性能が得られていることがわかる．この性能比率は CPU と GPU の理論 GFLOPS 値の比である 12.1 倍よりもむしろメモリバンド幅の比である 5.2 倍に近い．これは SpMV がランダムメモリアクセスが多くメモリ性能律速であることと合致している．

さらに，SpMV 以外の各種の計算も GPU 向けに実装した．行列やベクトルの単純な四則演算は各要素に対する計算を GPU 上のコアに順番に割り当てて処理を行わせれば高い並列度とコアレスなメモリアクセスが達成されて妥当な性能を得ることができる．これらの実装を行ったことにより CG 法全体の実行時間は図 6 に示すように大きく減少した．

表 3: 1 回あたり SpMV の実行時間 (括弧内は GFLOPS 性能換算)

	CPU	GPU
3x3 ブロック化無し	96 msec (2.62)	52 msec (4.85)
3x3 ブロック化有り	71 msec (3.55)	20 msec (11.36)

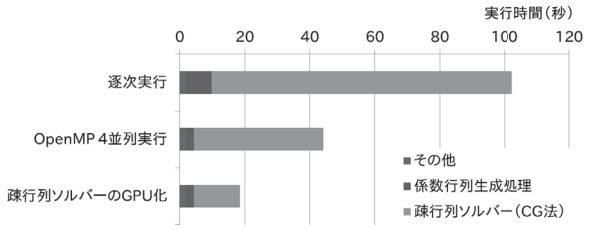


図 6: 実行時間内訳 (疎行列ソルバーの GPU 化まで)

3.2.2 係数行列生成の高速化

GPU を用いて疎行列ソルバー (CG 法) を高速化した結果，全体の実行時間に対する実行時間比も減少した．しかし依然として約 77%が疎行列ソルバーに占められている．一方で実行時間割合が 2 番目に大きかった係数行列生成処理の割合は約 13%へと上昇している．そこで本項では係数行列生成処理を GPU 化することにする．

まずは係数行列生成処理の手順と CPU 向けの並列化手法について確認する．係数行列生成処理の概要 (プログラムの概形) を図 7 に示す．図からわかるように，係数行列生成は大きく 2 つの多重ループに別れている．2 つの処理のうち前半のループについては計算量も実行時間も小さいため，今回は後半のループのみに着目する．

係数行列生成処理の後半ループ部は対象問題の要素や節点に対してそれぞれ処理を行うものであり，対象問題の規模がある程度大きければ十分な並列度が確保できると期待される．しかし，要素毎に計算した結果を全体行列に足し合わせる処理が必要である

```
do k = 1,2
 do j = 1,2
  do i = 1,2
   ガウス積分点(8点)における形状関数とその自然座標系における微分の算出
enddo enddo enddo

do icel = 1, 要素数
 8節点の座標から，ガウス積分点における形状関数の全体座標系における微分とヤコビアンを算出
 do ie = 1,8
  do je= 1,8
   全体接点番号ip,jpを元に係数行列における位置を算出(小規模探索処理)
   do k = 1,2
    do j = 1,2
     do i = 1,2
      要素積分，要素行列成分計算，全体行列への足し込み(完全に独立ではない)
   enddo enddo enddo
 enddo enddo
enddo
```

図 7: 係数行列生成処理の概要 (プログラムの概形)

ため，ループの各イタレーションは完全に独立しておらず，単純にループを並列化することはできない．

こうした問題を解決するための方法として，カラーリング (色の塗り分け) による並列処理可能部分の抽出が知られている．すなわち，事前に全要素をチェックして同時に処理して問題のない要素群を同じ色を持つグループとしてまとめておき，色毎に並列処理を行う手法である．カラーリング自体にも様々な手法が提案されているが，今回は代表的な手法の 1 つであるマルチカラー法 (MultiColor 法，MC 法) を用いて塗り分けを行う．ただし，色の塗り分け自体は探索問題であり GPU に向いている処理ではない．そこで今回は塗り分け処理自体は CPU 上で実行し，塗り分け後の計算・足しあわせ処理のみを GPU 上で実行することにした．

計算・足しあわせの多重ループは CG 法におけるベクトル計算や SpMV 計算と比べてやや構造が複雑であり，最外の icel ループの中に ie,je ループが，さらに ie,je ループの中に k,j,i ループが内包されている．そこで，単純に全体の処理を 1 つの GPU カーネルとして実装する (図 8-a，以下「一括実装」) 以外に，ループ構成にあわせて GPU カーネルを切り分けるという実装 (図 8-b，以下「分割実装」) も考えられる．分割実装には各 GPU カーネルに最適な並列度を与えて実行することが可能であるという利点がある．また分割実装は一括実装と比べて 1 回の GPU カーネル呼び出しで実行する処理が少ないが，一般的に GPU カーネルを小さくすることは共有メモリやレジスタの使用量を減らして多くのインスタンスを同時に走らせることができるという利点がある．しかしながら GPU カーネルを分割するということは CPU による GPU 制御のための時間が増えることにつながるためいずれの実装も利点と欠点を持っ

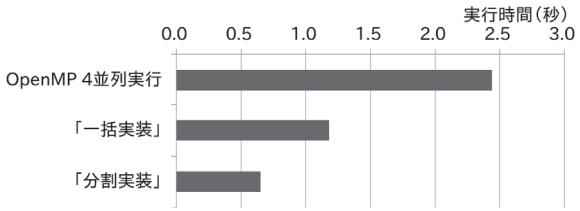


図 9: 係数行列生成処理の性能

ている．本研究ではこれらの実装をそれぞれ実装して性能の比較を行った．実装においては SpMV の実装と同様に十分な並列度の確保と同一 WARP 内でコアレスなメモリアクセスが起きやすいように努めた．ただしメモリアクセスパターンが SpMV ほど単純ではなく，また探索処理の結果によって計算対象が変わってしまう部分もあるため，対象問題の形状等によって性能が低下してしまうことがある可能性がある．

一括実装と分割実装の性能を図 9 に示す．性能を比較した結果，いずれの性能も CPU にて OpenMP を用いた場合よりも良い性能であったが，分割実装の方が一括実装より良い性能が得られていた．このデータはいずれも色数が 8 色の際の性能であるが，色数を 100 程度まで増やしても色数が性能に与える影響は無視できる程度であった．ただし，色数を 1000 程度まで大きく増やした際には CPU の実行時間がやや減少し，逆に GPU の実行時間はわずかに増加した．CPU の実行時間が減少した理由としては色数の増加にともない一色あたりの要素数が減少し，色毎に扱うデータの量が減ってキャッシュの効果が高まったことが考えられる．一方 GPU についても CPU と同様キャッシュの効果は高まっているはずであるが，色数が増えた分だけ GPU カーネルを呼び出す回数が増加し，キャッシュによるメリットを打ち消してしまったと考えられる．

CG 法と係数行列生成処理を GPU 上で実行した際の実行時間内訳を図 10 に示す．係数行列生成処理時間は確かに短縮されたが，全体の実行時間に占める割合が大きくないため全体に与える影響は限定的であった．

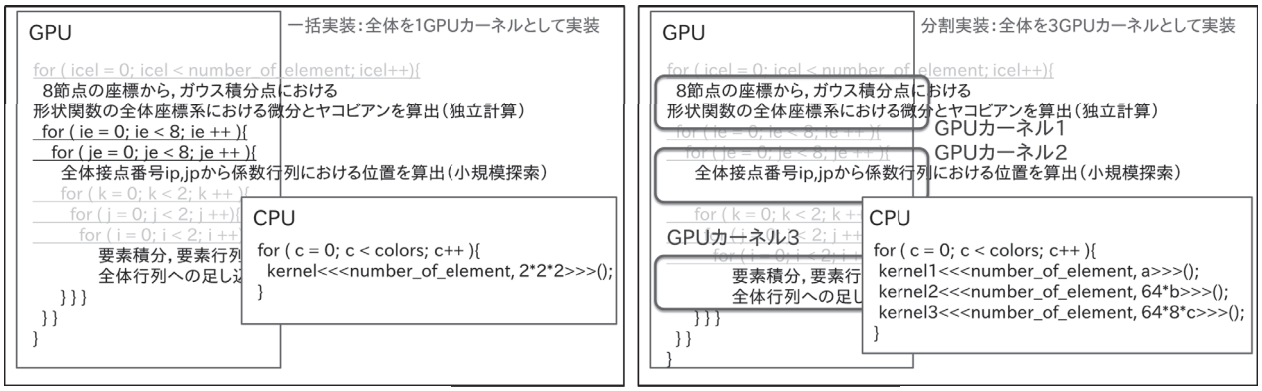


図 8: 係数行列生成の実装 (「一括実装」と「分割実装」)

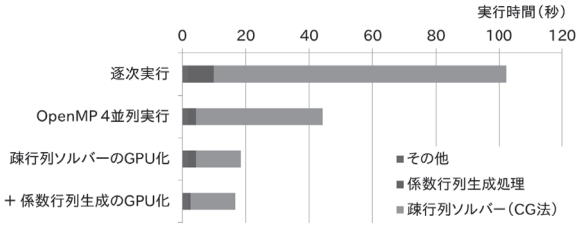


図 10: 実行時間内訳 (係数行列生成処理の GPU 化まで)

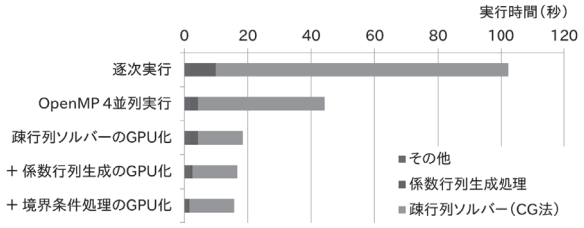


図 11: 実行時間内訳 (境界条件の GPU 化まで)

3.2.3　その他の実装

以上のように，ここまで FEM アプリケーションに対して主要な 2 部位を GPU 上で実行することで性能向上を行ってきた．本項ではこれら以外に行ったさらなる性能向上のための幾つかの取り組みについて紹介する．

まず境界条件処理の GPU 化について述べる．既に実行時間内訳のグラフで示しているように疎行列ソルバーと境界条件処理以外の処理が全体の実行時間に占める割合は小さい．今回用いている境界条件が非常に単純な端点処理のみであることもあり，境界条件処理を高速化しても全体に与える影響は微々たるものである．ただしこの処理は係数行列生成と疎行列ソルバーの間に位置している処理であるため，境界条件処理を GPU 化できれば係数行列生成から疎行列ソルバーまでの処理が GPU 上で行われることになり，CPU-GPU 間のデータ通信が減るなどの効果によって全体の実行時間が有意に短縮される可能性がある．そこで境界条件処理を GPU 化するとともに CPU-GPU 間のデータ転送処理等を見直し，係数行列生成の後半 (計算・足し込み処理) から疎行列ソルバーまでのほとんどの処理を GPU 上で行うようにした．これにより図 11 に示すように全体実行時間をわずかではあるが削減することができた．

ところで，GPU は PCI-Express バスの本数や電源容量が許せば 1 台の PC に複数搭載することが可能である．また CUDA では 1PC に複数 GPU を搭載し利用することができる．そこで複数 GPU を用いた FEM アプリケーションの高速化についても取り組んだ．

疎行列ソルバーの複数 GPU 化については，単純に行列やベクトルを均等分割することで対応した．基本的には用いる各行列とベクトルを GPU 数で分割し，必要に応じて計算結果の交換等を行えば正しい解を得ることができる．対象問題によっては適当に分割してしまうと行ごとの非ゼロ要素数の都合により実行時間に差が生じる可能性があるが，今回は単純な問題設定であることもあって単純な分割である程度の性能向上が得られることがわかった．1GPU では 1 回あたり 20 ミリ秒かかっていた SpMV は，

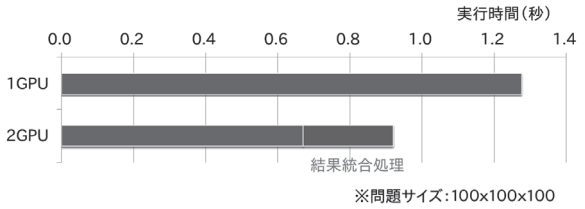


図 12: 2GPU を用いた係数行列生成処理の実行時間

2GPU で実行すると 13 ミリ秒に短縮された．しかし，SpMV 単体の性能はともかく，疎行列ソルバー全体の性能については収束判定のための計算結果とりまとめや GPU 間のデータ交換などオーバーヘッド要因が少なくはないため，1GPU で 14.15 秒かかっていた疎行列ソルバーの実行時間は 2GPU を用いても 12.07 秒にしか短くならなかった．さらに高い効果を得るためには，CPU-GPU 間・GPU-GPU 間の通信の最適化などを行う必要があり，配列の保持方法などから再考が必要であると考えられる．

係数行列生成処理の複数 GPU 化はさらに難しい問題である．疎行列ソルバーのように単純な分割を行うことはできず，計算結果の足しあわせ先がどこになるかを意識した分割が必要である．また足しあわせ先情報に基づいてデータ分割を行おうとすると今度は計算元のデータを単純には分割できないという問題が生じる．このような分割処理を行いたいのであれば，そもそも MPI 向けの実装のように FEM アプリケーション本体ではなくメッシュ生成の段階で問題を分割することを考えるべきである．そこで今回は参考として，計算元データは 2GPU それぞれで全て持ち，足しあわせ先の位置によって担当する GPU をわけ，係数行列生成処理の結果を CPU 上で併合して利用するという実装を行った．このときの実行時間は図 12 の通りであり，予想した通り計算・足しあわせ時間そのものは短縮された一方でデータの併合に無視できない長さの時間がかかってしまった．

このように，複数 GPU を用いた実装についてはまだ改善の余地がある．また上述のように，そもそも 1 プロセス内で複数の GPU を扱うよりもノード内で MPI を使用し MPI プロセスごとに単一の GPU を用いる方が良い性能が得られる可能性もある．これらも含めてさらなる実装と性能評価の余地があると考えられる．

4　おわりに

本稿では GPU を用いた FEM アプリケーションの高速化について紹介した．疎行列ソルバー，係数行列生成，境界条件処理を GPU に実行させることにより，1CPU による逐次処理と比べて実行時間を 15.3%にまで，また OpenMP を用いた並列処理と比べても 35.6%にまで短縮することができた (図 11)．本文中では CPU に Nehalem，GPU に Fermi というそれぞれ 1,2 世代古いハードウェアを用いていたが，より新しい世代の CPU や GPU でも本稿と同様の最適化手法は利用可能であり，程度は違えど性能向上が得られるだろう．

現在は 3.2.3 項で述べた複数 GPU を用いた実装などに取り組む一方で，ボルドー大学にて開発されている不均質環境向けの実行時ランタイム StarPU を用いた FEM アプリケーションの高速化にも取り組んでいる．またその他の今後の課題としては，最新世代 (Kepler アーキテクチャ) の GPU を用いたより高速な実装，疎行列ソルバーにおける複雑な前処理の実装，本研究の成果を他のアプリケーション等で活用できるようなライブラリ化などがあげられる．

参考文献

[1] 大島聡史:　これからの並列計算のための GPGPU 連載講座 (1) GPU と GPGPU の歴史と特徴, 東京大学 情報基盤センター スーパーコンピューティングニュース, Vol.12 No.1 (2010 年 1 月).

[2] NVIDIA, NVIDIA Developer Zone `https://developer.nvidia.com/category/zone/cuda-zone`.

[3] GeoFEM `http://geofem.tokyo.rist.or.jp/`.

[4] Nakajima, K.: Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC2003 (2003).

交通計画分野の高速処理演算需要

山﨑 浩気

工学研究科 都市社会工学専攻 助教 （兼任：学術情報メディアセンター コンピューティング研究部門 助教）

1 はじめに

近年，交通に関するあらゆるデータに対して大規模な収集が進みながらも，なかなか有効活用がなされていない現状がある．たとえば，我が国の有料道路の料金収受システムより得られるETC(Electric Toll Collection)データは，個人情報の保護に十分配慮した形でも，個別行動履歴データとして活用可能であり，高速道路ネットワークの整備効果を利用者行動変更の観点から評価分析することも可能である．また，高速道路上の交通流動の解析手法ならびにサービス水準評価，マネジメント手法の構築のためには，道路交通の管理者側に時々刻々蓄積されていく，いわゆるビッグデータに対して，データ処理・抽出・解析の段階で活用可能な高速道路データベースを構築したり，交通流データの並列計算技術を活用した大規模高速処理，オンライン交通データ利用に関する手法論を検討することが重要となってきている．

本稿では，これら交通計画分野で高速処理演算が求められている研究事例について紹介する．

2 研究事例１：ETC 行動履歴解析

ETC データには料金収受情報の一部としてカード番号が記載されており，同一カードユーザの行動を経時的に観測することが可能である．そこで，高速道路利用の習熟度がある程度高いと考えられる高頻度ユーザの行動に着目した行動分析をおこなうことで，往復での利用やチェーンでの利用，あるいは週末のみの利用など様々な周期性をもった行動として高速道路の利用特性についても考察できる．また，新規道路開通や集中工事，料金値下げの社会実験などの交通状況に大きなインパクトとなる事象の事前事後の交通行動を分析することによって，高速道路利用の経時的変化についても分析可能であろう．

一方で，ETC データにはそのデータ特性上，分析困難な事例も存在しているため，現時点における ETC データによる交通行動分析の抱える問題点について列挙することにより，今後の交通行動モニタリングに関するデータ収集の知見をまとめる．

2.1 分析対象

まずは，経時的交通行動分析の視点で ETC データを解析する．ETC カード番号に関する情報に着目して，高速道路供用前後での利用者行動に着目し，その変化を把握している．これにより限定的ではあるものの，新規路線供用によって，実際に利用 IC を変更しているユーザーの行動パターン，利用回数等に着目して，分析方法の検討をおこなうことが可能である．分析対象地域として新名神高速道路開通により交通行動に影響を与えると考えられた，琵琶湖の湖南地域（図-1）を対象として対象地域内の IC より流入をおこない，何度も行動履歴が記録された高速道路を多頻度で利用するユーザーの利用データのみを抽出して用いた．これは，従来の旅行速度・交通量の量的変化に着目して新規路線開通効果を分析する手法と比較すると，湖南地域以外で流入出をおこなう湖南地域通過交通や高速道路低頻度利用者によるトリップを対象としないため，限定的に新規路線開通効果を評価することになる．しかし，本手法で着目するのは，従来用いられてきた量的な変化では捉えられなかった，新規路線開通に伴い個々人の高速道路の利用形態が変化するという，交通の質的変化を捉えることであり，限定的な高速道路利

用者の行動変化の把握に留まるものの，その意義は少なくない．また，利用形態を変化させないユーザー，利用回数を減少させるユーザーの行動については，ETC 仮 ID データを用いても行動を捉えることが困難であると考えられるため，ひとまず分析対象から除外する．

図-1　個人行動分析対象地域（湖南地域）

2.2 交通変化をとらえるためのデータ活用法

ETC データを活用する本研究の特徴としては，事前事後比較が個人の単位で分かること，非常に多様なトリップデータに対して分析がおこなえること，空間的変化を捉えられることなどが挙げられる．一方，課題としてはトリップの真の出発地・目的地が分からないなど個人属性に関するデータが不足している点が挙げられる．

つまり，都市間高速道路新規供用に伴う利用行動の変化に着目して分析をおこなうこととなる．供用前後で行動変化を確実に把握するため，2007 年，2008 年の各分析対象の 3 ヶ月間で 20 回以上湖南地域の IC から高速道路に流入したユーザーを対象とする．以下では着目する行動変化の性質ごとに分けて，その可能性を列挙する．

i.　新規利用開始ユーザーに関する分析

まず新規路線開通に伴い高速道路を利用するという新規利用開始行動が考えられる．すなわち，「08 年度新規利用開始ユーザーが多いのはどの IC なのか」，「(IC 転換行動が起こっていると推察した）栗東 IC の新規利用開始ユーザーは少ないのか」といった視点で分析をおこなう．ここで，新規利用開始ユーザーとは，「08 年度のみに高速道路利用が記録されているユーザー」であり，

1)　07 年度に一般道を利用していたユーザー
2)　ETC 車載器を 08 年度にセットアップして，ETC 利用を始めたユーザー
3)　カードの期限更新などで，07 年度とは違う ETC カードを使用しているユーザー

の 3 種類のユーザーを含んでおり，これら 1)〜3) の分類は ETC データの特性上不可能である．しかしながら，特定の IC，OD のみで突出して 2) や 3)の割合が高いとする明確な根拠もないことより，本研究では IC，OD によらず 1)のような誘発交通需要に当てはまるユーザー割合はほぼ一定であると考えている．新規利用開始ユーザー数を把握することで，IC ごとの新規ユーザー比率が分かるため，誘発交通需要が多い IC を明らかにすることが可能となる．

ii.　利用回数の変化に関する分析

一方で，従来から高速道路を利用していたユーザーに対しても，さまざまな行動変化が生じていることが考えられる．まずは，利用回数の変化が考えられる．特に，開通事前事後で高速道路利用回数が増加するユーザーは，影響要因考察も比較的容易に可能であるため，分析対象ユーザーをサービス改善面に絞って詳細に分析してみる．サービス悪化の影響を被ったユーザーについては，その分析手法を十分に検討する必要がある．

iii.　利用距離帯の変更に関する分析

次に利用距離帯の変化が考えられる．新名神高速道路開通に伴い高速道路利便性が向上して，短距離帯である湖南地域の内内トリップについても高速道路利用増加や新規ユーザーの獲得につながったことから，渋滞が頻発する地点が生じたことが推察される．

iv.　利用 IC の変更に関する分析

新名神高速道路上に新設された 2 つの IC（信楽・甲賀土山）の影響により，各ユーザーが目的地に合わせて IC を選択している傾向があることも推察される．比較的広域の移動で，目的とする IC が湖南地域を基準として西側に位置する場合は以前から選択していた名神高速道路 IC から流入をおこなうが，目的 IC が東側にある場合は，新名神高速道路経由ルートの方が所要時間が短いなどの要因があるため，上手く使い分けをおこなう可能性が挙げられる．このことに関して，ETC 履歴データを用いて，新規路線開通に伴う湖南地

域内での利用 IC の変化を把握することは非常に大きな意義があると考える．

以上のように，ETC データを活用することによって，新規路線供用効果を実証可能と考える．そこで，次のような分析をおこなうことで，行動変化を定量化して供用効果検証を試みる．

A) 利用回数変化に関する分析
B) 利用 IC 選択肢に関する分析
C) 流入 IC，流出 IC，利用回数などの利用パターンに関する分析

2.3 ETC データを用いた IC 選択行動モデル

実際に時間帯，方向，車種の 3 つを考慮したロジットモデルを作成する．ここでは，栗東 IC に対する効用を説明変数を用いて算出して，甲賀土山 IC を選択する効用を定数項にて表現する．用いる説明変数は，以下のとおりである．ETC データから抽出してきた説明変数候補（選択要因）をまとめた結果，合計 4549 トリップについて，以下の情報を作成して栗東 IC を選択する効用と，他 IC を選択する効用にてモデル推定を行った．

x_1 :深夜・早朝ダミー（0:00～9:59 もしくは 21:00～23:59 の時間帯であれば，1．それ以外の時間帯であれば，0.）

x_2 :昼ダミー（10:00～16:59 の時間帯であれば，1．それ以外 0.）

x_3 ：西向きダミー（流出 IC が湖南地域よりも西側であれば，1．東側であれば，0.）

x_4 ：（実際所要時間）―（その時間帯の 3 ヶ月の間の IC 間平均所要時間）によって算出される所要時間差

x_5 ：車種

$d_1 \sim d_5$ ：パラメータ

$$Ritto = \exp(d_1 \cdot x_1 + d_2 \cdot x_2 + d_3 \cdot x_3 + d_4 \cdot x_4 + d_5 \cdot x_5) \quad (1)$$

$$Other = \exp(\text{定数項}) \quad (2)$$

表-1 の t 値より，西向きダミー，所要時間差の 2 つの因子が栗東 IC 選択に影響していることが分かる．このとき，西向きに高速道路利用する際には栗東 IC 選択を行う．また，栗東 IC から高速道路利用を行った方が所要時間が大きくなると判断したとき，すなわち，より長く高速道路利用を行いたいと判断したときには栗東 IC を選択する傾向が強いことが分かった．車種についても 5%水準で有意に影響していて，中型・大型・特大車であるときに栗東を選択する効用が低くなることが分かる．

以上より，クラスタ分析によって絞り出されたあるグループ内のユーザ行動に対して，ETC データのみから得られる情報をうまく活用して行動モデル推定をおこなえることが示された．

表-1　IC 選択モデル推定結果

	パラメータ	t値	
定数項	2.915	14.352	*
深夜・早朝ダミー	0.042	0.465	
昼ダミー	−0.168	−1.484	
西向きダミー	3.458	21.746	*
所要時間差	0.018	14.878	*
車種	−0.335	−2.681	*
ρ 2値	0.210		
ρ 2値（修正済）	0.208		

＊p<0.05

3　研究事例２：交通流並列計算

IT 技術の発展により日々蓄積される交通データは，いまだ充分に活用できている状況とはいいがたいが，これら工学的に活用出来るデータを限定された時間内で効率的に処理して，サービス水準の低下を招く原因の特定・診断をおこなうことは重要な課題である．21 世紀に入って以降，情報学分野における並列計算技術などの計算処理プロセス研究は爆発的に進歩を遂げており，情報爆発時代における最先端の IT 基盤技術は他分野との融合を積極的に図る段階にあると考えられる．

交通工学分野において計算処理プロセス研究が活用出来る事例として，交通サービス水準の阻害要因の系統的把握がある．そこで，計算速度・計算可能範囲に優れて，大規模エリアで計算可能な，並列型交通流シミュレーション構築に向けた交通流の数理的表現に関して基本条件の考察をおこなう．流体模型を援用して，スーパーコンピュータの高い計算力を活かした並列型交通流シミュレーション構築をおこない，将来的には道路施策の効果検証へと繋げられる．

3.1 ミクロ交通流シミュレーション

地点的な交通問題に対するアプローチであり，車一台毎の挙動を考えるミクロ交通流シミュレーションにおいては，限られた数日間の観測データを元に交通状況を再現する事例が多くおこなわれてきた．しかしながら，地点的な問題を捉える上で，渋滞を誘引している要因は様々であり，豊富なデータ解析が望まれてきている．また，道路上の渋滞は部分的に解消出来ても根本的な解決とはならず，少なくとも高速道路のIC（インターチェンジ）区間単位をターゲットとした状況再現が，今後必要となってくる．地点的な交通阻害要因が組み合わさり，日々異なる渋滞状況が生じているため，ミクロシミュレーションモデルを並列で用いて，より広い分析対象地域において渋滞状況解析をおこなっていければ，より正確な渋滞要因解明が出来るであろう．本研究では，ミクロ交通流シミュレーションにマクロモデルで用いられる流体模型の表現方法を用いて，ある程度車両一台ずつへの計算量を簡素化することによって，より広いエリアに関して状況再現をおこなうことを考える．

3.2 交通流の数理的表現

既往のミクロシミュレーションモデルにおいては，パネルやセルと言った概念が見られるが，並列処理をおこなう上では離散的に取り扱う項目が多くなるため，通信量の負荷の面からこれら概念の活用は現実的では無いと考える．そこで，マクロ交通流モデルで用いられる流体模型における数理的表現を参考とし，並列シミュレーションに適切な数理表現について考える．ミクロ視点のうち，

$$\frac{d^2x_n(t)}{dt^2}=a\left(U(x_{n+1}(t)-x_n(t))-\frac{dx_n(t)}{dt}\right) \quad (3)$$

式(1)追従挙動モデル(最適速度関数)

t：時間[sec]，$x_n(t)$：時刻tにおけるn番目の座標

a：感応度パラメータ

$$\frac{d\rho}{dt}+\frac{d\rho u}{du}=0 \quad (4)$$

$$U\left(x_{n+1}(t)-x_n(t)\right)=u_{\max}\left(1-\frac{1}{\rho_{\max}(x_{n+1}(t)-x_n(t))}\right)$$

・・・(5)

$$\frac{du}{dt}+u\frac{du}{dx}=\frac{1}{T}(U(\rho)-u)-\frac{a^2}{\rho}\frac{d\rho}{dx} \quad (6)$$

式(4)～(6)　Navier-Stokes方程式を中心とした交通流数理モデル基礎方程式(未知数ρ，u)

ρ：車の密度，T：時間遅れ，u：車の平均速度

V(ρ)：車の密度で決まる速度関数，l：定数

前方車両との速度差と位置関係の2要素をもとに自車の加速度を決定するという，追従挙動モデル研究では，最適速度(Optimal Velocity)関数が坂東らによって提案されている．また，最適速度関数に適した関数形として，双曲線関数 tanh を用いたものがよく知られている．

追従挙動モデルで，全ての項目が関数形で表現出来れば，シミュレーションをおこなう上でも扱いやすい．式(3)は，車が前方車間距離に応じて加減速をおこなう一般的な速度条件式を表している．この中で $U(\Delta x)$の関数形を検討すること，および感応度パラメータに対する分析が交通工学的におこなわれてきている．ここで，マクロ交通流モデルで用いられる流体モデルのうち Navier-Stokes 方程式を基本として渋滞を考慮した式形と，最適速度関数 $U(\Delta x)$には，数理的に類似した項目が見られ，交通流の特徴記述をおこなう上で用いやすい．

式(4)～(6)のように，基礎方程式をおく．式(5)より，最適速度関数に対して交通量と密度の関係性を規定すると，水面形の運動方程式として交通流を記述することが出来る．式(6)の流体模型の典

型的な運動方程式に対して，速度関数形へと書き換えをおこなう．ここで，Kerner-Konhauser 模型と呼ばれ，粘性項を定数の逆数で表現することで，密度が増加する区間において車が速度を出しにくい状況となることを表現することも可能となる．慣性項・圧力項に関しても，車の密度 ρ と車の平均速度 u の関係式記述によって，交通流が流体模型によって表現されている．また，右辺第 3 項，外力(粘性)の非線形項が並列化をおこなう上で取扱いが難しくなると考えられる一方で，シミュレーションをする上ではその数理的特性が地点による密度と速度の細かな差異が結果を左右する重要な因子となるため，今後の検討課題として考えている．また，今回示している流体模型を用いて並列計算をおこなう上で，この関数形を基本として速度関数を密度 ρ だけでなく，地点的な構造特性などを変数として加え，速度関数を定義することで，交通状態に影響を与える要因を表現出来るであろう．

これらの基礎方程式に対して数値解析手法を用いることによって交通流シミュレーションをおこなう．

3.3 交通流の数理的表現

ここでは，ミクロな追従に関する最適速度関数の考えを導入して，交通流シミュレーションにおいて用いられているマクロモデルに関する数理的表現が Kerner-Konhauser 模型によって可能であることを確認している．すなわち，マクロな都市間とミクロな地点の中間概念に位置する，高速道路区間に存在する IC 区間内の渋滞連鎖を解析することが可能となると考えている．計算負荷を軽減するため，複数のパソコンでそれぞれ異なる区間をシミュレーションする上で，ひとつの式形で基礎式が定義されることが重要である．

並列化計算した上での解析方法について，ここで目標を概説すると，高速道路上に 2km 毎に設置された交通検知器データ区間内の交通流動を即時的に処理，模擬できる交通流シミュレーションの開発を考えている．例えば，速度関数 $V(\rho,y,z)$ について(y，z は任意の変数)，その区間の構造要因や交通状況要因に対して変動が起こる関数形として表現することによって，流体模型より援用した式形ひとつの支配方程式に基づき，並列的に地点毎の異なる交通変動要因を分析出来るシミュレータとなるであろう．

今後は，実際の交通流データを用いる際の並列型交通流シミュレーションにおける大きな課題である変数の取扱い方法や交通の非定常状態に関する検討をおこなっていく．また，並列処理による実際の交通状況との差異について考察するため，既往のミクロシミュレーションモデルの演算結果と比較をおこなうことを考えている．

4　研究事例 3：交通データの並列処理

限定された時間内で効率的にデータ処理して，サービス水準の低下を招く原因の特定・診断をおこなうことは重要な課題である．そこで我々は，並列アルゴリズムや並列ライブラリを検討することで，主に以下の 2 点に取り組んでいる．

1) 計算速度・計算可能範囲に優れた交通流シミュレーションの並列アルゴリズム構築
2) 膨大なデータを即時的に処理し，道路利用者に必要な情報を即座に還元できる即時的サービス水準評価アルゴリズム検討

交通工学・交通計画の分野においても従来型の少量のデータからより多くの評価・分析情報を引き出すという研究スタンスから，時空間的解像度の高い大量のデータを効率的に処理し，交通状態の変動を的確に把握するというスタンスに変革しつつある．このように研究パラダイムの転換が進みつつある状況を鑑みると，円滑かつ安定的な交通サービスを提供する上で，日々蓄積され長期間にわたる種々の観測データの長所を活かし，適切に組み合わせることが望まれる．また，道路ネットワークのサービス水準を定量的に評価し，サービス水準の低下を招く原因の特定・診断，交通システムのマネジメントに関する方法論の開発をおこなうことが重要な課題である．我が国は成熟社会を迎えており，個人の移動に対する時間価値が大きくなっている．そのため，予想外の遅延が深刻な損失を招く可能性も高く，ただ単に旅行時間が短縮されるだけでなく，常にある時間の範囲内で目的地に確実に到達できるという，安定的な道路交通サービスの提供が求められている．すなわち，今後は「どの程度の時間で確実に到着できる

か」，「何時までには確実に到着できるか」というサービスの質，旅行時間信頼性という概念を用いて評価して，それらの有益情報を利用者へと還元して，賢い運転で交通渋滞の現象に寄与してもらうシステム改善が重要である．旅行時間信頼性の観点から道路ネットワークを評価するためには莫大なデータが必要であり，データ制約上の理由からこれまで旅行時間信頼性を実証的に評価することは困難であった．近年では，高速道路においてETC(Electronic Toll Collection)搭載車が飛躍的に増加しており，料金収受によって得られる行動履歴データだけでなく，本線上に設置された交通量検知器データ，道路構造要因データ，天候データ，事故・工事規制情報等といった多種多様なデータを統合的に活用する必要がある．しかしながら，統計分析に耐えうるデータ作成を効率的におこない，即時的に事象解析をおこない交通情報を作成するアルゴリズム構築に対する道路管理者側の需要はあるものの，いまだにその手法論の検討は充分でない．

一方，情報学分野における並列計算技術，グリッドコンピューティングなどの計算処理プロセス研究は，21 世紀に入って以降，爆発的に進歩を遂げており，情報爆発時代における最先端の IT 基盤技術は他分野との融合を積極的に図る段階にあるものと考えられている．即ち，大量で多様な情報から真に必要とする情報を効率良くかつ偏りなく安心して取り出すことを可能とする技術，大量の情報を管理する大規模な情報システムを安定・安全に運用するための新しいサステナブルな技術，並びに，人間同士の対話により誰もが容易に情報を利活用できるようにする技術の研究が進んでいる．さらに，多様な情報を活用した先進的な IT サービスを人間社会に受け入れ易くするための社会制度設計も視野に入れ，情報学諸分野において様々な先端的手法を有機的に融合することによる総合的取組みがおこなわれている．

現在，理論・実験に加えてコンピュータによる計算・シミュレーションは重要な研究手法となっており，並列アルゴリズムや並列ライブラリを検討することで，大量データを即時的に処理して利活用することが可能となった．そのため，進展する IT 化の中で大量の交通データから交通工学・交通計画にフィードバックできる研究結果，例えば交通サービス水準の阻害要因の系統的把握，オンライン利用を想定した突発事象の精度の高い自動検知などの方法論を得られるよう，スーパーコンピュータの高い計算力を活かすための，交通データ処理方法検討，および演算性能が速い並列型交通流シミュレーション構築を目指す．これまで充分な検討がおこなわれていない先端的なデータマイニングの方法論の適用も視野に入れ，各種データを長期間にわたって時系列的に統合したデータベース構築をおこない，分析する単位に合わせて統計分析に耐えうるデータセット作成をおこなうアルゴリズム構築を目指している．道路利用者に還元し得る有益情報を即時的に計算，提供できるアルゴリズムを構築することで，将来的には交通変動の少ない安定的な道路運用に繋げたい．

交通の質を評価する際，旅行時間信頼性という評価指標の重要性は概念的には広く理解されているが，分布形を得るためには，多くのデータを収集する必要があるため，実測データに基づき旅行時間信頼性を評価した研究事例はわずかである．また，従来型の交通シミュレーションにおいては，限られた数日間の観測データをもとに推測するのが常であった．申請者は，日々大量に蓄積されている交通に関するデータを活用して，サービス水準を同定するための方法論の提案をおこなってきている．今後，進展した計算処理プロセス研究を活用して，多種多様なデータの統合的な扱い，即時的処理に関する融合的研究をおこなうことにより，簡易な操作で大規模データハンドリングをおこないながら，広域から局所的な問題まで包括的に交通サービス水準を評価・診断できるシステム構築へと繋げられることが本研究の学術的な特色である．

高速道路上で得られるデータは，さまざまなユーザー層のデータが混在して記録されているため，着目するユーザーを選定する作業にある程度の処理が必要となるが，交通行動に影響を与えるインパクトの事前事後における空間的・時間的行動変化を見る上で，その活用法が提案できるならば，非常に有意義なデータとなり得る．例えば，2009 年 3 月末に経済対策の一環として，ETC 車は休日高速道路 1000 円という施策が取り入れられたものの，2011 年 6 月をもって一旦凍結されている．この間の料金体系の変遷のもとで工学的に非常に

価値があると考えられるユーザーの行動履歴データは日々蓄積されているが，あまりにも大規模であるがために全てのデータを解析出来ず，得られた知見を高速道路利用者にいまだ還元したとは言いがたい．ETC データは，プライバシーの問題等克服すべき課題はあるが，空間的・時間的行動変化を理解する上で貴重なデータであり，分析手法の検討をおこなうことで新たな交通計画，交通管理の端緒となる可能性を秘めている．利用者にとってニーズの高い正確な交通情報や管理者にとってニーズの高い正確な交通シミュレーションを生成するうえで，空間的・時間的密度の細かな大規模データを効率的に処理するアルゴリズム生成が望まれる．情報技術高度化の流れの中で，従来ならば紙ベースで記録されていたものも電子化されてきたことも踏まえ，大量かつ良質なデータをそれぞれの長所を活かし，適切に組み合わせて用いることで道路ネットワークのサービスレベル解析に向けた，即時的な情報提供アルゴリズムを検討していくべきであると考える．また，円滑かつ安定的な交通サービスを提供するためには，道路ネットワークのサービスレベルを定量的に評価するとともに，サービスレベルの低下を招く原因の特定・診断をおこない，道路交通マネジメントの高度化を図ることで，提供サービスの水準を向上させることが強く望まれる．多様なデータを活用して道路ネットワーク評価の方法論を開発して，道路網改善計画に関する合意形成を支援する分析・評価手法の構築を図ることが求められる．

上記の研究課題に関して，いくつかのテーマに取り組んでいる．たとえば，

4.1 多種データの計算処理プロセス改善に関する研究

ETC データ・交通量検知器等の大量に蓄積された交通データを，分析可能なレベルのデータベースに作り込むための，計算処理アルゴリズム構築に関する検討をおこなう．ここで作成したデータベース，即時処理計算手法を研究基盤として，道路ネットワークのサービス水準評価手法，正確性が高いかつ演算性能の速い並列型交通流シミュレーション検討へとつなげていく．

4.2 道路交通サービス水準の影響要因把握のための統合データベース構築に関する研究

道路交通サービス水準への影響要因を把握するため，長期間の交通データを統合する手法を検討する．個々の影響要因データを一元的にまとめた統合データベースを作成し，複雑に相互影響を与え合うサービス変動要因をひも付けするため，どのような分析手法が適当かについて検討する．

4.3 ETC カード ID に着目した高速道路利用行動データの処理手法に関する研究

休日高速道路 1000 円や東日本大震災を踏まえた料金体系に関して，同一カード利用者の利用行動履歴の経時的かつ自動的な観測可能性に着目し，ETC の仮カード ID 情報のデータベースを構築し，これを効率的に処理する手法を検討する．利用者行動の面から交通サービス水準の変化の影響を把握するために，先端的なデータマイニング方法の適用を試みる．

4.4 並列計算手法を用いた交通シミュレーション高速化検討に関する研究

流体力学の分野で利用されている非圧縮性流体の並列シミュレーション技術を活用して，交通流計算を区間分割して効率的に計算するアルゴリズムを検討する．これにより，従来よりも高速でかつ大規模なデータを取り扱った，正確性のより高い交通状況予測を目指す．

4.5 即時的サービス水準評価情報の提供アルゴリズムに関する検討

提案した方法論を活用し，交通に関する多種データの高速化・効率化をおこなうことにより，日常的な交通マネジメントにおける即時的，かつ効率的に，利用者に有意義な情報提供を検討する．また将来的には，現状の交通状況の即時処理だけでなく，将来の交通変動を考慮した交通状況予報をおこなうアルゴリズムを構築し，それを利用者へと情報還元していける，実効性のある道路マネジメント手法を提案することを最終目標としている．

5　まとめ

本稿では，筆者が主に取り組んでいる交通計画分野における高速処理演算の研究事例，研究需要について紹介した．筆者が両分野の研究理解を今後も深めて橋渡し的役割をすることによって，より学際的な取り組みが交通計画分野でも生まれていくことを願っている．

参考文献

1．山﨑，宇野，倉内，塩見，嶋本(2011)：経時的交通行動把握のための ETC データ活用法とその課題，土木計画学研究・講演集，No.43.
2．山﨑，岸本，牛島(2012)：並列シミュレーションのための交通流体表現に関する研究，日本流体力学会年会 2012.
3．Jaume Balcero (eds.) (2010): Fundamentals of Traffic Simulation, Springer.
4．M. Bando et al. (1995): Dynamical model of traffic congestion and numerical simulation, Phys. Rev. E 51, p.p.1035-1042.
5．石川，中島，中島，朴 (2009)：T2K オープンスパコンが創る新しい計算機環境，計算工学，Vol. 14.
6．Bell, M.G.H. and Iida, Y. (eds.): “Transportation Network Analysis”, Wiley, 1997.

謝辞

上記 3 つの研究事例は，NEXCO 西日本株式会社に ETC データに関してご協力をいただき，研究へと活用させていただきました．また，研究を進めるにあたり多数のご助言をいただいた関係の先生方・コンサルタント会社の諸氏にも，ここに記して謝意を表します．

システム A 運転状況 （2012 年 10 月 ～ 2013 年 3 月 ）

1）保守作業に伴うサービス休止およびシステムダウン障害発生状況

保守作業に伴うサービス休止

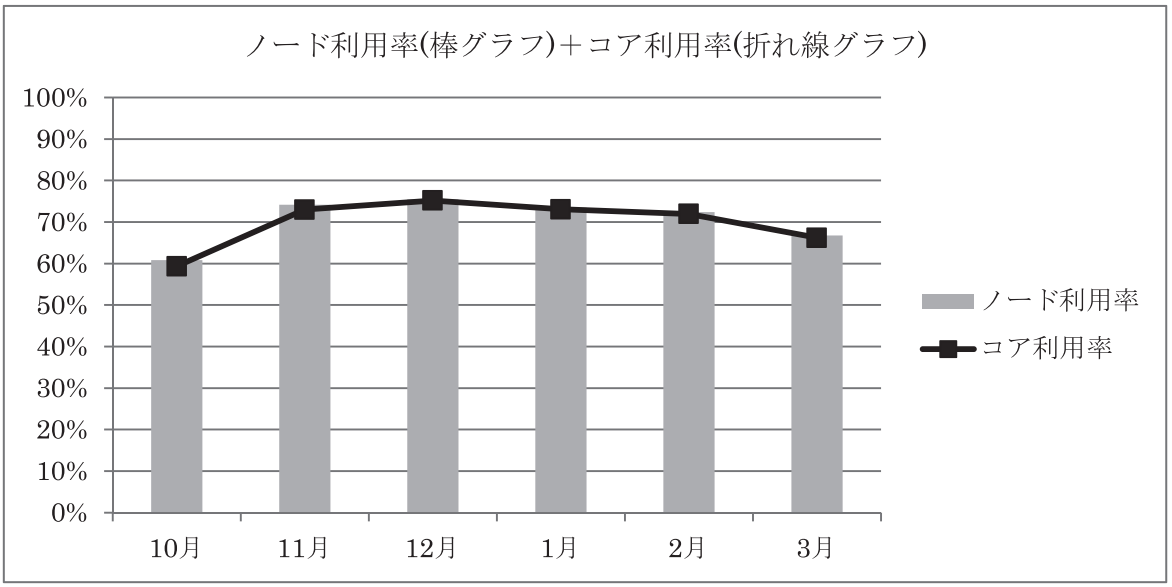
保守開始日時		サービス再開日時		保守時間[h]
2012/10/08	6:30	2012/10/15	9:00	170.50
2012/12/10	9:00	2012/12/11	9:00	24.00
2013/03/02	6:30	2013/03/03	20:35	38.08
2013/03/29	9:00	2013/04/01	0:00	63.00

システムダウン障害発生状況

障害発生日時		サービス再開日時		ダウン時間[h]
2012/10/15	9:00	2012/10/15	9:40	0.67
2012/11/06	10:25	2012/11/06	15:15	4.83
2012/11/14	11:20	2012/11/14	15:00	3.67
2012/11/21	11:00	2012/11/21	14:35	3.58
2013/02/12	23:15	2013/02/13	2:45	3.50

2）サービス状況

	サービス時間[h]	バッチ					
		処理件数	経過時間[h]	占有時間[h]	CPU時間[h]	平均稼動ノード数	ノード利用率
10 月	572.83	11,427	69,824	11,739,600	8,988,630	916.6	61 %
11 月	707.92	23,946	105,080	15,613,700	13,617,600	933.4	74 %
12 月	720.00	28,104	137,130	16,699,600	12,877,300	937.4	75 %
1 月	744.00	23,980	135,683	16,442,600	12,870,200	938.1	73 %
2 月	668.50	11,952	100,261	15,022,900	11,880,200	938.8	72 %
3 月	642.92	13,583	90,872	13,624,000	11,073,100	939.9	67 %
計	4056.17	11,427	69,824	11,739,600	8,988,630	916.6	61 %



- 占有時間 = 合計(経過時間×占有コア数)
- 平均稼動ノード数 = 電源 ON 状態のノード数の月平均 (10 分間隔のサンプリングデータより算出)
- ノード利用率 = 稼動ノードに対するジョブが実行されているノードの割合

システム B 運転状況 （2012 年 10 月 ～ 2013 年 3 月 ）

1）保守作業に伴うサービス休止およびシステムダウン障害発生状況

保守作業に伴うサービス休止

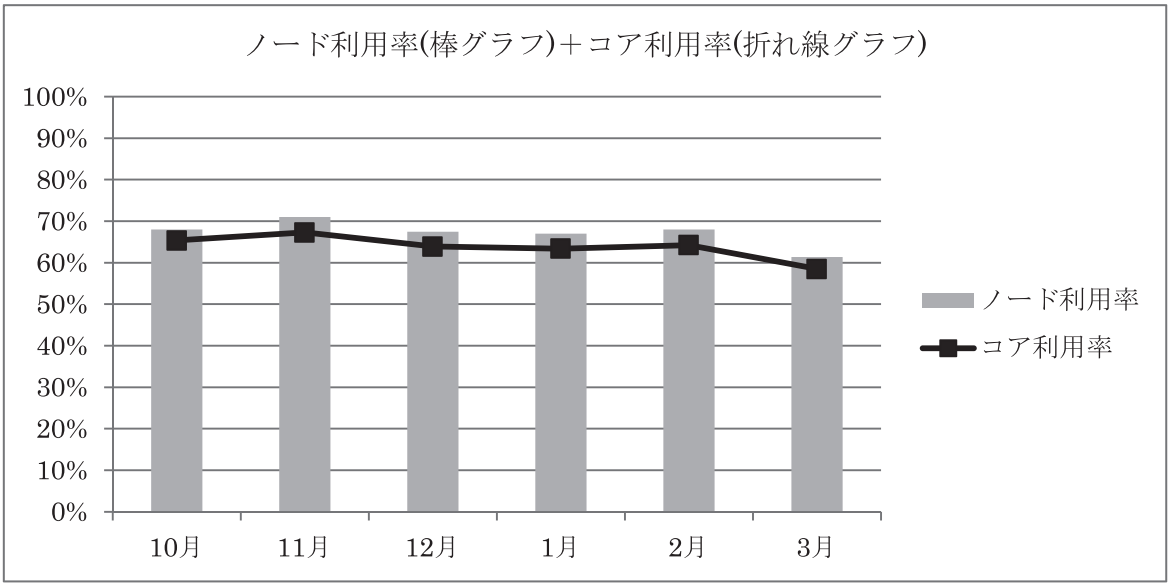
保守開始日時		サービス再開日時		保守時間[h]
2012/10/08	6:30	2012/10/09	17:00	34.50
2012/12/10	9:00	2012/12/11	0:00	15.00
2013/03/02	6:30	2013/03/03	20:35	38.08
2013/03/29	9:00	2013/04/01	0:00	63.00

システムダウン障害発生状況

障害発生日時		サービス再開日時		ダウン時間[h]
2012/10/15	9:00	2012/10/15	9:40	0.67
2012/11/06	10:25	2012/11/06	15:15	4.83
2012/11/21	11:00	2012/11/21	14:35	3.58

2）サービス状況

	サービス時間[h]	バッチ					
		処理件数	経過時間[h]	占有時間[h]	CPU時間[h]	平均稼動ノード数	ノード利用率
10 月	708.83	84,917	247,733	4,374,940	365,981	556.7	68 %
11 月	711.58	83,009	300,421	4,283,130	313,080	553.5	71 %
12 月	729.00	95,400	230,117	4,241,080	359,753	552.2	67 %
1 月	744.00	99,778	279,317	4,384,010	333,189	556.9	67 %
2 月	672.00	61,447	249,228	3,883,030	327,222	557.0	68 %
3 月	642.92	26,219	154,588	3,513,580	295,850	534.5	61 %
計	4208.33	450,770	1,461,404	24,679,770	1,995,075	551.8	67 %



- 占有時間 = 合計(経過時間×占有コア数)
- 平均稼動ノード数 = 電源 ON 状態のノード数の月平均 (10 分間隔のサンプリングデータより算出)
- ノード利用率 = 稼動ノードに対するジョブが実行されているノードの割合

システム C 運転状況 （2012 年 10 月 ～ 2013 年 3 月 ）

1）保守作業に伴うサービス休止およびシステムダウン障害発生状況

保守作業に伴うサービス休止

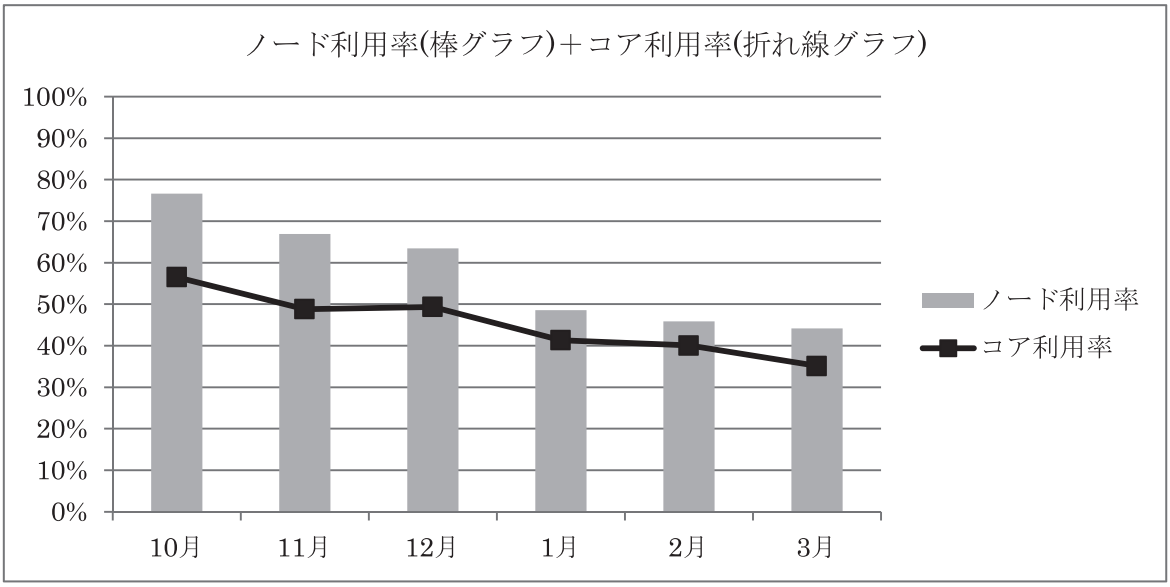
保守開始日時		サービス再開日時		保守時間[h]
2012/10/08	6:30	2012/10/09	17:00	34.50
2012/12/10	9:00	2012/12/11	0:00	15.00
2013/03/02	6:30	2013/03/03	20:35	38.08
2013/03/29	9:00	2013/04/01	0:00	63.00

システムダウン障害発生状況

障害発生日時		サービス再開日時		ダウン時間[h]
2012/10/15	9:00	2012/10/15	9:40	0.67
2012/11/06	10:25	2012/11/06	15:15	4.83
2012/11/21	11:00	2012/11/21	14:35	3.58

2）サービス状況

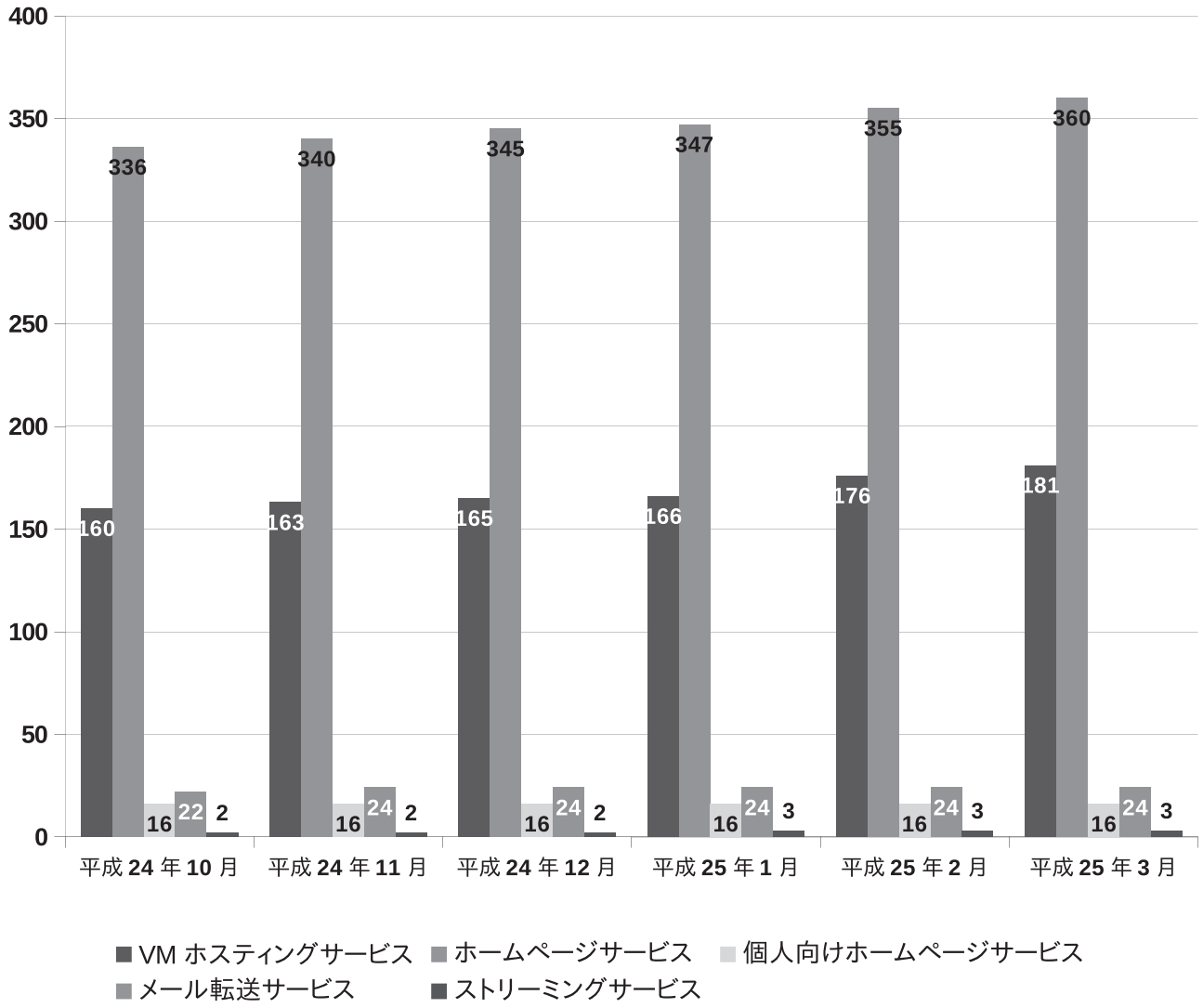
	サービス時間[h]	バッチ					
		処理件数	経過時間[h]	占有時間[h]	CPU時間[h]	平均稼動ノード数	ノード利用率
10 月	708.83	1,507	11,148	172,959	128,478	15.5	77 %
11 月	711.58	1,258	12,312	211,921	106,050	15.9	67 %
12 月	729.00	1,045	7,854	164,530	96,087	15.8	63 %
1 月	744.00	1,076	5,802	181,411	70,948	16.0	49 %
2 月	672.00	955	3,635	140,720	33,991	16.0	46 %
3 月	642.92	406	3,737	121,443	41,189	15.4	44 %
計	4208.33	6,247	44,486	992,984	476,743	15.8	58 %



- 占有時間 = 合計(経過時間×占有コア数)
- 平均稼動ノード数 = 電源 ON 状態のノード数の月平均 (10 分間隔のサンプリングデータより算出)
- ノード利用率 = 稼動ノードに対するジョブが実行されているノードの割合

汎用コンピュータシステムのサービス状況

1　ホスティング・ホームページサービス利用状況



（平成　24 年 10 月から平成　25 年 3 月）

大型計算機システム利用承認件数について

平成 25 年 3 月末現在、大型計算機システムの利用件数は、　2,630　件となっています。

大型計算機システム利用負担金

別表1　スーパーコンピュータシステム

コース	タイプ	セット	利用負担額	提供サービス					
				システム	バッチ	システム資源	経過時間（時間）	ディスク（GB）	利用者番号
エントリ	−	基本	12,600 円/年	B	共有	最大1ノード相当((16コア、64GBメモリ)×1)	1	60	−
パーソナル	タイプA	基本	100,000 円/年	A	共有	最大4ノード相当((32コア、64GBメモリ)×4)	168	1,000	−
	タイプB	基本	100,000 円/年	B	共有	最大4ノード相当((16コア、64GBメモリ)×4)	168	1,000	−
	タイプC	基本	100,000 円/年	C	共有	最大2ソケット相当((8コア、384GBメモリ)×2	168	1,000	−
グループ	タイプA1	最小	200,000 円/年	A	優先	4ノード((32コア、64GBメモリ)×4)	336	8,000	8
		追加単位	200,000 円/年			4ノード((32コア、64GBメモリ)×4)	−	8,000	8
	タイプA2	最小	240,000 円/年		準優先	8ノード((32コア、64GBメモリ)×8)	336	9,600	16
		追加単位	120,000 円/年			4ノード((32コア、64GBメモリ)×4)	−	4,800	8
	タイプA3	最小	600,000 円/年		占有	8ノード((32コア、64GBメモリ)×8)	336	16,000	16
		追加単位	300,000 円/年			4ノード((32コア、64GBメモリ)×4)	−	8,000	8
	タイプB1	最小	250,000 円/年	B	優先	4ノード((16コア、64GBメモリ)×4)	336	8,000	8
		追加単位	250,000 円/年			4ノード((16コア、64GBメモリ)×4)	−	8,000	8
	タイプB2	最小	300,000 円/年		準優先	8ノード((16コア、64GBメモリ)×8)	336	9,600	16
		追加単位	150,000 円/年			4ノード((16コア、64GBメモリ)×4)	−	4,800	8
	タイプB3	最小	750,000 円/年		占有	8ノード((16コア、64GBメモリ)×8)	336	16,000	16
		追加単位	375,000 円/年			4ノード((16コア、64GBメモリ)×4)	−	8,000	8
	タイプC1	最小	400,000 円/年	C	優先	4ソケット((8コア、384GBメモリ)×4)	336	8,000	16
		追加単位	200,000 円/年			2ソケット((8コア、384GBメモリ)×2)	−	4,000	8
	タイプC2	最小	240,000 円/年		準優先	4ソケット((8コア、384GBメモリ)×4)	336	4,800	16
		追加単位	120,000 円/年			2ソケット((8コア、384GBメモリ)×2)	−	2,400	8
	タイプG1	最小	250,000 円/年	B（GPU）	優先	2ノード((16コア、64GBメモリ + 1GPU)×2)	336	4,000	8
		追加単位	250,000 円/年			2ノード((16コア、64GBメモリ + 1GPU)×2)	−	4,000	8
大規模ジョブ	タイプA	最小	20,000 円/週(7日)	A	占有	8ノード((32コア、64GBメモリ)×8)	−	−	−
		追加単位	5,000 円/週(7日)			2ノード((32コア、64GBメモリ)×2)	−	−	−
	タイプB	最小	24,000 円/週(7日)	B	占有	8ノード((16コア、64GBメモリ)×8)	−	−	−
		追加単位	6,000 円/週(7日)			2ノード((16コア、64GBメモリ)×2)	−	−	−
	タイプC	最小	20,000 円/週(7日)	C	占有	4ソケット((8コア、384GBメモリ)×4)	−	−	−
		追加単位	10,000 円/週(7日)			2ソケット((8コア、384GBメモリ)×2)	−	−	−
専用クラスタ	—	最小	750,000 円/年	B	−	8ノード((16コア、64GBメモリ)×8)	−	16,000	16
		追加単位	375,000 円/年			4ノード((16コア、64GBメモリ)×4)	−	8,000	8
ライセンスサービス			20,000 円/年	可視化ソフト（AVS,ENVI/IDL)およびプリポストウェアの1ライセンスにつき					

備考

1. 利用負担額は、年度単位で算定している。また、総額表示である。
2. 大型計算機システムの全ての利用者は、上記表のサービスの他、次のサービスを受けることができる。
   1） 大判プリンタサービス
   2） その他、大型計算機システムが提供するサービス、機器の利用
3. 上記表の大規模ジョブコース、ライセンスサービスの申請には、大型計算機システムの利用者であることが必要である。
4. 「共有」: 当該カテゴリのユーザ間で一定の計算資源を共有するベストエフォートのスケジューリングを行う。
   「準優先」: 定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
   また、稼働状況によらず記載値の1/4の計算資源が確保されることを保証する。
   「優先」: 定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
   また、稼働状況によらず記載値の1/2の計算資源が確保されることを保証する。
   「占有」: 稼働状況によらず記載値（以上）の計算資源が確保されることを保証する。
5. ディスク容量はバックアップ領域（最大で総容量の1/2）を含む。
6. グループコース及び専用クラスタコースのシステム資源は、下記の負担額を支払うことにより増量することができる。
   なお増量は各月1日に実施し、増量した資源は当該年度末までの期間にわたって利用されるものとする。

コース	タイプ	追加負担金額（増量単位あたり）	システム資源増量単位	ディスク増量（GB）
グループ	タイプA1	20,000 円/月	4ノード((32コア、64GBメモリ)×4)	8,000
	タイプA2	12,000 円/月	4ノード((32コア、64GBメモリ)×4)	4,800
	タイプA3	30,000 円/月	4ノード((32コア、64GBメモリ)×4)	8,000
	タイプB1	25,000 円/月	4ノード((16コア、64GBメモリ)×4)	8,000
	タイプB2	15,000 円/月	4ノード((16コア、64GBメモリ)×4)	4,800
	タイプB3	37,500 円/月	4ノード((16コア、64GBメモリ)×4)	8,000
	タイプC1	20,000 円/月	2ソケット((8コア、384GBメモリ)×2)	4,000
	タイプC2	12,000 円/月	2ソケット((8コア、384GBメモリ)×2)	2,400
	タイプG1	25,000 円/月	2ノード((16コア、64GBメモリ + 1GPU)×2)	4,000
専用クラスタ	—	37,500 円/月	4ノード((16コア、64GBメモリ)×4)	8,000

7. グループコース及び専用クラスタコースを通年でなく利用する場合には、下記の負担額を支払うものとする。
   ただし、利用期間は当該年度内に限るものとする。

利用期間			3ヶ月	6ヶ月	9ヶ月
グループコース	タイプA1	最小	80,000 円	120,000 円	180,000 円
		追加単位	80,000 円	120,000 円	180,000 円
	タイプA2	最小	96,000 円	144,000 円	216,000 円
		追加単位	48,000 円	72,000 円	108,000 円
	タイプA3	最小	240,000 円	360,000 円	540,000 円
		追加単位	120,000 円	180,000 円	270,000 円
	タイプB1	最小	100,000 円	150,000 円	225,000 円
		追加単位	100,000 円	150,000 円	225,000 円
	タイプB2	最小	120,000 円	180,000 円	270,000 円
		追加単位	60,000 円	90,000 円	135,000 円
	タイプB3	最小	300,000 円	450,000 円	675,000 円
		追加単位	150,000 円	225,000 円	337,500 円
	タイプC1	最小	160,000 円	240,000 円	360,000 円
		追加単位	80,000 円	120,000 円	180,000 円
	タイプC2	最小	96,000 円	144,000 円	216,000 円
		追加単位	48,000 円	72,000 円	108,000 円
	タイプG1	最小	100,000 円	150,000 円	225,000 円
		追加単位	100,000 円	150,000 円	225,000 円
専用クラスタ コース	—	最小	300,000 円	450,000 円	675,000 円
		追加単位	150,000 円	225,000 円	337,500 円

8. グループコース及び専用クラスタコースの利用者番号は利用者あたり年額5,000円を負担することで追加できる。

9. 機関・部局定額制度
他機関又は学内における部局(『国立大学法人京都大学の組織に関する規程』第3章第2節から第11節で定める組織をいう。)の組織が、その組織単位でグループコースサービス(年間)の利用を申請する場合、料金表(年間)に掲載額の1.5倍を利用負担金とする。なお、利用負担金額が150万円未満の場合は100人、150万円を超える場合は、150万円毎に100人までの利用者を認める。

別表2(汎用コンピュータシステム)

区分	利用負担額	単位
VMホスティングサービス	72,000円／年	1仮想マシンにつき
ホームページサービス	6,000円／年	1ドメイン名につき
ストリーミングサービス	6,000円／年	1申請につき

備考
1. 利用負担額は、総額表示である。
2. 上記表の汎用コンピュータシステムのサービスを利用するためには、大型計算機システムの利用者であることが必要である。
3. VM ホスティングサービスにおいて、下記の負担額を支払うことにより CPU、メモリ、ディスクを増量することができる。

区分	利用負担額	単位
CPU増量	18,000円／年	2コアにつき（最大8コアまで）
メモリ増量	18,000円／年	8GBにつき（最大64GBまで）
ディスク増量	18,000円／年	200GBにつき（最大1,000GBまで）

4. VM ホスティングサービスにおいて VMware を用いる場合は、下記の負担額を支払うことにより VMware の利用及び CPU、メモリ、ディスクを増量することができる。ただし、システム資源が限られているためサービスの提供を限定することがある。

区分	利用負担額	単位
VMware利用	72,000円／年	1仮想マシンにつき
CPU増量	36,000円／年	2コアにつき（最大8コアまで）
メモリ増量	36,000円／年	8GBにつき（最大64GBまで）
ディスク増量	18,000円／年	200GBにつき（最大1,000GBまで）

5. ホームページサービス及びストリーミングサービスにおいて、下記の負担額を支払うことにより公開スペースの上限を拡大することができる。

区分	利用負担額
公開スペース20GBプラン	3,000円／年
公開スペース50GBプラン	9,000円／年

6. 利用負担額は、当該年度（4月から翌年3月まで）の利用に対して年額として算定するが、年度途中から利用を開始する場合には月数に応じて減額する。

別表3 スーパーコンピュータシステム(民間機関利用)

システム	システム資源	経過時間(時間)	ディスク(GB)	利用者番号	利用負担額
A	8ノード(32コア、64GBメモリ)×8)	336	9,600	16	960,000 円/年
	12ノード(32コア、64GBメモリ)×12)	336	14,400	24	1,440,000 円/年
	16ノード(32コア、64GBメモリ)×16)	336	19,200	32	1,920,000 円/年
B	8ノード(16コア、64GBメモリ)×8)	336	9,600	16	1,200,000 円/年
	12ノード(16コア、64GBメモリ)×12)	336	14,400	24	1,800,000 円/年
	16ノード(16コア、64GBメモリ)×16)	336	19,200	32	2,400,000 円/年

備考
1. 利用負担額は、年度単位で算定している。また、総額表示である。
2. ディスク容量はバックアップ領域(最大で総容量の1/2)を含む。
3. 通年でなく利用する場合には、下記の負担額を支払うものとする。
ただし、利用期間は当該年度内に限るものとする。

システム	システム資源	利用期間		
		3ヶ月	6ヶ月	9ヶ月
A	8ノード	240,000 円	480,000 円	720,000 円
	12ノード	360,000 円	720,000 円	1,080,000 円
	16ノード	480,000 円	960,000 円	1,440,000 円
B	8ノード	300,000 円	600,000 円	900,000 円
	12ノード	450,000 円	900,000 円	1,350,000 円
	16ノード	600,000 円	1,200,000 円	1,800,000 円

全国共同利用版広報・Vol.11(2012)総目次

[巻頭言]

Vol.11, No.1 号の発刊にあたって 1-1
機構長再任にあたって 2-1

[特集「新スーパーコンピュータ運用開始」]

新スーパーコンピュータシステム　－そのコンセプトと構成－ 1-2
スーパーコンピュータ利用ガイド　－新システムを利用するために－ 1-7
新スーパーコンピュータにおけるベンチマークテスト報告 1-19

[スーパーコンピュータ共同研究制度（若手研究者奨励枠）研究報告]

分子軌道法計算による光学活性高分子の構造解析 2-2
色素増感太陽電池を指向したポルフィリン化合物の開発 2-5
生体軟部組織に対する非線形有限要素解析
　－残留応力，血管平滑筋の能動的応力を考慮した血管壁の応力解析－ 2-7
有機材料における電荷輸送特性解析 2-9
蔵本-シバシンスキー方程式における不安定周期軌道の統計性質 2-11
fMRI を用いたヒト頭頂間溝視覚野の集団受容野推定 2-14
フラグメント分子軌道法を用いた光合成反応中心の電子移動経路解析 2-16

[プログラム高度化支援事業研究報告]

H-matrices（階層型行列）法を用いた準動的地震発生サイクルシミュレーションの
　省メモリ化・高速化(2) 2-18

[スーパーコンピュータ共同研究制度（大規模計算支援枠）研究報告]

ジャイロ運動論に基づいた位相空間 5 次元ブラソフ方程式による乱流輸送の
　シミュレーション研究 2-24

[サービスの記録・報告]

スーパーコンピュータシステムの稼働状況とサービスの利用状況 1-23, 2-28
センター利用による研究成果（平成 23 年度） 2-32
2012 年度京都大学学術情報メディアセンターコンテンツ作成共同研究　募集・採択結果 2-34

[資料]

大型計算機システム利用負担金　別表 1-26, 2-36
全国共同利用版広報・Vol.10(2011)総目次 1-28
サービス利用のための資料一覧 1-30, 2-38

[編集後記]

編集後記、奥付 1-.31, 2-39

― サービス利用のための資料一覧 ―

1. スーパーコンピュータシステム・ホスト一覧

- システム A：camphor.kudpc.kyoto-u.ac.jp
- システム B・C：laurel. kudpc.kyoto-u.ac.jp
  - システム B（SAS 利用時）：sas.kudpc.kyoto-u.ac.jp

※ ホストへの接続は SSH(Secure SHell) 鍵認証のみ、パスワード認証は不可

2. 問い合わせ先 & リンク集

- 情報環境機構のホームページ
  http://www.iimc.kyoto-u.ac.jp/

- 学術情報メディアセンターのホームページ
  http://www.media.kyoto-u.ac.jp/

- スーパーコンピュータシステムに関する問い合わせ先
  - 利用申請などに関する問合わせ先

  【共同利用掛】
  E-mail：zenkoku-kyo@media.kyoto-u.ac.jp / Tel：075-753-7424
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/comp/
  - システムの利用など技術的な問い合わせ

  【研究支援グループ】
  E-mail：consult@kudpc.kyoto-u.ac.jp / Tel：075-753-7426
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/comp/contact.html

- ホームページ・ホスティングサービス関する問合わせ先
  【情報環境支援グループ】
  E-mail：whs-qa@media.kyoto-u.ac.jp / Tel：075-753-7494
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/whs/

- コンテンツ作成支援サービスに関する問合わせ先
  【コンテンツ作成室】
  E-mail：cpt@media.kyoto-u.ac.jp / Tel：075-753-9012
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/content/

編 集 後 記

先日、油を使用せず熱と空気で揚げ物を作るという話題の調理家電を購入した。使用方法は、揚げる前の状態の食品（冷凍でもよい）をトレイに並べ、温度を選択しタイマーセットするだけ。調理中にキッチンを離れて他の作業が出来るのと、油の後始末が不要なのは感動的だ。おかげで忙しい平日の夕食メニューにも揚げ物が加わるようになった。さらに、油を使用しないので、カロリーも控えめと良いことばかり。肝心の味はというと、フライドポテトや揚げ出し豆腐は中々の出来栄えだが、鶏の唐揚げは、油の少ない胸肉を使用したところパサパサになってしまった。また、トンカツは、味は悪くないものの、こんがりきつね色にはならず見た目がイマイチだ。とりあえず、我が家では、娘の大好物のフライドポテトを作る機械として活躍することになりそうだ。

手抜き料理人

最近、つくづく「お金は大事だなあ」と感じるようになりました。生活の拠点を作るのに必要。食事にも必要。どこかへ移動するのにも必要。趣味を楽しむにも必要。体の具合が悪くなったら、通院したり薬を買うためにも必要。税金を払い、社会人としての義務を果たすのにも必要。そこまでの贅沢を望まなくても、生きていくためにある程度のお金は必要不可欠、そんな当たり前のことを最近身にしみて感じるようになりました。ただ一方であまりお金のことばかり考え過ぎると、精神的に余裕がなくなってくるのも事実。心身ともに充実して生きていくというのも、なかなか難しいものですね・・・。

暇なし

京都大学学術情報メディアセンター全国共同利用版広報　Vol. 12, No. 1

2013年　7月　31日発行

編集者　京都大学学術情報メディアセンター
　　広報教育委員会・全国共同利用版広報編集部会
発行者　〒606-8501　京都市左京区吉田本町
　　京都大学学術情報メディアセンター
　　Academic Center for Computing and Media Studies
　　Kyoto University
　　Tel. 075-753-7400
　　http://www.media.kyoto-u.ac.jp/
印刷所　〒616-8102 京都市右京区太秦森ヶ東町21-10
　　株式会社エヌジーピー

目次

[巻頭言]

·Vol.12, No.1 号の発刊にあたって 1
岩下武史

[新汎用コンピュータシステム運用開始]

·新しい汎用コンピュータシステムの概要ー大学の情報基盤を支えるクラウド型システムにむけてー 2
河原達也、森信介、赤坂浩一

·ホスティング·ホームページサービスの改定についてー改定のポイントとサービスの利用方法ー 7
赤尾健介、赤坂浩一、針木剛、秋田祐哉、森信介、河原達也

[解説]

·CrayPat による性能解析 13
武田大輔

·GPUを用いたFEMアプリケーションの高速化 20
大島聡史

[研究紹介]

·交通計画分野の高速処理演算需要 30
山﨑浩気

[サービスの記録·報告]

·スーパーコンピュータシステムの稼働状況とサービスの利用状況 38

[資料]

·大型計算機システム利用負担金 別表 42

·全国共同利用版広報·Vol.11(2012)総目次 44

·サービス利用のための資料一覧 45

[編集後記]

·編集後記、奥付 46

Cover : Designed by TANI Takushi / T&T Design Lab. (July,2013)