ISSN 0286-7419

TOHOKU
UNIVERSITY

東北大学
サイバーサイエンスセンター

大規模科学計算システム広報

SENAC

Vol.48 No.3 2015–7

Cyberscience
Center

Supercomputing System
Tohoku University
www.ss.cc.tohoku.ac.jp

大規模科学計算システム関連案内

＜大規模科学計算システム関連業務は、サイバーサイエンスセンター本館内の情報部情報基盤課が担当しています。＞

http://www.ss.cc.tohoku.ac.jp/

階	係・室名	電話番号(内線)* e-mail	主なサービス内容	サービス時間 平 日
一階	利用相談室	022-795-6153(6153) sodan05@cc.tohoku.ac.jp 相談員不在時 022-795-3406(3406)	計算機利用全般に関する相談 大判プリンタ、利用者端末等の利用	8:30～17:15 8:30～21:00
	利用者談話室	(3444)	各センター広報の閲覧	8:30～21:00
	展 示 室 (分散コンピュータ博物館)		歴代の大型計算機等の展示	9:00～17:00
	可視化機器室	(3428)	三次元可視化システムの利用	9:00～21:00
三階	総務係	022-795-3407(3407) som@cc.tohoku.ac.jp	総務に関すること	8:30～17:15
	会計係	022-795-3405(3405) kaikei@cc.tohoku.ac.jp	会計に関すること、負担金の請求に関すること	8:30～17:15
	共同研究支援係	022-795-6252(6252) rs-sec@cc.tohoku.ac.jp	共同研究、計算機システムに関すること	8:30～17:15
	共同利用支援係 (受 付)	022-795-3406(3406) 022-795-6251(6251) uketuke@cc.tohoku.ac.jp	利用手続き、利用相談、講習会、ライブラリ、アプリケーションに関すること	8:30～17:15
	ネットワーク係	022-795-6253(6253) net-sec@cc.tohoku.ac.jp	ネットワークに関すること	8:30～17:15
四階	研究開発部	022-795-6095(6095)		
五階	端末機室	(3445)	PC 端末機(X 端末)	

＊(　) 内は東北大学内のみの内線電話番号です。青葉山・川内地区以外からは頭に 92 を加えます。

本誌の名前「SENAC」の由来

昭和 33 年に東北地区の最初の電子計算機として、東北大学電気通信研究所において完成されたパラメトロン式計算機の名前で SENAC-1(SENdai Automatic Computer-1)からとって命名された。

［共同研究成果］

X 線自由電子レーザーパルスによるフラーレン超多価カチオン C_{60}^{q+}の爆発解離の動力学シミュレーション

— 2 段階爆発機構の提案 —

山崎 馨[1,2]・上田 潔[3]・河野 裕彦[1]
[1] 東北大学大学院理学研究科化学専攻,
[2] 北海道大学大学院理学研究科化学部門,
[3] 東北大学多元物質科学研究所

X 線自由電子レーザー照射によって生成する超多価カチオン C_{60}^{q+} (q = 20–60) の解離機構を Self-consistent charge density-functional based tight-binding (SCC-DFTB) 法に基づく分子動力学計算を用いて研究した．その結果，電荷 q によらず(1) 正電荷間のクーロン反発によって多価原子カチオンが 10–20 fs 程度で非統計的に放出されるクーロン爆発と(2) 1 価・中性分子フラグメントの統計的脱離(100 fs から 1 ps 程度)の 2 段階機構で解離することを明らかにした．

1. 序論：X 線自由電子レーザーとクーロン爆発

我々が利用する医薬品は，有効成分の分子がタンパク質などの生体分子の特定の部位に作用して効果を示す物が多い．このため，タンパク質の構造を決めることは病気の発症機構を解明し有効な医薬品を開発するための第 1 歩である．我々が目に見えない小さなものを観測する際に使用する顕微鏡は使用する光の波長の半分程度の大きさの構造を見分けることができる．例えば，中学・高校で使用する普通の光学顕微鏡は，波長数百 nm の可視光を使用するため，μm 程度の大きさの細胞の内部構造を観測することができるが，細胞の中にある数十から数百 nm 程度のタンパク質の構造を観測することはできない．このため，ナノ分子の構造や化学結合の組み替えを観測できる新たな顕微鏡の開発が求められていた．この様な目的のために作られた X 線レーザーパルス顕微鏡が日本の兵庫県にある SACLA や米国カリフォルニア州にある LCLS などの X 線自由電子レーザー(XFEL)施設である．XFEL 施設では化学結合の長さと同程度(0.1 nm)かそれ以下の波長を持つ X 線を用いて今まで測定が困難であったタンパク質などのナノ分子 1 分子の構造や化学反応の様子を直接観察することを究極的な目標の一つとしている[1]．しかし，このような測定を可能にする高強度の X 線レーザーパルス(XFEL)[1]では，タンパク質分子が容易にイオン化され，分子内の強いクーロン反発によって爆発してしまうクーロン爆発という現象がおきる[1-3]．我々の体の中で有用な働きをしているタンパク質は，構造や反応の仕組みを決めるために質・量共に十分な試料を用意することが難しいものも多く，XFEL パルスによる試料分子のクーロン爆発は可能な限り回避しなければならない．よって，この爆発の影響を最小限に抑えるために，ナノ分子が X 線レーザーパルスによってどの程度の時間でどの様に爆発していくかを明らかにすることが喫緊の課題となっている．そこで筆者らはナノ分子としてフラーレン C_{60} を取り上げ，XFEL パルスの照射によって数十個の電子がはぎ取られてできた多価のフラーレン正イオン C_{60}^{q+}[2] [2]がどの様に壊れていくかを，量子化学計算と分子動力学シミュレーションを組み合わせて調べた[4].

---

[1] 典型的な反応動力学イメージング実験の場合，パルスの半値幅は 10 fs 前後，パルス 1 つに含まれる光子の数は 1 μm^2 あたり 10^{11} 個程度である[3].

[2] 実験的には 100 個ぐらいまでの電子がはぎ取られる可能性が示唆されている．詳しくは[2]を参照.

2. 計算手法

2.1 内殻イオン化とオージェイオン化の垂直イオン化モデル

XFEL パルスが C_{60} に照射されると，図 1 に示すように，強いクーロン引力で原子核近くに束縛された内殻 1s 軌道から電子が放出され，1s 軌道に非常に不安定な空孔ができる(内殻イオン化)．この空孔を埋めるために，より外側にゆるく束縛された 2p 軌道などの価電子軌道の電子が 1*s* 軌道へ移ると同時に価電子軌道の電子がもう 1 電子イオン化されるオージェイオン化がおこる．XFEL パルスは非常に多くの光子を含んでいるために，この内殻イオン化・オージェイオン化は連続して複数回起き，C_{60}^{q+}が生成する[2]．1 回の内殻イオン化とオージェイオン化で計 2 個の電子が分子から放出されるので，q 価の多価カチオンを作るためには，$q/2$ 回の内殻イオン化・オージェイオン化過程が必要である．このイオン化過程では，電子と原子核との相互作用によって，1 回当り 10 eV，$q/2$ 回で $5q$ eV の余剰振動エネルギーE_{in}が原子核の自由度に注入される[5]．本研究では，C_{60}^{q+}の解離機構が電荷 q に応じてどの様に変化するかを調べるため，まずこのイオン化過程を大幅に簡略化した垂直イオン化モデルを採用した．すなわち，C_{60} がサッカーボール状の平衡構造を保ったままシミュレーション開始時に瞬間的に q 価までイオン化され，それと同時に $E_{in} = 5q$ eV が注入されると仮定した．

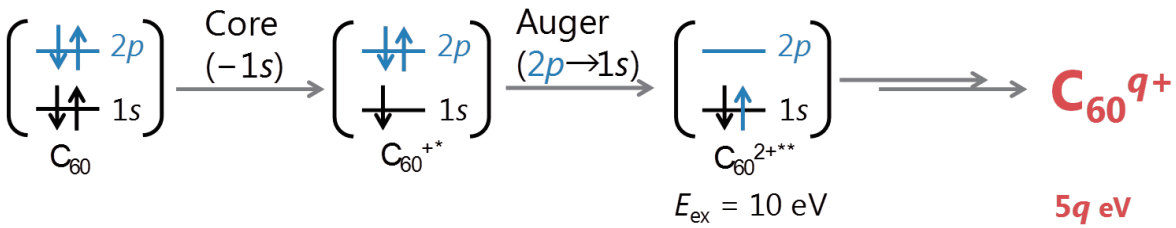


図 1：X 線自由電子レーザーによる C_{60} の多重イオン化過程．1*s* 軌道の内殻イオン化とそれに続いて起きるオージェ緩和が $q/2$ 回繰り返されて C_{60}^{q+}が生成する．

2.2 Self-consistent charge density-functional based tight binding (SCC-DFTB)法

本研究では，半経験的な分子軌道法の一つである Self-consistent charge density-functional based tight binding (SCC-DFTB)法[6,7]を用い，C_{60}^{q+}に対して全エネルギー一定の条件で分子動力学シミュレーションを行った．この動力学計算には量子化学計算パッケージ DFTB+ 1.2.2[8]を用いた．SCC-DFTB 法は密度汎関数(DFT)法を使って決められたパラメータを用いると共に分子の構造変化による電荷分布の揺らぎを考慮に入れることによって，DFT 法に匹敵する計算精度の計算を 1/100 以下の計算コストで可能にしている．例えば，XFEL の光エネルギーが C_{60}^{60+}を生成するために全て使われた場合に起きると考えられる等方的な爆発過程では，生成する 60 個の C^+カチオンの平均運動エネルギーは SCC-DFTB 法では 94 eV と計算され，代表的な DFT の汎関数である B3LYP を用いた場合(90 eV)とよく一致する[3]．この際の計算時間は 16 コアをノード内並列で用いた場合，B3LYP 法で 500 fs あたり約 1 ヶ月，SCC-DFTB 法では 1 時間以下(設定にもよるが，15-30

[3] 解離極限では，C^+の電子配置は 2p 軌道に 1 個不対電子を持つラジカルになると考えられる．この様な開殻電子配置をより適切に記述するために，α スピンと β スピンの電子が入る分子軌道の形状を別々に最適化する非制限 DFT 法および spin-polarized DFTB 法[7]を用いた．また，多価カチオンの最高占有軌道と最低非占有軌道間のエネルギー差は非常に小さくなる．このため，占有軌道の電子が非占有軌道に遷移している励起電子配置が無視できなくなり，エネルギー計算の収束性が非常に悪くなる．この様な状況を改善するために，SCC-DFTB 法における計算では電子を温度 5000 K(~0.43 eV)で被占有軌道にも Fermi-Dirac 分布させることによって（励起電子配置の効果を熱平均する)，収束性を改善すると共に低エネルギー励起電子配置の効果を現象論的に取り込んだ．

分程度)となった．そこで本研究では $q = 20, 28, 60$ において，$E_{in} = 5q$ eV を注入した場合の全長 10 ps のトラジェクトリをそれぞれ 31 回計算し，C_{60}^{q+}の解離機構を統計的に解析した．

2.3 フラグメントの解離判別法および形式電荷 z_n の割り当て

トラジェクトリごとに C–C 間の結合距離 r_{C-C} を逐次計算してフラグメントの解離判別を行った.今回は，$r_{c-c} \leq 3.2$ Å となる炭素原子を一つのフラグメントとしてまとめた[4]．なお，$r_{c-c} = 3.2$ Å という値は， 直線 C_n^+ ($n = 2$–5)フラグメントの最低 2 重項状態において，C–C 結合の開裂が完了して SCC-DFTB 法によるポテンシャルエネルギー曲線がほぼ平坦になる結合長である．
SCC-DFTB 法によって計算されたフラグメントのマリケン電荷は非整数の値をとる．このため，フラグメントの分岐比を計算するには この非整数電荷から整数の形式電荷 z_n へと何らかのモデルを用いて変換してやる必要がある．我々はある時間において n 個の炭素原子からなるフラグメントの形式電荷が z_n と $z_n + 1$ の二つしかとらないと仮定した．さらに，z_n と $z_n + 1$ は SCC-DFTB 法によって求められたサイズ n のフラグメントのマリケン電荷の平均値 $\langle z_n \rangle$に対して $z_n < \langle z_n \rangle < z_n + 1$ の関係を持つと仮定した．以上 2 つの条件の下で，サイズ n のフラグメントのマリケン電荷の総計と形式電荷の総計の差の絶対値が最小になるように，z_n と $z_n + 1$ の電荷を持つフラグメントの数を決定した．詳細については[4]の Supplemental Material を参照されたい．

3. C_{60}^{q+} (q = 20-60) 多価カチオンの 2 段階爆発機構[4]

本研究で， $q \geq 20$ および $E_{in} \geq 100$ eV の条件において，C_{60}^{q+} 多価カチオンの爆発的解離が 2 段階で起きることを見いだした．この 2 段階機構では，図 2 に示す様に多価の原子フラグメント C^{z+} ($z \geq 2$)が 10 fs 程度の時間スケールで放出される．この C^{z+} の放出に伴って多くの正電荷が放出され，解離せずに残っている core cluster と呼ばれる構造が静電的に安定化される．この core cluster は，内部に残っている振動(熱)エネルギーによって 100 fs から 1 ps 程度の時間スケールで 1 価の原子フラグメントや C_2^+，C_3^+などの小さな分子フラグメントに分解する．ここでは具体例として， C_{60}^{60+} に $E_{in} = 300$ (= $5q$) eV を注入した場合について詳しく見ていくことにする．

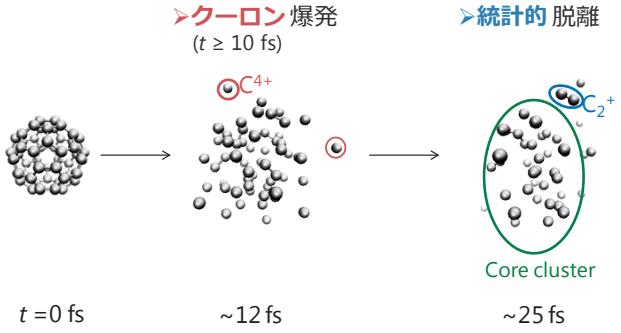


図 2 フラーレン超多価カチオン C_{60}^{q+} (q = 20–60)における 2 段階解離機構の q = 60 の例. 多価の原子カチオン C^{z+}(z = 2–4)がクーロン反発によって 10–20 fs 程度で放出された後に, 解離せずに残っている炭素クラスターから C_n^{z+}($n \geq 1, z = 1, 2$)が統計的に脱離する[5].

---

[4] このグループ化は，階層クラスター解析法の一つである Minimum linkage アルゴリズム（例えば文献[9,10]を参照）を応用して実装した．

[5] なお，この爆発過程のトラジェクトリはサイバーサイエンスセンターの協力により 3 次元動画として可視化されており，本センターにおいて閲覧することが可能である．

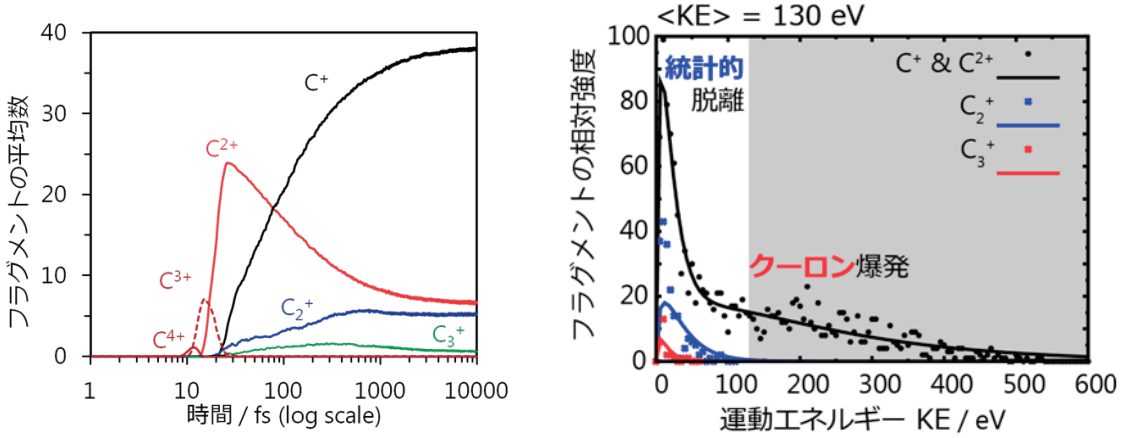


図 3 : C_{60}^{60+} に $E_{in} = 300\ (= 5q)$ eV を注入した場合の(a)フラグメントの種類と数の時間変化および(b)シミュレーション開始から 10 ps 後におけるフラグメントの並進運動エネルギー分布.

図 2 および図 3(a)に示す様に，C_{60}^{60+} の爆発過程は $t \sim 12$ fs からの C^{4+} 放出によって始まる．この原子性フラグメントの放出過程は正電荷を持つ炭素原子間の強いクーロン反発によるものである．このクーロン爆発過程に引き続いて core cluster C_n^{z+} ($n \sim 25$ and $z \sim 5$) が C^{z+}(z = 1–3) に分解する過程 $t \sim 25$ fs 付近から始まる． C–C 間のクーロン反発エネルギーを計算すると，シミュレーションを開始した $t = 0$ fs における C_{60}^{60+} では結合一本当り 69.8 eV であった．これは，典型的な C=C 2 重結合を回裂させるために必要な解離エネルギー(6.3–6.5 eV)の約 10 倍に相当する．一方クーロン爆発が終了して分子フラグメントが放出されるようになる $t = 25$ fs では，core cluster (~C_{25}^{5+})におけるクーロンエネルギーは結合 1 本当り約 0.5 eV まで減少していた．なお, 0.5 eV という値は C=C 2 重結合の解離エネルギーのおよそ 10 % にしか相当しない．その代わり core cluster の中には 24 本の C=C 2 重結合を切断できるエネルギーに相当する 152.3 eV(4.9 万℃に相当)もの振動・回転エネルギーが熱として蓄えられており，この熱エネルギーを使って分子フラグメントの生成が進行する．以上より，分子フラグメントの脱離は，core cluster 内の振動・回転エネルギーがその内部で統計的に分配されて起きる過程だと考えられる．

この 2 段階爆発機構をより詳細に解析するために，筆者らは解離フラグメントの並進運動エネルギー分布を解析した．一般に，解離する親分子(この場合は C_{60}^{60+})が高いエネルギーを持っている状態から放出されたフラグメントは高い並進運動エネルギーを持ち，系からのエネルギー放出が進んで系内のエネルギーが少なくなってから放出されるフラグメントの並進運動エネルギーは低くなる傾向がある. 我々の統計解析の結果, 各フラグメントの並進運動エネルギーの分布関数 f は式(1)にフラグメントの並進運動エネルギー K_T の関数である Maxwell-Boltzmann エネルギー分布[11] 1 つまたは 2 つでフィッティングできることが明らかになった ．

$$f(K_T) = \sum_{i=1}^{2} N_i \frac{2\sqrt{K_T}}{\sqrt{\pi}(k_B T_i)^{3/2}} \exp\left(-\frac{K_T}{k_B T_i}\right), \qquad (1)$$

ここで T_i と N_i は i 番目の運動エネルギー分布関数における実効的な並進運動温度と対応するフラグメントの相対強度である．また，k_B は Boltzmann 定数を表す. T_i と N_i の値は，シミュレーションで得られた並進運動エネルギー分布に f (K_T)を最小自乗法でフィッティングして求めた. C_{60}^{60+} に $E_{in} = 300$ eV を注入した場合におけるサイズ n (C_n^{z+}, z = 0, 1, …) のフラグメントの並進運動エネルギー分布を図 3(b)に示す. C^+ における並進運動温度の高速成分 $k_B T_1$ と低速成分 $k_B T_2$ の値はそれぞれ 150.9, 14.3 eV と求められた．C^+ の $k_B T_1$ の値は，電荷 q が小さくなるにつれて急激に減少する．これは， Xe^{25+}衝突による C_{60}^{q+}のクーロン爆発実験において，C^+ の運動エネルギー分布の高速成分の値が，q の減少と共に小さくなるという Tomita らの実験結果と一致す

る[12]．また，n = 2, 3 の分子フラグメントにおける k_BT_1 と k_BT_2 はどちらもおよそ 10 eV であり，系全体の電荷 q および余剰振動エネルギー E_{in} によらず定性的には一定であると見なせることが分かった．このことは，Tomita らの実験で，分子フラグメントの平均並進運動エネルギーが q によらず 10–20 eV とほぼ一定であったこと[12]に一致する．以上より，C_{60}^{q+} から生成したフラグメントの高速成分と低速成分はそれぞれクーロン爆発とそれに引き続いて起きる統計的な解離に起因することが明らかになった．この 2 段階爆発機構を実験的に証明するためには，並進運動エネルギーの大きな C^+ と低エネルギーの C_n^+ (n = 1–3)を同時に質量分析装置で観測するか，2 つ以上の種類のフラグメントの運動量または並進運動エネルギーの相関を解析すればよい．また，図 3(a)に示したフラグメント数の時間変化については XFEL パルスでクーロン爆発を引き起こし，紫外線パルスでフラグメント数の変化を追跡する XFEL ポンプ―紫外プローブ分光法で実験的に観測できると期待される．

4. 結論

本研究では，X 線自由電子レーザー照射によって生成する超多価カチオン C_{60}^{q+} (q = 20–60) の解離機構を Self-consistent charge density-functional based tight-binding (SCC-DFTB) 法に基づく分子動力学計算を用いて研究した．その結果，電荷 q によらず(1) 正電荷間のクーロン反発によって多価原子カチオンが 10–20 fs 程度で非統計的に放出されるクーロン爆発と(2)1 価・中性分子フラグメントの統計的脱離(100 fs から 1 ps 程度)の 2 段階機構で解離することを明らかにした．本機構は幅広い電荷において成り立つため，タンパク質などの他のナノ・バイオ分子に対しても一般的に成り立つことが期待される．このため，この 2 段階爆発機構の適応範囲を理論と実験の緊密なインタープレーによって明らかにすることで，ナノ・バイオ分子の XFEL による反応ダイナミクスイメージングの実用化に繋がることが期待される．

謝辞

本研究の計算の一部は東北大学サイバーサイエンスセンターの並列コンピュータを利用することで行った．また，本センターの山下毅氏には，DFTB+1.2.2 の並列コンピュータへの最適化および計算結果の 3 次元可視化についてご協力いただいた．なお，本研究は日本学術振興会特別研究員研究奨励費（山崎：No. 251672），科研費（河野：No.24245001），文部科学省 X 線自由電子レーザー重点戦略研究課題（上田）の支援によって行われた．

参考文献

[1] K. J. Gaffney & H. N. Chapman, “Imaging Atomic Structure and Dynamics with Ultrafast X-ray Scattering,” *Science* **316**, 1444-1448 (2007).

[2] B. F. Murphy, K. Ueda et al., “Femtosecond X-ray-induced explosion of C_{60} at extreme intensity,” *Nature Commun.* **5**, 4281 (2014).

[3] M. Hoener *et al., Phys. Rev. Lett.* **104**, 253002 (2010).

[4] K. Yamazaki, T. Nakamura, N. Niitsu, M. Kanno, K. Ueda, H. Kono, “Communication: Two-step explosion processes of highly charged fullerene cations C_{60}^{q+} (q = 20–60) ,” *J. Chem. Phys.* **141**, 121105 (2014).

[5] K. Endo et al.“Analysis of Electron Spectra of Carbon Allotropes (Diamond, Graphite, Fullerene) by Density Functional Theory Calculations Using the Model Molecules,” *J. Phys. Chem. A* **107**, 9403-9408 (2003).

[6] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, Th. Frauenheim, S. Suhai, and G. Seifert “Self-consistent-charge density-functional tight-binding method for simulations of complex materials

properties," *Phys. Rev. B* **58**, 7260-7268 (1998).
[7] C. Köhler, G. Seifert, and T. Frauenheim, "Density-Functional based calculations for Fe_n (n≤32)," *Chem. Phys.* **309**, 23 (2005).
[8] B. Aradi , B. Hourahine , Th. Frauenheim, "DFTB+, a Sparse Matrix-Based Implementation of the DFTB Method," *J. Phys. Chem. A* **111**, 5678–5684 (2007).
[9] W.H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes* (*3rd ed.*), Cambridge Univ. Press (2007), Chap. 16.4
[10] 小西 貞則「多変量解析入門―線系から非線形へ―」岩波書店 (2010) 第 10 章.
[11] C. Nordling, & J. Österman, *Physics handbook for science and engineering 8th ed.* (Studentlitteratur: Lund, Sweden, 2006) pp 346
[12] S. Tomita, H. Lebius, A. Brenac, F. Chandezon, B. A. Huber "Kinetic-energy release and fragment distribution of exploding highly charged C_{60} molecules." *Phys. Rev. A* **65**, 053201 (2002).

[共同研究成果]

界面反応の第一原理シミュレーション

柳澤将[1,2], 稲垣耕司[2], 濵本雄治[2], 木﨑栄年[2], 森川良忠[2]
[1]琉球大学理学部, [2]大阪大学大学院工学研究科

1. 緒言

本研究では半導体デバイスや有機デバイス界面、燃料電池電極界面など、応用上重要な界面での電子状態と反応過程を高精度かつ高効率に計算し、界面の性質を支配する要因を明らかにし、それによって、より良い界面を設計する為の指針を与えることを目指している。本年度は第一原理電子状態計算プログラム STATE の高速化、特にレプリカ並列によって反応経路上の複数のレプリカを一つのジョブによって効率的に最適化し、エネルギー最小の反応経路を求める計算を可能にした。MPI プロセスでデータのやり取りを行う際に問題が生じていたが、計算機科学のエキスパートの助けを借りてこの問題を解決することができ、これによって、半導体表面エッチング過程や二酸化炭素還元触媒反応などの計算が効率的に行えるようになった。具体的には GaN 表面の水によるエッチング過程の研究、および、CO_2 の銅表面上での解離反応過程、水素化反応過程の研究を行った。CO_2 の銅表面上での解離反応過程については論文化するところまで進み、出版を行った。

また、有機機能性材料として重要なピセンの単結晶の電子状態について研究を行った。密度汎関数法により求めた固有関数・固有エネルギーを 0 次とし、多体摂動法の GW 近似に基づく自己エネルギーの計算を行った。GW 近似で得られる準粒子エネルギーは光電子分光の実験で観測される占有・非占有状態と物理的に対応する。GW 近似は、DFT 計算に比べて計算機負荷が膨大だが、我々は、計算対象のサイズに対するスケーリングで有利な GW space-time 法のプログラムを東北大学サイバーサイエンスセンターSX-9(以下 SX-9)上で整備し研究を進めた。本研究の計算シミュレーションは演算性能に加えて 2TB 以上の大きな主記憶が必要であり、SX-9 上で 32〜64 CPU を使用することによって初めて可能となった。

2. GW 近似による高精度なバンド計算の有機半導体結晶への適用

有機半導体を材料とする電子デバイスは、次世代の電子材料として国際的に研究開発がさかんである。最近の実験で、ルブレン、ペンタセン、ピセンなど一部の有機半導体の単結晶や薄膜で無機半導体に劣らない正孔移動度が報告され、従来考えられたホッピングによるキャリア伝導だけでなく、バンド伝導的なキャリア伝導機構の寄与の可能性が指摘されている。

著者らは最近、GW 近似による高精度なバンド計算を SX-9 上で、ルブレン、ピセンおよびフタロシアニンの単結晶について行い、一般にファン・デル・ワールス(vdW)力で緩く結合する分子間に、パイ電子雲が非局在化して広がることについての知見を得てきた[1-3]。本稿で

は、それらの研究例のうち、ピセンの単結晶に関する成果[2]を報告する。

2.1　計算方法

ピセン(分子式: $C_{22}H_{14}$)の単結晶の単位格子は 2 分子からなる単斜晶(図 1)で、格子定数は実験値に固定した。STATE プログラムコード[4]を用い、格子内の原子位置を最適化した。原子核のクーロンポテンシャルをノルム保存擬ポテンシャルで表し、価電子波動関数を平面波基底で展開した。電子の交換・相関ポテンシャルには PBE 汎関数を用い、分子間の vdW 力の記述のため、半経験的に vdW 力を PBE 汎関数に取り込んだ[5]。

こうして得られた固有関数・固有エネルギーを 0 次とし、多体摂動法の GW 近似に基づく自己エネルギーの計算を行った。GW 近似で得られる準粒子エネルギーは光電子分光の実験で観測される占有・非占有状態と物理的に対応する[6]。GW 近似は、DFT 計算に比べて計算機負荷が膨大だが、我々は、計算対象のサイズに対するスケーリングで有利な GW space-time 法[7]のプログラムを SX-9 上で整備し研究を進めた。本研究の計算シミュレーションは主に SX9 上で 32〜64 コアを使用して行われた。

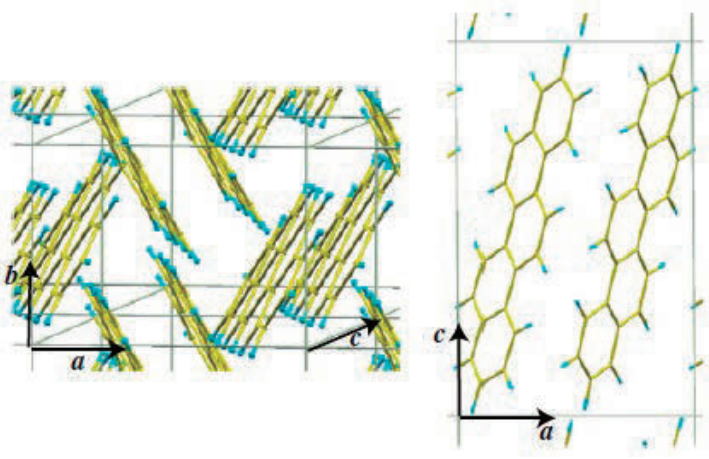


図 1.　ピセンの単結晶の図。*a, b, c* は結晶軸を表し、*b* 軸方向に分子がヘリングボーン状に積層する様子を示す。実線は単位格子を表し、単位格子の長さは、*a*: 0.8480 nm, *b*: 0.6154 nm, *c*: 1.3515 nm。格子ベクトルのなす角度は、$\alpha = \gamma = 90^\circ$ 、$\beta = 90.46^\circ$ 。

2.2　計算結果・考察

PBE によるバンド構造(図 2 の点線)は先行研究[8]とよく一致し、最高占有軌道に由来するバンド(HOMO バンド)がΓ-X, Γ-Y 方向で 0.2-0.5 eV 程度の、比較的大きなバンド幅を与えた。

た。最低空軌道由来のバンド(LUMO バンド)は、それに比べて分散が小さい。

次に、GW 近似によるバンド計算の結果について述べる(図 2 の実線)。本研究では SX-9 を利用し大規模な系での計算実行を可能にしており、収束の達成も容易になって、以降の議論に十分な計算値の収束 (HOMO バンド幅: 0.01 eV 以内)に達している。

直接バンドギャップは PBE より 1.4 eV 増加して、3.8 eV 程度となり、DFT で一般に過小評価されるバンドギャップについて大幅な改善が見られる。バンドギャップの実験値の報告はないが、結晶の励起エネルギー(光学ギャップ)の実験値が 3.1-3.3 eV 程度で、バンドギャップとのエネルギー差 0.5-0.7 eV が励起子結合エネルギーに相当すると考えると、有機結晶の一般的な励起子の描像に近く、バンドギャップの計算値は妥当と考えられる。

HOMO バンドをΓ点付近で放物線にフィットすることによって、Γ-Y 方向の有効正孔質量[m_h^*/m_e]は、PBE で 1.26、GW 近似で 1.06 と見積もられた。GW 近似による固体中の多体電子間相互作用の効果により、正孔キャリアは、より自由電子的になると説明される。このことは、ピセンで報告された数 $cm^2\ V^{-1}\ s^{-1}$ 程度の比較的高い正孔移動度[9,10]が、バンド伝導的な機構の寄与に由来する可能性を示唆すると考えられる。

上述のバンドの曲率の増加に対応し、特にΓ-Y 方向で、HOMO バンド(H^u)の幅が、GW 近似の自己エネルギー補正で 0.1 eV ほど増加するのが分かる(図 2)。これは、著者によるルブレン単結晶の先行研究[1]でも明らかになったように、対応する結晶中の方向(ここでは b 軸の、分子がスタックする方向)で、各分子の HOMO に由来する状態が分子間位置にまで非局在化することに対応している。実際、H^uバンドの Y 点での波動関数の空間分布を見ると、分子間で少なからず混成して広がる様子が分かる(図 3)。

最近の角度分解紫外光電子分光(ARPES)の実験によって、結晶中の 2 分子に由来する 2 枚のサブバンド(H^u, H^l)が分解されて観測され、高エネルギー側の H^uバンドのΓ-Y 方向の分散が 0.18 eV に対して、低エネルギー側の H^lバンドのΓ-Y 方向の分散は 0.15 eV 以下、と報告された[11]。本研究の計算結果は、H^lの方が H^uよりも分散は小さく、実験結果を定性的に再現できているが、実験値よりも倍程度大きな値であり、定量的に一致していない。その不一致の原因として、室温付近の実験であるために ARPES のピークに熱によるゆらぎの効果が現れるのに加え、電子–振動結合の効果もバンド幅の大きさと同程度のオーダーで寄与し、その結果、絶対零度下の精密な電子状態計算の結果とずれてくることが挙げられる。

今後は、電子状態計算の精密化に加え、電子–振動結合の評価も含めたバンド構造の再現も可能にし、実験結果の解釈や、有機半導体中のキャリア伝導の支配要因の解明に寄与していきたいと考える。

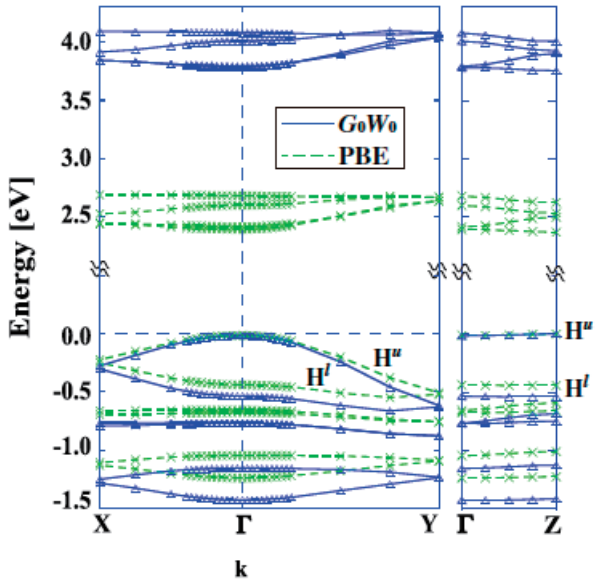


図 2. ピセン単結晶のバンド計算の結果。**Γ**-X、**Γ**-Y、**Γ**-Z 方向がそれぞれ結晶 a，b，c 軸に対応。バンド図は、**Γ**点の HOMO バンドエネルギーを基準とするエネルギー分散を表し、PBE によるバンド図を点線で、GW 近似によるバンド図を実線で示した。Hu（Hl）は、単位格子中の各分子の最高占有軌道(HOMO)由来の 2 つのバンドのうち、エネルギーが高い(低い)バンドを表す。

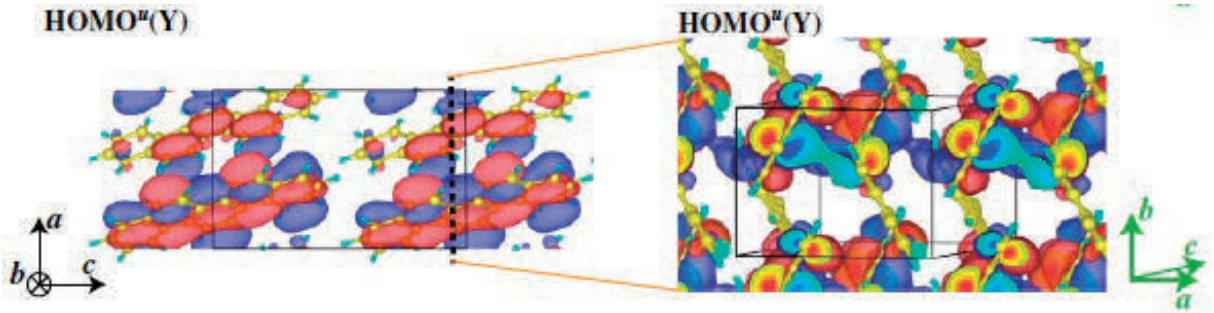


図 3. ピセン単結晶での HOMO(H^{u})の波動関数の等高面図。等高面は大きさ 2.21 Å$^{-3/2}$ に対応し、色の違いは波動関数の符号の違いを表す。単位格子を実線で示し、右図では、左図の太い点線による等高面の断面を表す。

3. 触媒援用加工法におけるエッチング化学反応解析

本章では触媒援用加工法(CAtalyst Reffered Etching method; CARE)におけるエッチング化学反応解析に第一原理量子シミュレーションを適用した例について述べる。CARE 法は、Pt 等の触媒をエッチング液中で加工対象表面に接触・揺動させ、その部分でのみエッチング液中の化学種を活性化して表面原子をエッチングすることにより、原子レベルで平滑な表面を得る方法(図 4) [12]であり、現在、SiC、GaN などのワイドバンドギャップ半導体に対しての加工技術開発が精力的に進められている。本研究の目的は第一原理量子シミュレーションを用いてエッチング反応を解析し、平坦化や触媒機能などのメカニズムを明らかにすることにより技術開発の方向性に示唆を与えることにある。ここでは LED やパワー制御半導体デバイスで重要な材料である GaN 表面の水による加工で平坦化される機構を明らかにした例について述べる。

計算には我々が開発した第一原理分子動力学シミュレーションコード STATE を用いた。STATE は密度汎関数法に基づいており、波動関数を平面波展開法、原子核ポテンシャルをウルトラソフト擬ポテンシャル、交換相関項を GGA-PBE の方法でそれぞれ扱っている。反応の活性化エネルギーは CI-NEB 法によって求めた。この手法では、2 つの準安定原子構造の間をつなぐ反応経路に沿って複数の原子構造を仮定し、反応経路方向以外の自由度について構造最適化しつつ反応座標を収束させていくことで最小エネルギー障壁の反応経路を探索する。また障壁近くの構造でのみ反応座標方向に極大エネルギーの構造を探索することで遷移状態を求める。複数の本研究では、1 つの MPI プロセスでこれらの複数の構造を同時に扱って反応経路最適化を行うプログラムを開発し、これを利用して解析を行った。ここでは、水分子の GaN 表面への解離吸着の反応性が表面のステップ部分とステップから原子が欠損したキンク部でどのように異なるかで平坦化メカニズムを調べた結果について示す。モデルは計算の簡単化のため現実のウルツ鉱構造でなく閃亜鉛鉱構造を用いている。(111)面のステップ-テラス構造とステップのキンク部を模した構造で生じる表面への水分子の解離吸着反応をそれぞれ NEB 計算により解析した(図 5、6)。活性化障壁はステップ部、キンク様部でそれぞれ 1.18eV、0.81eV となっておりキンク様部での反応が容易である結果となった。この差は、図からわかるようにキンク様モデルの場合は N 原子が回転することにより Ga を束縛する作用が弱められているためであると考えられる。この解析は水分子 1 つが解離吸着するエッチングのごく初期の過程だけであるが、水分子が次々に解離吸着して Ga をエッチングすると予想される反応過程において、

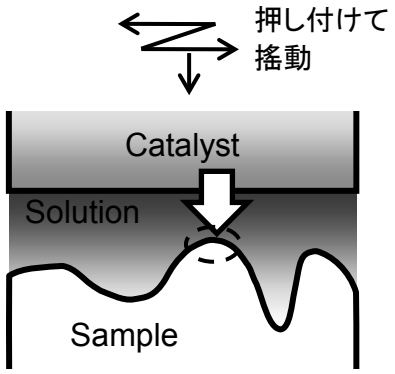
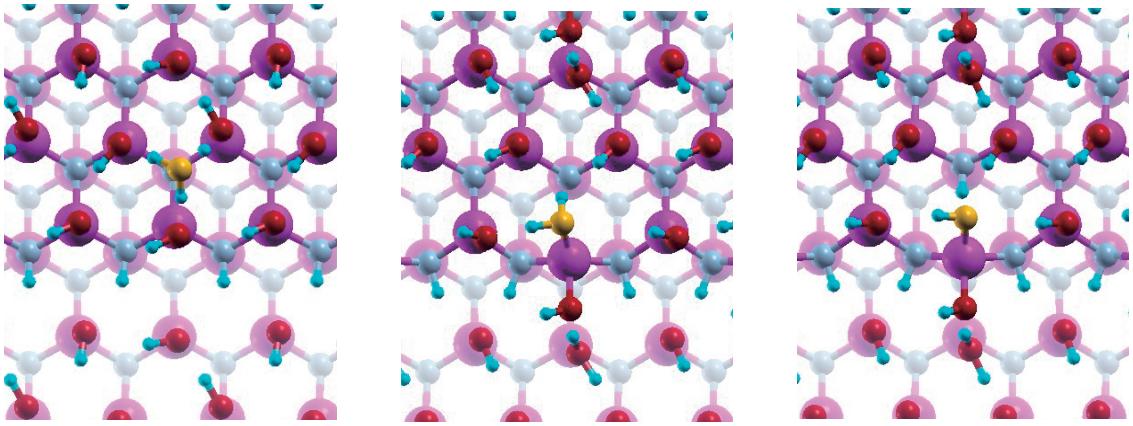


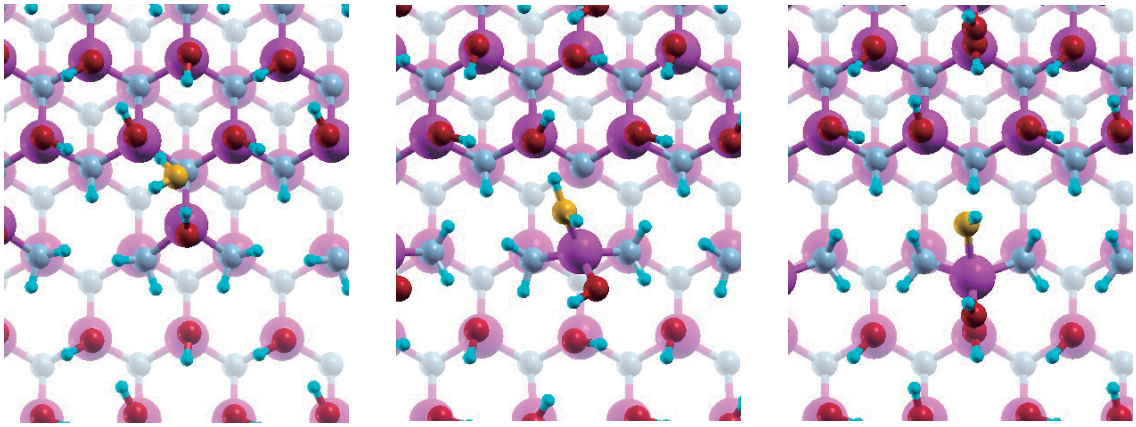
図 4. CARE 加工法の概念図
液中で触媒とサンプルを接触させ、そこでのみエッチング反応を起こさせることにより加工対象物の表面を平坦化する。

キンク様部分での反応では変形の自由度が大きく、原子構造の構造緩和の程度が大きいため障壁が下がることが容易に推察できる。この結果から、表面平坦化のメカニズムが、エッチングが表面ステップ構造のキンク部で起こりやすいためと理解される。



(a)　Initial model　　(b) Meta-stable　　(c) Final

図 5.　GaN ステップサイトへの水分子の解離吸着過程。球は原子を表し、色により元素が区別される。紫：Ga、灰：N、赤：O、水色：水素、黄：吸着する水分子の O。表面を上から見ている。奥にある原子ほど薄く描かれている。Ga-N のバックボンドが切れて Ga に水分子が吸着した(b)後、水分子の H が N を終端する。活性化障壁は 1.18eV であった。



(a)　Initial model　　(b) Meta-stable　　(c) Final

図 6.　GaN キンク様サイトへの水分子の解離吸着過程。図 2 と同様の表示である。ステップ部での吸着と同様に Ga-N のバックボンドが切れて Ga に水分子が吸着した(b)後、水分子の H が N を終端する。このモデルでは Ga の動きに伴って結合している N が回転していることが見て取られる。活性化障壁は 0.81eV であった。

今回の解析では、触媒の作用や反応点周辺に存在する他の水の分子は取り扱っておらず、今後の課題となっている。特に、水は水素結合を介してネットワークを形成しており、これ

が反応において重要な役割を果たしていることは言うまでもない。周囲の水を含めた系全体での反応障壁を解析するためには、時々刻々変化するネットワーク構造に対応した反応障壁を熱力学的に平均化する必要があり、非常に多量の計算資源が要求される。今後、ブルームーン法やメタダイナミックス法などを用いた自由エネルギー障壁計算を効率化するプログラム開発を実施する予定である。

謝辞

本研究は、東北大学サイバーサイエンスセンターのスーパーコンピュータ SX-9 を利用することで実現することができた。また、研究にあたっては同センター関係各位に有益なご指導とご協力をいただいた。本研究の一部は JSPS 科研費 23226004 の助成を受けた。

参考文献

[1] S. Yanagisawa, Y. Morikawa and A. Schindlmayr, Phys. Rev. B **88**, 115438 (2013).

[2] S. Yanagisawa, Y. Morikawa and A. Schindlmayr, Jpn. J. Appl. Phys. **53**, 05FY02 (2014).

[3] S. Yanagisawa, K. Yamauchi, T. Inaoka, T. Oguchi and I. Hamada, Phys. Rev. B **90**, 245141 (2014).

[4] Y. Morikawa, H. Ishii and K. Seki, Phys. Rev. B **69**, 041403 (2004).

[5] S. Grimme, J. Comput. Chem. **27**, 1787 (2006).

[6] L. Hedin, Phys. Rev. **139**, A796 (1965).

[7] M. M. Rieger et al., Comput. Phys. Commun. **117**, 211 (1999); C. Freysoldt *et al*., Comput. Phys. Commun. **176**, 1 (2007).

[8] T. Kosugi, T. Miyake, S. Ishibashi, R. Arita, and H. Aoki, J. Phys. Soc. Jpn. **78**, 113704 (2009).

[9] N. Kawasaki, Y. Kubozono, H. Okamoto, A. Fujiwara, and M. Yamaji, Appl. Phys. Lett. **94**, 043310 (2009).

[10] H. Okamoto, N. Kawasaki, Y. Kaji, Y. Kubozono, A. Fujiwara, and M. Yamaji, J. Am. Chem. Soc. **130**, 10470 (2008).

[11] Q. Xin, S. Duhm, F. Bussolotti, K. Akaike, Y. Kubozono, H. Aoki, T. Kosugi, S. Kera, and N. Ueno, Phys. Rev. Lett. **108**, 226401 (2012).

[12] 応用物理:原英之,佐野泰久,有馬健太,山内和人,触媒基準エッチング法, 77, 2, 168-171, (2008).

[大規模科学計算システム]

SX-ACE における HPCG ベンチマークの性能評価

小松 一彦[1)]，江川 隆輔[1)]，磯部 洋子[1)3)]，緒方隆盛[3)]，滝沢 寛之[2)]，小林 広明[1)]
1)東北大学サイバーサイエンスセンター，
2)東北大学大学院情報科学研究科，
3)日本電気株式会社

1. 背景

これまでスーパーコンピュータシステムの性能を測定するベンチマークとして，Linpack[1]，HPC Challenge，NAP Parallel Benchmark など，さまざまなベンチマークが提唱されている．1993 年に始まったスーパーコンピュータをランク付けする TOP500[2]では，Linpack が採用されている．Linpack は密係数行列の連立一次方程式の解を求めることで，1 秒あたりの浮動小数点演算回数(FLOPS)を算出している．この性能値に基づいて，TOP500 では世界のスーパーコンピュータの上位 500 位までを年に 2 回，スーパーコンピュータに関する国際会議である ISC(International Supercomputing Conference)と SC(International Conference on High Performance Computing, Networking, Storage and Analysis)において，発表している．

しかしながら，Linpack と実アプリケーションで実行される計算内容が乖離しており，Linpack によって得られた性能値が，必ずしも実際のスーパーコンピュータで実行されている実アプリケーションに当てはまらないという問題がある．これは，Linpack が主にプロセッサの理論演算性能やシステム規模などの総演算性能を測定するためのベンチマークとして開発されており，実アプリケーションの実効性能に重要となる演算性能以外のメモリアクセス性能やネットワーク性能を測定することが難しいためである．さらに，Linpack の実行時間も問題に挙がっている．Linpack で高い性能値を出すためには，出来るだけ大規模な問題を解く必要があり，スーパーコンピュータの規模が大きくなるにつれ，その実行時間が長くなっている．大規模なスーパーコンピュータでは部品点数が多いこともあり，故障による長時間実行の困難さや，性能測定のためだけに費やされる電気代などの費用などが問題となっている．このような実アプリケーションとの乖離や実行時間の長さなどのほか，使い勝手などの演算性能以外の評価をできないなどの問題を解決するための，短時間で，かつ，より実アプリケーションに求められる要因を測定できるベンチマークが長い間求められている．

2. HPCG の概要

High Performance Conjugate Gradient (HPCG)[3]は，前述した Linpack の問題点を解決するこ

とを目的として，より実アプリケーションの特徴に即したベンチマークとして開発されている．HPCG は Linpack の特徴である，分かりやすさ，実行しやすさなどの Linpack の特徴を引き継ぎつつ，演算性能だけでなく，メモリアクセス性能・ネットワーク性能にも比重が置かれており，より実アプリケーションに近いベンチマークとなっている．例えば，集団縮約通信や，様々なサイズにおける 1 対 1 通信，間接メモリアクセスなども含まれている．2013 年 6 月の ISC’13 において新たなベンチマークとして提唱され，11 月 SC13 における実行方法，実行時間，最適化方法などの検討，2014 年 1 月と 3 月に開催されたワークショップにおける議論を経て，2014 年 6 月の ISC’14 において初めて HPCG ランキングが公開されている．

HPCG は疎行列の連立一次方程式の解を求めることで性能値を算出している．有限要素法を用いて離散化された対称疎行列を係数行列とする連立一次方程式を，ガウス・ザイデル法(Symmetric Gauss-Seidel Method)を用いたマルチグリッド前処理(Multigrid Preconditioning)付き共役分配法(Conjugate Gradient)で解いている．性能値は，この連立一次方程式を解くのに規定されている浮動小数点演算回数を実行時間(最適化に必要な時間も含む)で割ることで算出している．

図 1 に HPCG(Release 2.4)の処理の流れを示す．HPCG は前処理，準備実行，本番実行，後処理の 4 つのフェーズに大別されている．前処理フェーズでは，連立一次方程式のための疎行列やマルチグリッド前処理のための粗い行列，行列の検証，プロセスの割当などが行われる．準備実行フェーズでは，最適化など何も手を加えられていない状態で疎行列ベクトル積(SpMV)，マルチグリッド(MG)関数，共役分配(CG)関数全体，残差の集約などが実行され，最終結果に必要なデータが測定される．本番実行では，まず最適化を施された CG 関数を 1 回実行することで，試行回数が決定される．次に，試行回数に応じた CG 関数が実行され,その実行時間が計測される．最後の後処理フェーズでは，前処理フェーズでの検証結果や実行の検証を踏まえ，正常であれば，測定条件(プロセス数・スレッド数，問題サイズなど)，検証結果，そして，測定結果(経過時間，浮動小数点演算回数の理論値，GFLOPS 値)が出力される．

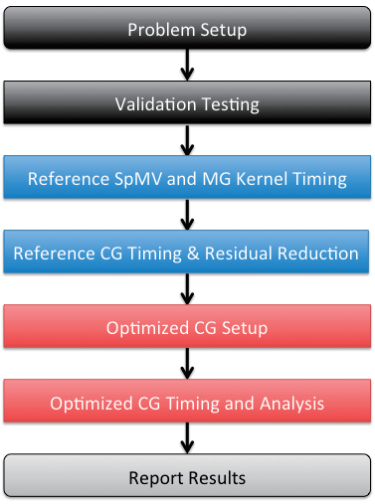


図 1. HPCG の処理の流れ

3. サイバーサイエンスセンターにおける HPCG の取り組み

サイバーサイエンスセンターと NEC は，HPCG の提案に先立って， 2012 年後半から実アプリケーションにおいてよく用いられる疎行列ベクトル積の高速処理に取り組んでいる．それらの知見を生かし，HPCG をターゲットとして，サイバーサイエンスセンターで運用されているベクトル型スーパーコンピュータにおける最適化・高速化を進めてきた．

まずは HPCG の特徴を分析するために，コード中の浮動小数点演算命令，メモリアクセス命令から演算密度(Operational Intensity)の算出を行った．その結果，HPCG に使われている主要な関数のデータ転送量と浮動小数点演算数の比(Bytes/Flop)が 6 以上であることが分かった．これによりノード性能を高めるためには，メモリアクセス性能が非常に重要であることが分かる．次に，コードの初期分析とともに，コードの最適化を検討した．特に，HPCG において主要な計算であるガウス・ザイデルを用いたマルチグリッド法の並列化手法について検討を重ねた．図 2，3 に示すような同一色の点において，節点の値の更新順序に依存関係がないとみなし並列に計算するマルチカラーオーダリング法や最適化前の依存関係を保つハイパープレーン法などを試行した．その結果，収束回数の増加を抑えることができ，かつ，ベクトル型スーパーコンピュータ SX-ACE において高い性能を引き出すことができるハイパープレーン法を採用した．さらに，SX-ACE に搭載されているソフトウェア制御可能なオンチップメモリ ADB を最大限に活用するために，保存すべきデータの選別やグリッドサイズの調整などのチューニングを行った．その他，疎行列の格納方式，通信の最適化などのさまざまな最適化を通じて，本番実行フェーズで実行するコードを作成した．

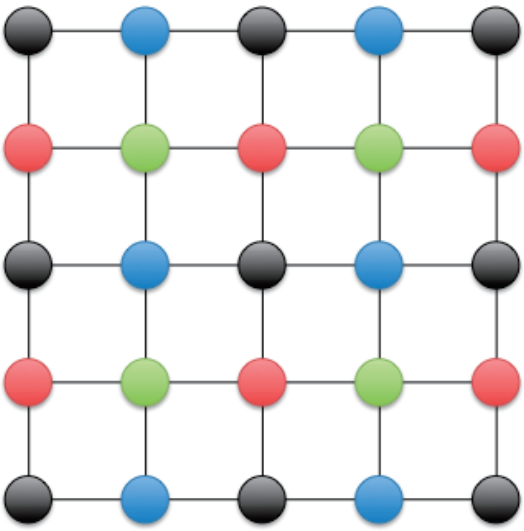


図 2. 8 カラーオーダリング法(2 次元平面図)

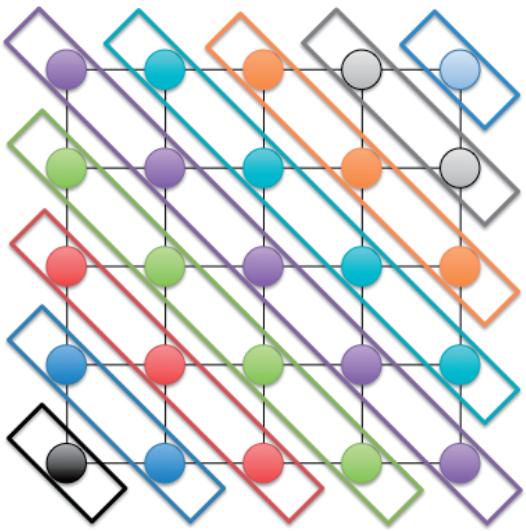


図 3. ハイパープレイン法

4. SX-ACE における HPCG の評価

SX-ACE 512 ノードを用いて最適化を施した HPCG の評価を行い，SC14 でのランキングへ登録を行った．表 1 に，SC14(2014 年 11 月)の HPCG BOF で発表されたランキングを示す．全 25 のスーパーコンピュータが登録されており，HPCG における性能が高い順にランキングされている．その他，Linpack における性能値(HPL)やランキング(HPL Rank)，HPCG と Linpack や理論性能値との比率である効率も発表されている．

この表を見ると，必ずしも Linpack における性能値が HPCG における性能値と同一ではないことが分かる． HPCG ランキングの 2 位と 3 位のスーパーコンピュータを例に取ってみてみると，HPCG ランキング 2 位の京コンピュータは HPL ランキングでは 4 位，HPCG3 位の Titan が HPL2 位となっており，HPCG と HPL における順位が入れ替わっているのが分かる．このように，HPCG は演算性能の他にメモリアクセス性能やネットワーク性能などスーパーコンピュータの総合的な性能を評価しているため，順位の入れ替わりがしばしば発生する．

東北大学の SX-ACE は，他のシステムに比べて理論性能値が高くなく，Linapck の性能を見ると，0.123Pflops と Top500 にはランクインできないシステム規模であるにも関わらず，HPCG 性能値は 0.0135Pflops と 18 位にランク付けされている．これは SX-ACE の実行効率が他のスーパーコンピュータと比べ，格段に高いためである．図 4 に理論演算性能に対する実行効率を示す．横軸はスーパーコンピュータを示し，実行効率が高い順番に並べている．青，黄，緑，橙，赤，紫の各色は，SX-ACE，GPU，Intel Xeon，BlueGene，富士通 SPARC，Intel Xeon Phi のプロセッサをそれぞれ搭載するスーパーコンピュータである．図 4 や表 1 を見ると，実行効率が他のスーパーコンピュータは 1.0～4.8%であるのに対して，SX-ACE は 10%超えと非常に高い効率であることが分かる．SX-ACE はメモリバンド幅が 256GB/s と高く，演算性能とメモリバンド幅の比率も 1.0B/F と高いためである．また，3 章で述べた最適化より，その演算性能およびメモリアクセス性能を効率的に引き出すことができたため，高い実行効率を達成できた．

表 1. SC14 において発表された最新 HPCG ランキング

Rank	Site	HPL [Pflops]	HPL Rank	HPCG [Pflops]	HPCG/ HPL[%]	HPCG/ Peak[%]
1	NSCC/Guangihou	33.9	1	0.632	1.86	1.2
2	RIKEN AICS	10.5	4	0.461	4.39	3.8
3	DOE/OS ORNL	17.6	2	0.322	1.83	1.2
4	DOE/OS Argonne Lab.	8.59	5	0.167	1.94	1.0
5	Sewiss CSCS	6.27	6	0.105	1.67	1.3
6	Leibniz Rechenzentrum	2.90	14	0.0833	2.87	2.6
7	DOE/OS L Barkley Nat Lab.	1.65	24	0.0786	4.76	3.1
8	GSIC Center, TiTech	2.78	15	0.07300	2.63	1.3
9	Max-Planck	1.28	34	0.06100	4.77	4.2
10	CEA/TGCC-GENCI	1.36	33	0.0510	3.75	3.1
11	Exploration and Production Eni S.p.A	3.00	12	0.0489	1.63	1.2
12	Grand Equipement National de calcul intensif	2.07	N/A	0.0448	N/A	2.2
13	U. of Tokyo	1.04	36	0.0448	4.30	3.9
14	Texas Advanced Computing center	5.168	7	0.0440	0.85	0.5
15	IFERC	1.240	30	0.0426	3.44	2.8
16	HWC U of Stuttgart	2.763	N/A	0.0391	N/A	1.0
17	SURF sara	0.848	N/A	0.0195	N/A	1.8
18	Cyberscience Center Tohoku U	0.123	---	0.0134	10.89	10.2
19	Meteo France	0.469	79	0.0110	2.35	2.2
20	Meteo France	0.465	80	0.00998	2.15	2.2
21	Bull Angers	0.430	N/A	0.00970	N/A	1.8
22	U of Toulouse	0.255	184	0.00725	2.84	2.6
23	Cambridge U.	0.240	241	0.00385	1.60	1.0
24	GSIC Center, TiTech	0.148	392	0.00370	2.50	1.7
25	SURF Sara	0.154	499	0.00250	1.63	1.2

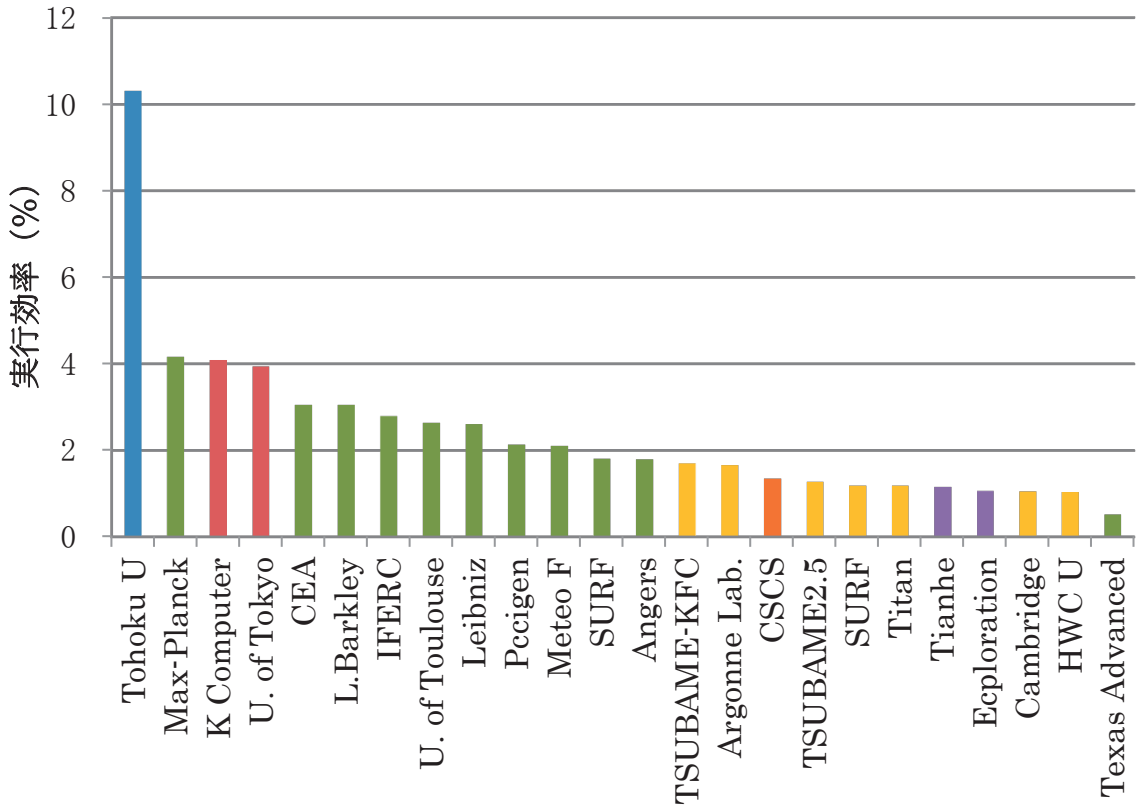


図 4. HPCG における実行効率の比較

5. まとめ

演算性能だけでなくメモリアクセス性能やネットワーク性能などスーパーコンピュータの総合力を評価するベンチマークとして，新たに提案されている HPCG におけるサイバーサイエンスセンターの取り組みについて述べた．サイバーサイエンスセンターにおいて運用されている SX-ACE に向けた HPCG の最適化，その評価結果について述べた．SC14 において発表された最新の HPCG ランキングを通じて，SX-ACE が他のスーパーコンピュータと比べて，非常に高い実行効率を達成できることが分かった．SX-ACE は演算性能だけでなくメモリアクセス性能も重視しているため，HPCG だけでなく実アプリケーションにおいても高い実効性能を実現できると期待できる．今後は HPCG の取り組みを通じて得た知見を実アプリケーションにも応用し，これまで以上に研究・社会に役立つスーパーコンピュータを目指していきたい．

参考文献

[1] Linpack, http://www.netlib.org/linpack/

[2] TOP500 Supercomputing Sites, http://www.top500.org/

[3] HPCG Benchmark, http://www.hpcg-benchmark.org/

［大規模科学計算システム］高速化推進研究活動報告第６号より転載

ベクトルコンピュータにおける高速化

小林広明[1]　江川隆輔[1]　小松一彦[1]　岡部公起[1]　大泉健治[2]　小野　敏[2]　山下　毅[2]
佐々木大輔[2]　森谷友映[2]　齋藤敦子[2]　撫佐昭裕[3]　松岡浩司[3]　渡部修[3]　曽我　隆[4]　山口健太[4]

[1] スーパーコンピューティング研究部
[2] 情報部情報基盤課
[3] 日本電気株式会社
[4] NEC ソリューションイノベータ株式会社

本章では，本センターにおいてこれまで培われてきた SX-9 のベクトル性能を向上するための手法や事例を紹介する．これらの手法や事例は今後導入されるベクトルスーパーコンピュータ SX-ACE においても有用である．

1.　ベクトルコンピュータの特徴

近年，高性能計算機（HPC）システムを活用したシミュレーションの応用範囲は益々広がっており，同時にシミュレーションの規模も拡大し続けている．シミュレーションを行う研究者からの要求は計算規模を拡大すると同時に，計算結果を得るまでの時間は短縮したいというものである．このような研究者の要求を満たすため HPC システムの性能も年々向上し続けている．

HPC システムに採用されるプロセッサあるいはコア(以下コア)には大きく分類して，スカラ型とベクトル型がある．スカラ型コアは命令レベル並列性に基づきデータを一つずつ実行するのに対して，ベクトル型コアはデータレベルの並列性に着目して複数の演算器(パイプライン)を用いて一度に複数の処理を同時に実行する．図 1-1 にスカラ命令とベクトル命令の動作を示す．この図は A(I)=B(I)+C(I)と D(I)=E(I)+F(I)の 2 つの計算式を 100 回繰り返す DO ループの処理を表している．スカラ命令では DO 変数が 1 の時の A(1)と D(1)の計算結果を求め，次に DO 変数が 2 の時の A(2)と D(2)の計算結果を求めるという処理を，DO 変数が 100 になるまで繰り返す．それに対してベクトル命令は A(I)=B(I)+C(I)の計算を DO 変数 1 から 100 まで一度に行い，次に D(I)=E(I)+F(I)の計算を DO 変数 1 から 100 までを一度に行うことにより実行時間の短縮を図る．ベクトル型コアを搭載する HPC システムは，このベクトル命令の特性を活かすため，高いメモリ帯域を具備してメモリからコアへの大量のデータ供給を可能にしている．

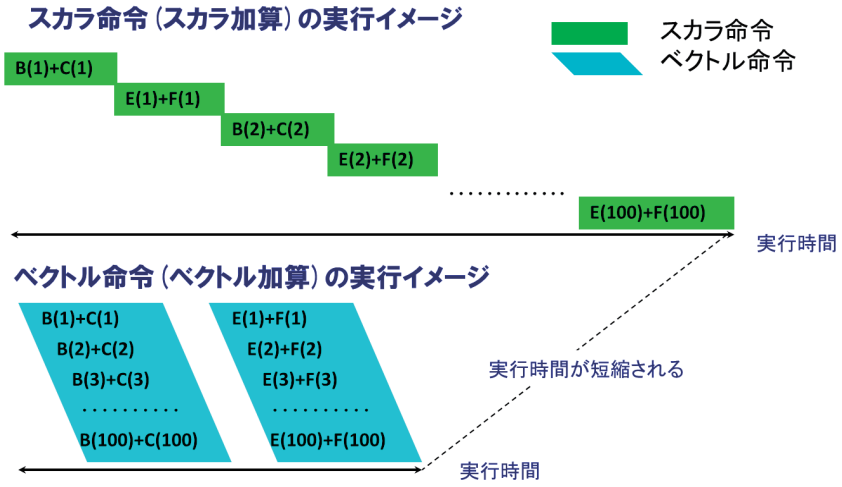


図 1-1　スカラ命令とベクトル命令の様子

表 1-1 に近年の HPC システムが搭載しているプロセッサの性能と内包するコア数，プロセッサ当たりのメモリバンド幅を示す．SX-9 以外のプロセッサでは複数の演算コアを搭載するマルチコア構成により演算性能の向上を図っているため，コア当たりのメモリバンド幅は小さくなる．しかしスカラプロセッサではメモリのバンド幅を最大限に使うためには複数のコアを使用する必要があるが，SX-ACE では 1 つコアで最大のメモリバンド幅を使うことが可能となっている．

このようにベクトルコンピュータは，高い理論演算性能とメモリ帯域を実装することで，実アプリケーションの実行において高い実効性能の達成を可能にしている．

表 1-1 HPC システムのプロセッサ性能と B/F 値

HPC システム名	プロセッサ			メモリバンド幅 (GB/s)
	名称	理論性能(GFlop/s)	コア数	
SX-9	—	102.4	1	256
LX406Re-2	Intel Xeon E5-269v2	230.4	12	59.7
SR16000	POWER7	245.1	8	128
FX10	SPARC64 IXfx	236.5	16	85.3
「京」	SPARC64 VIIIfx	128.0	8	64
SX-ACE	—	256.0	4	256

2.　高速化の手法

SX システムの超高速性を十分に引き出すために重要なことは，プログラムの中でベクトル命令によって処理される部分の比率，すなわちベクトル化率と，生成されるベクトル命令の効率を可能な限り高めることである．そのためには，プログラムの最適化状況の把握・分析を行い，主要な性能指標に基づき，高速化を行う必要がある．ここでは，そのためのツール類及び，それらの性能指標と各指標に沿った高速化の取り組みかたについて述べる．

2.1 性能解析ツール

(1)　編集リスト

編集リストはベクトル化および自動並列化に関する情報をソースプログラムの左側に表示したリストであり，どのループがベクトル化されたかを確認することができる．編集リストはコンパイラオプション「-R5」を指定することにより"入力ファイル名.L"という名前で出力される．

```
!編集リスト
FILE NAME:t5.f
PROGRAM NAME: sub
FORMAT LIST
 LINE    LOOP     FORTRAN STATEMENT
    1:               subroutine sub(a,b,c,d,z,ix)
    2:               real,dimension(100)::a,b,c,d,x,y,z
    3:               integer ix(100)
    4: V------>        do I = 1,100
    5: |       I          call sub2(x,a,b,I)
    6: |                  y(I) = c(I) + d(I)
    7: |       S          z(ix(I)) = z(ix(I)) + x(I) + y(I)
    8: V------         enddo
    9:               end
```

図 2-1　編集リストの例

主なループ情報，スカラ情報，手続インライン情報など編集リストの出力イメージを次に示す．

①　ループ全体がベクトル化される場合

ベクトル化されたループに"V"が表示される．

```
V------>   do I=1,100
|            a(I)=b(I)+c(I)
V------    enddo
```

図 2-2　ベクトル化された DO ループの編集リスト

②　ベクトル化されない場合

ベクトル化されないループには"+"が表示される．

```
+------>   do I=1,100
|            print *,a(I)
+------    enddo
```

図 2-3　ベクトル化されない DO ループの編集リスト

③ 部分ベクトル化の場合

ベクトル化不可の処理がある行には“S”が表示される.

※部分ベクトルとは，ループにベクトル化を阻害する部分が含まれている場合，その前後で自動的にループを分割し，ベクトル化可能な部分だけをベクトル化する拡張機能である.

```
V------>   do I =1,100
|            a(I)=b(I)+c(I)
|      S     print *,a(I)
V------      enddo
```

図 2-4 部分ベクトル化の編集リスト

④ 配列式をベクトル化した場合(1)

ループの先頭と最後の行が同じである場合，ループの構造は“=”で表示される.

```
             real a(90),b(90),c
 V=======    a(1:90)=b(1:90)+c
```

図 2-5 ベクトル化された配列式の編集リスト

⑤ 配列式をベクトル化した場合(2)

ループ融合している場合は，その範囲について“V”で表示される.

※ループ融合とは，繰返し数が等しい DO ループまたは配列式が複数個，連続して存在している場合一つのループに融合することである. ただし，各ループで使われているデータの定義・参照関係に，融合によって矛盾が生じる場合は行わない.

```
            real a(90),b(90),c
           integer d(90),e(90)
V------>    a(1:90)=b(1:90)+c
|           d(1:90)=int(a(1:90))
V------     e(1:90)=d(1:90)+1
```

図 2-6 ループ融合とベクトル化された配列式の編集リスト

⑥ 手続呼出しがインライン展開された場合

インライン展開された手続がある行には“I”が表示される.

```
         I  call sub2(x,a,b,c,I)
```

図 2-7 インライン展開の編集リスト

⑦ 多重ループが一重化された場合

一重化されたループの外側ループに“W”，内側ループに“＊”が表示される.

```
W------>   do J =1,100
|*---->      do I =1,100
||             a(I,J)=b(I,J)+c(I,J)
|*----       enddo
W------     enddo
```

図 2-8 多重 DO ループの一重化の編集リスト

⑧ ループの入れ換えが行われた場合

入れ換えた結果ベクトル化されるループに“X”が表示され，ベクトル化されなくなるループには“+”が表示される.

```
X------>    do J =1,1000
|+------>     do I =1,10
||              a(I,J)=b(I,J)+c(I,J)
|+------       enddo
X------      enddo
```

図 2-9 DO ループの入れ換えの編集リスト

⑨ ADB にデータがバッファリングされる場合

ADB を使用したベクトルロード/ストアがある行に文字“A”が表示される.

```
+------>    do J =1,100
|V------>    do I =1,100
||      A      a(I,J)=a(I,J+1)+b(I,J)
|V------      enddo
+------      enddo
```

図 2-10 ADB の使用の編集リスト

⑩ ベクトル化と並列化される場合

ベクトル化と並列化が行われたループに“Y”が表示される.

```
Y------>      do i=1, 10000
|               c(i) = c(i) + a(i) * b(i)
Y------       end do
```

図 2-11 ベクトル化と並列化された DO ループの編集リスト

⑪ 並列化される場合

並列化が行われたループに“P”が表示される.

```
P------>          do j = 1, 100
|V------>          do i = 1, 100
||                   d(i,j) = 0.0d0
|V------            end do
P------           end do
```

図 2-12 並列化された DO ループの編集リスト

⑫ 複数の情報がある場合

一行に対して複数の情報がある場合, “M”が表示される.

※配列 a はベクトル化され, 配列 b はループ長が短いためループが展開されるので, この一行には複数の情報がある.

```
M======           a(1:10000)=0.0d0 ; b(1:3)=0.0d0
```

図 2-13 複数の情報がある場合の編集リスト

(2) 簡易性能解析機能(FTRACE)

簡易性能解析情報はコンパイラオプションに「-ftrace」を指定することで, 手続(サブルーチンや関数)ごとの性能解析情報を採取することができ, プログラム実行後に解析情報が標準出力ファイルに出力される. 特に, チューニング対象とする手続を選択する際に活用すると良い. 図 2-14 に FTRACE 機能による解析リストの例を示す. FTRACE では実効性能と平均ベクトル長などの情報を手続(サブルーチンや関数)単位で取得することができる.

① PROC.NAME	② FREQUENCY	③ EXCLUSIVE TIME[sec](%)	④ AVER.TIME [msec]	⑤ MOPS	⑥ MFLOPS	⑦ V.OP RATIO	⑧ AVER. V.LEN	⑨ VECTOR TIME	⑩ I-CACHE MISS	⑪ O-CACHE MISS	⑫ BANK CONFLICT CPU PORT	⑫ BANK CONFLICT NETWORK
subc	100	7.594(42.2)	75.936	79426.5	52676.9	99.48	256.0	7.594	0.0000	0.0000	0.032	0.000
subb	100	5.227(29.1)	52.272	77121.3	38261.9	99.22	256.0	5.227	0.0000	0.0000	0.000	0.000
suba	100	5.173(28.7)	51.728	58600.1	1933.29	98.97	256.0	5.173	0.0000	0.0000	0.000	0.000
test	1	0.000(0.0)	0.518	772.0	0.0	0.00	0.0	0.000	0.0000	0.0000	0.000	0.000
total	301	17.994(100.0)	59.780	7276.83	38902.9	99.28	256.0	17.993	0.0001	0.0001	0.032	0.000

図 2-14 解析リストの出力例

表示される各項目の意味は以下のとおりである.

①PROC.NAME :手続名
②FREQUENCY :手続の呼び出し回数
③EXCLUSIVE TIME :手続の実行に要した占有の CPU 時間(秒)と, 手続全体の実行に要した CPU 時間に対する比率
④AVER.TIME :手続の 1 回の実行に要した平均 CPU 時間(ミリ秒)
⑤MOPS :1 秒間に実行された演算数を 100 万単位で示した値
⑥MFLOPS :1 秒間に実行された浮動小数点演算数を 100 万単位で示した値
⑦V.OP RATIO :ベクトル演算率
⑧AVER V.LEN :平均ベクトル長
⑨VECTOR TIME :ベクトル命令実行時間(秒)
⑩I-CACHE MISS :命令キャッシュミスにより発生した合計時間(秒)
⑪O-CACHE MISS :オペランドキャッシュミスにより発生した合計時間(秒)
⑫BANK CONFLICT :バンクコンフリクト時間(秒)
CPU PORT :CPU ポート競合時間(秒)
NETWORK :メモリネットワーク競合時間(秒)

2.2. ベクトル化率の向上

(1) ベクトル化率

ベクトル型スーパーコンピュータではプログラム中のベクトル処理可能な部分を高速に実行している. 図 2-15 に同一のプログラムをスカラ処理する場合とベクトル処理する場合の実行時間に関する概念図を示す.

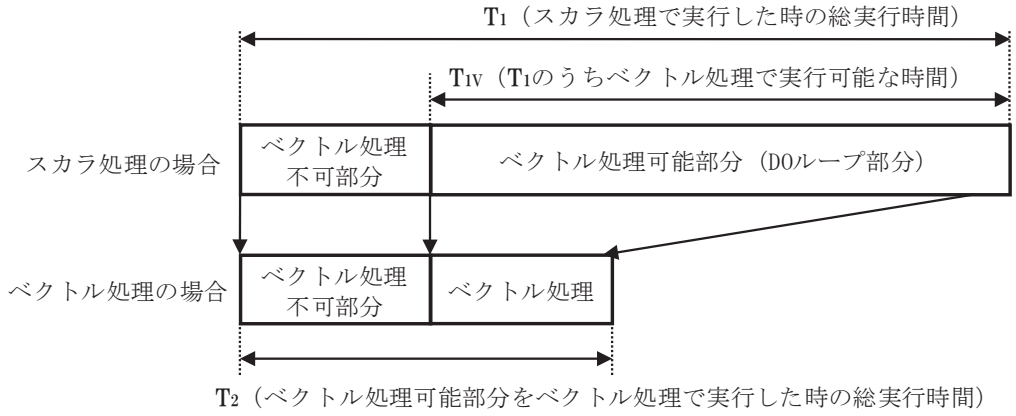


図 2-15　ベクトル処理による実行時間短縮のイメージ

ここで，あるプログラムをスカラ処理で実行する場合の総実行時間を T_1，そのプログラムでベクトル処理が可能な部分の実行時間を T_{1V} とすると，ベクトル化率 α は以下の式で定義される．

$$\text{ベクトル化率}\quad \alpha = \frac{T_{1V}}{T_1}$$

また，そのプログラムをベクトル処理する場合の性能向上比 P は，スカラ処理性能とベクトル処理性能の比を β とすると，

$$\text{性能向上比}\quad P = \frac{1}{(1-\alpha)+\frac{\alpha}{\beta}}$$

と表される．

この式から，ベクトル化率と性能向上比の関係は図 2-16 のようになる（アムダールの法則）．この図からベクトル化率80%程度では大きな性能向上は見られず，ベクトル化率90%を超えたあたりから急激に性能が向上していることがわかる．ベクトル型スーパーコンピュータで高い実効性能を得るためにはベクトル化率を 100%にできる限り近付ける必要がある．

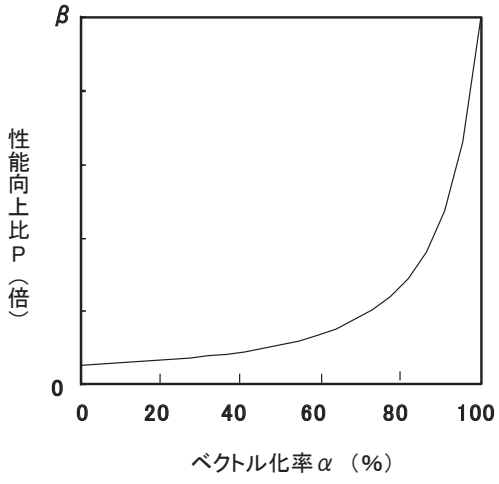


図 2-16　ベクトル化率と性能向上比（アムダールの法則）

ベクトル化率を求めるため，プログラムをスカラ実行する場合とベクトル実行する場合のそれぞれの実行時間が必要である．しかし，これらの実行時間は同時に得られないのでベクトル化率の算出は困難である．そのため SX-9 ではベクトル化率の代わりにベクトル演算率をベクトル化の指標としている．FTRACE などの性能解析機能に出力される情報はベクトル演算率であり，ベクトル演算率は，プログラムで処理される全演算要素数に対するベクトル演算命令で処理される演算要素数の割合で算出し，ほぼベクトル化率とみなせる指標であるため，後述のベクトル化率はベクトル演算率のこととする．

(2) ベクトル化を阻害する要因

ベクトル化を阻害する要因を以下に示す．

① 依存関係の有無が判断できない
※前の繰り返しで定義した値を参照する場合や間接参照

② I/O 処理(OPEN/CLOSE/READ/WRITE 文)

③ ユーザ関数(SUBROUTINE, FUNCTION)の呼び出し

④ ループの繰り返し数がループ本体の実行前に決定しない

⑤ ループの入口および出口が一つではない

⑥ ベクトル化対象外の精度を使用している
※2 バイト整数型，4 倍精度実数型，4 倍精度複素数型，1 バイト論理型，文字型,構造型はベクトル化対象外の精度

2.3 平均ベクトル長の拡大

(1) 平均ベクトル長

ベクトル処理を効率的に行う上で重要な指標にベクトル長がある．ベクトル長はベクトル化対象の DO ループの繰り返し回数のことであり，効率良くベクトル処理を行うためには，できるだけループ長を長くする必要がある．

一般に命令を発行してから結果が返ってくるまでに遅延時間が存在する．この遅延時間を立ち上がり時間と呼ぶ．図 2-17 にベクトル処理における立ち上がり時間および演算時間の概念図を示す．網掛け部分は演算器が稼働する時間を示しており，演算を行うレジスタのデータを読み込み，演算結果をレジスタに格納するまでの時間となる．図 2-18 にベクトル長と立ち上がり時間の関係を示す．立ち上がり時間はベクトル長が異なる場合でも一定である．よって総演算量が等しい場合，短ベクトル長処理を繰り返すよりも長ベクトル長処理を一括して行う方が実行時間を短くすることができる．このように高速化を図るためには，DO ループのベクトル長を十分に確保し，演算時間に対する立ち上がり時間の占める割合を低くすることが重要である．図 2-19 にベクトル長と立ち上がり時間の関係を示す．ベクトル処理の立ち上り時間が発生するため，ベクトル長が交差ループ長より短い(4 以下)場合はベクトル化を行わない方が良い場合がある．交差ベクトル長とは，ベクトル化した場合とベクトル化しない場合とで実行時間が等しくなるループ長のことである．

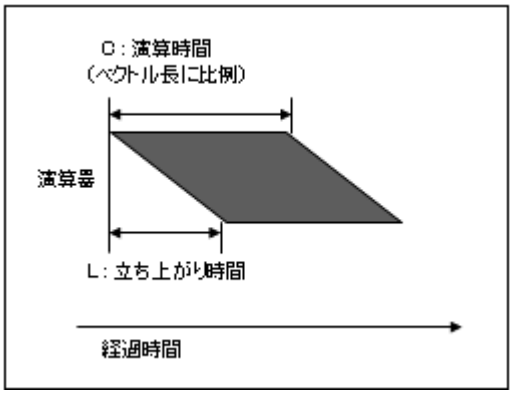


図 2-17　立ち上がり時間

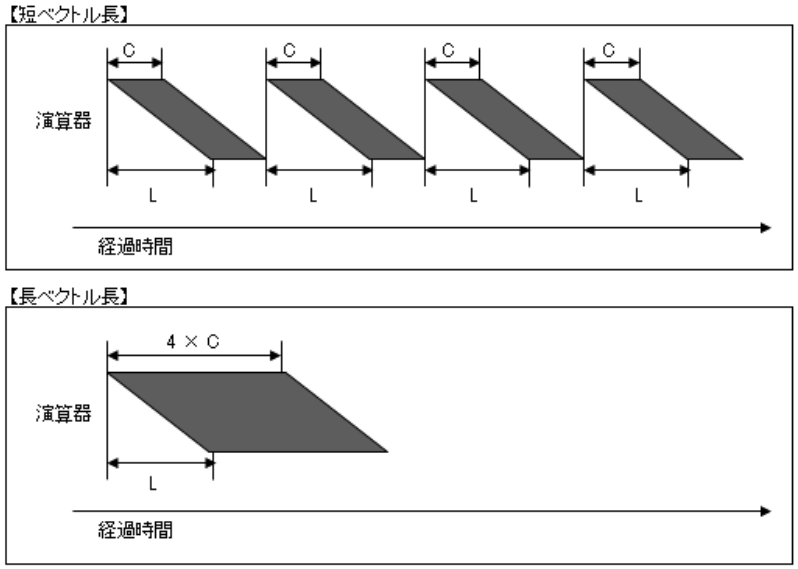


図 2-18　ベクトル長と立ち上がり時間

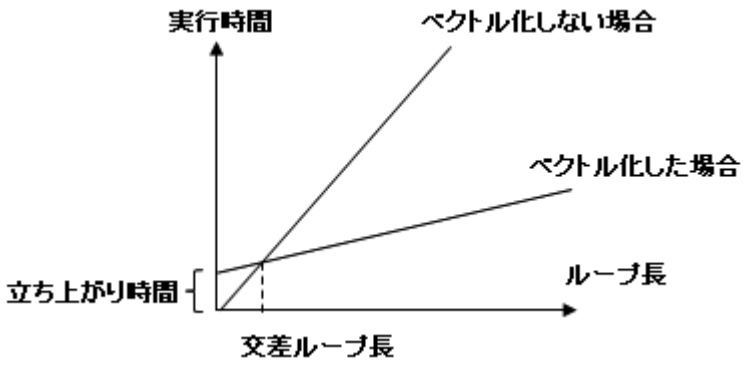


図 2-19　ベクトル長と立ち上がり時間の関係

2.4 メモリアクセスの効率化

(1) メモリ競合

SX-9 ではマルチメモリバンクシステムを採用しており，メモリを 32 のバンクグループに分割している．各バンクグループには 2 要素(16 バイト)ずつ配列データが格納される．

図 2-20 は連続アクセスとなる DO ループの例，図 2-21 は格納される配列データ(二次元の配列サイズが(128,64)のとき)の概念図を示しており，CPU からメモリにアクセスする際，CPU 内の全ポートを使用して効率的にメモリアクセスを行う．

```
 !連続アクセス
do j=1,ny
do i=1,nx
  :
  Rs(i,j)=Rs(i,j)+Cf(i,j)
end do
end do
```

図 2-20 連続アクセスの DO ループの例

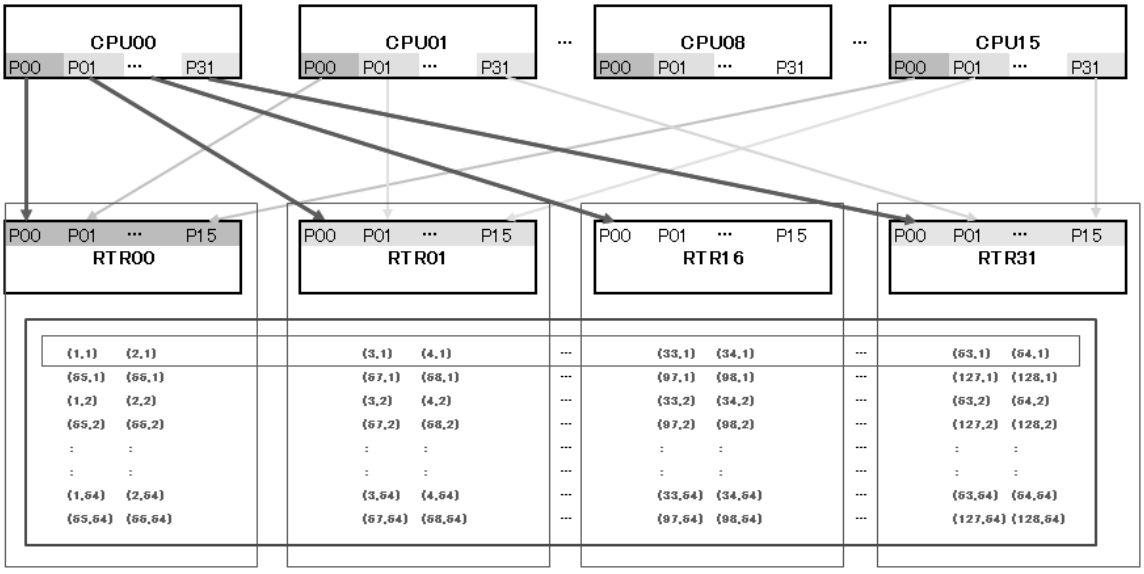


図 2-21 メモリバンクに格納される配列データの概念図(連続アクセス)

一方，図 2-22 はストライドアクセスとなる DO ループの例，図 2-23 は格納される配列データ(二次元の配列サイズが(128,64)のとき)の概念図を示しており，CPU からメモリをアクセスする際，アクセスするバンクグループが限定され CPU 内の特定のポート(00, 16)だけを使用してメモリアクセスを行うことになる．このように CPU 内における同一のポートにロード・ストアが集中した時に発生する CPU ポート競合や，同一のメモリバンクへのアクセスなどで発生するメモリネットワーク競合はメモリアクセスが遅延するため高速化の妨げとなる．

```
!ストライドアクセス
do  j=1,ny
do  i=1,nx,32
   :
  Rs(i,j)=Rs(i,j)+Cf(i,j)
end do
end do
```

図 2-22 ストライドアクセスの DO ループの例

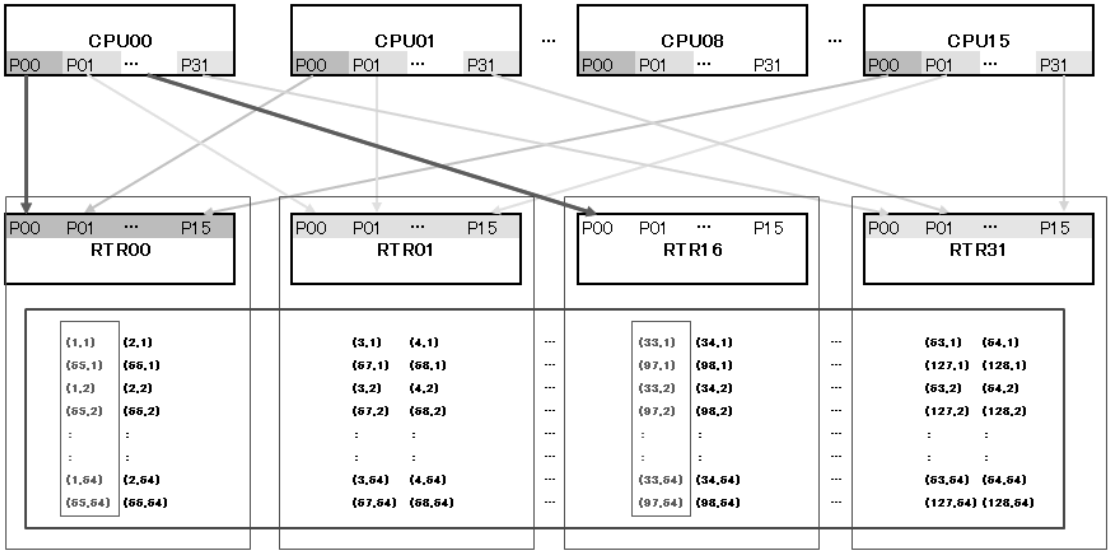


図 2-23 メモリバンクに格納される配列データのイメージ(ストライドアクセス)

以降にメモリアクセス性能を改善するための指針として，メモリアクセスパターンの改善と ADB の活用について述べる．

(2) メモリアクセスパターンの改善

図 2-24 にメモリへアクセスパターンの種類を示す．アクセスパターンの種類としては，連続アクセス，ストライドアクセス，間接アクセスの 3 種類に分けることができる．

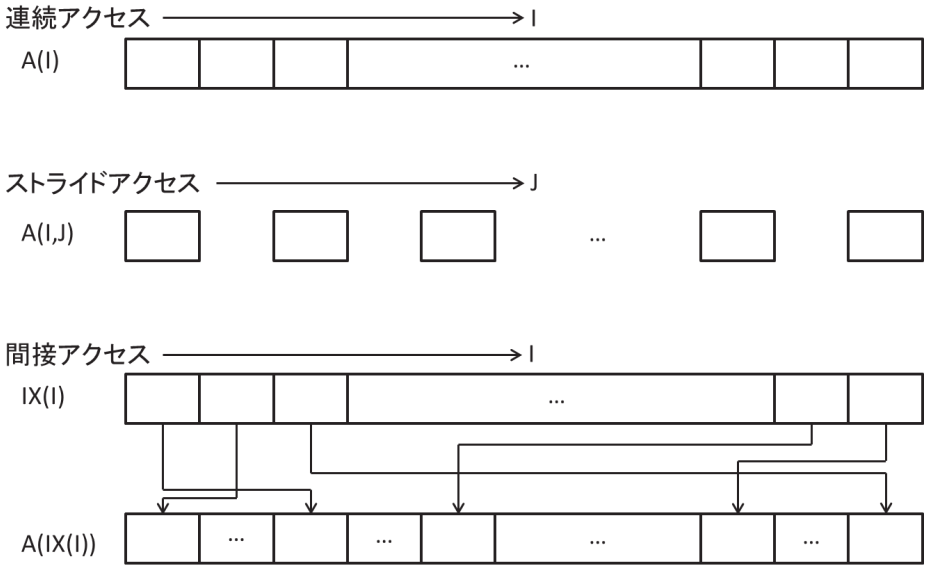


図 2-24 メモリのアクセスパターンの種類

図 2-21 に示すように，連続アクセスの場合が最も効率的なメモリアクセスを行うことができる．ストライドアクセスの場合には，図 2-23 に示すように，使用できるバンク数が限られてくる．この場合，ループ入れ換えなどにより，なるべく最初の次元でメモリアクセスを行うようにする．その際，ループインデックスが繰り返しごとに 1 ずつ増加あるいは減少するようにできでば連続アクセスとなり，より効率的なメモリアクセスが可能となる．間接アクセスの場合，間接アクセスとなる配列(図 2-24 の間接アクセスにおける配列 A)が何度も参照されるならば，最初に，この配列のデータを作業用の配列にコピーしておき，以降のループでは作業用の配列をアクセスするようにすることにより，連続アクセスにすることができる．

(3) ADB の活用

SX-9 では CPU-主記憶装置間に ADB(Assinable Data Buffer)と呼ばれるベクトルデータを選択的にバッファする機能を有している．この ADB を活用することによって，メモリアクセス性能を改善できる．

コンパイラは，再利用性があると判断した配列を自動的に ADB にバッファリングするが，コンパイラが判断できなかった配列については，ON_ADB 指示行を用いることによって，明示的に該当配列を ADB にバッファリングすることを指示することができる．例えば，間接アクセスにおいて，間接アクセスとなる配列(図 2-24 の間接アクセスにおける配列 A)に対し ON_ADB 指示行を用いて ADB にバッファリングしておくことにより，この配列のロードを高速に行えるようになる．

なお，ADB の容量は限られているため，ADB にバッファリングするデータのサイズに注意し，再利用性のあるデータを ADB にバッファリングするように指定することが重要である．

3.　ベクトル化率向上の事例

3.1. ループ内の WRITE 文の括り出し

(1)　チューニング方針

SX-9 では，DO ループの一部にベクトル化対象外の文が存在すると，ベクトル化されない．もしくは，部分ベクトル化となり，完全にはベクトル化されない．　図 3-1 にチューニング前のコードを示す．このコードでは，DO ループ内にエラー検出時に実行される WRITE 文が含まれている．入出力文はベクトル化対象外の処理であるため，DO ループ全体がベクトル化されていない．図 3-2 のチューニング前の FTRACE 情報を確認すると，ベクトル化率が 0.0%であり，全くベクトル化されていないことが分かる．そこで，ベクトル化阻害要因である WRITE 文を DO ループの外に移動させることにより，DO ループをベクトル化可能にする．

```
!チューニング前
12: +------>          DO J=2, JF
13: |+----->             DO I=2, IF
 :
18: ||                     IF (HZ(I,J) .GE. -30.0) THEN
19: ||                        ZZ = Z(I,J,1) - RX*(M(I,J,1)-M(I-1,J,1))
20: ||                &                     - RY*(N(I,J,1)-N(I,J-1,1))
21: ||                        IF (ABS(ZZ) .LT. GX) ZZ = 0.0
22: ||                        DD = ZZ + HZ(I,J)
23: ||                        IF (DD .LT. GX) DD = 0.0
24: ||                        DZ(I,J,2) = DD
25: ||                        Z(I,J,2)  = DD - HZ(I,J)
28: ||                        IF(ABS(Z(I,J,2)).GT.100.0) THEN
29: ||                        NC=1
30: ||                        WRITE(*,'(A24,3I6)')'Over flow Z at (K,I,J) :', KK,I,J
31: ||                        WRITE(*,*)'Within Region :', NREG
32: ||                        WRITE(*,*)'Computation is unstable.'
34: ||                        END IF
36: ||                     END IF
37: |+-----             END DO
 38: +------          END DO
```

図 3-1　WRITE 文括り出し前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
40	6.863(3.1)	171.564	558.3	195.4	0.00	0.0	0.000	0.000	2.239	0.000	0.000	【チューニング前】

図 3-2　WRITE 文括り出し前の FTRACE 情報

(2)　チューニング内容

図 3-3 にチューニング後のコードを示す．チューニング前の DO ループではエラー検出時に WRITE 文によるエラー情報の出力を行っているが，本修正においては，DO ループ内ではエラー検出時にフラグ検出用配列にフラグをセットするようにした．そして，WRITE 文によるエラー情報の出力処理は DO ループ外に移動させ，出力が必要な場合のみ処理を行うようにした．これにより，DO ループ内にはベクトル化対象外の文がなくなるため，ベクトル化が可能となる．

```
!チューニング後
! フラグの初期化
23:                    IFLG1 = IF * JF + 1
24: W*=====            IDX = IFLG1
 :
! メイン処理
!   エラーの検出のみ実施して，エラー出力処理をループの外に出す
27: +------>           DO J=2, JF
28: |V----->              DO I=2, IF
 :
33: ||                       IF (HZ(I,J) .GE. -30.0) THEN
34: ||                          ZZ = Z(I,J,1) - RX*(M(I,J,1)-M(I-1,J,1))
35: ||                 &                      - RY*(N(I,J,1)-N(I,J-1,1))
36: ||                          IF (ABS(ZZ) .LT. GX) ZZ = 0.0
37: ||                          DD = ZZ + HZ(I,J)
38: ||
39: ||                          IF (DD .LT. GX) DD = 0.0
40: ||                          DZ(I,J,2) = DD
41: ||                          Z(I,J,2)  = DD - HZ(I,J)
 :
45: ||                          IF(ABS(Z(I,J,2)).GT.100.0) THEN
53: ||                             IDX(I,J) = I + (J - 1) * IF
54: ||                          END IF
56: ||                       END IF
57: |V-----              END DO
58: +------            END DO
 :
! エラー出力処理
61: +------>           DO J=2, JF
62: |V----->              DO I=2, IF
63: ||                       IFLG1 = MIN(IDX(I,J), IFLG1)
64: |V-----              ENDDO
65: +------            ENDDO
66:                    IF ( IFLG1 .LT. IF * JF + 1 ) THEN
67:                       INDX_J = (IFLG1 - 1) / IF + 1
68:                       INDX_I = IFLG1 - (INDX_J - 1) * IF
69:                       NC=1
70:
71:                       WRITE(*,'(A24,3I6)')'Over flow Z at (K,I,J) :',
73:                       WRITE(*,*)'Within Region :', NREG
74:                       WRITE(*,*)'Computation is unstable.'
75:                    ENDIF
```

図 3-3　WRITE 文括り出し後のコード

(3)　性能分析

図 3-4 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
40	6.863(3.1)	171.564	558.3	195.4	**0.00**	0.0	0.000	0.000	2.239	0.000	0.000	【チューニング前】
40	0.079(0.0)	1.966	63568.9	18871.6	**99.54**	238.9	0.078	0.000	0.000	0.004	0.012	【チューニング後】

図 3-4　WRITE 文括り出し前後 FTRACE 情報

DO ループ全体がベクトル化されたことにより，ベクトル化率が 0.0%から 99.5%に向上し，実行時間は 6.86 秒から 0.08 秒に短縮することができた.

3.2. 部分ベクトル化の回避

(1) チューニング方針

図 3-5 にチューニング前のコードを示す. 39 行目の変数 vmax は前の繰り返しで定義された値を参照しているためベクトル化の阻害要因となるが, コンパイラは最大値を求める演算をマクロ演算に置き換えてベクトル化を行おうとする. 最大値もしくは最小値を求める演算がある場合, ベクトルパイプライン毎にベクトル演算を行い, 最後に各演算結果のマスクを取るマクロ演算を適用してベクトル化を行う. しかし, 39 行目で求める vmax の値を 40 行目で参照する必要があるため, ベクトル化を阻害する依存関係となり, 39 行目と 40 行目をスカラ処理する部分ベクトル化となる. 図 3-6 に示すチューニング前の FTRACE 情報を確認するとベクトル化率が 67.5%で十分にベクトル化がされていないことが分かる. ベクトル化を阻害する依存関係を排除し, マクロ演算を適用してループ全体をベクトル化する.

```
!チューニング前
23: +------>          do iy = 1,ny
24: |V----->            do ix = 1,nx
 :
37: ||                     absv =max(absvxi,absvxj,absvyi,absvyj)
38: ||                     cf   = sqrt(cs2(ix,iy)+ca2(ix,iy))
39: ||       S             vmax = max(vmax,absv+cf)
40: ||       S             dtmin = min(dtmin,dlmin/vmax)
41: |V-----              enddo
42: +------            enddo
```

図 3-5 部分ベクトル化回避前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
821	26.020(7.2)	31.693	1664.6	496.5	**67.53**	255.9	2.017	0.001	1.503	0.003	1.113	**【チューニング前】**

図 3-6 部分ベクトル化回避前の FTRACE 情報

(2) チューニング内容

チューニング前では vmax の最大値判定と更新, dtmin の最小値判定と更新をループ回数分行っている. dtmin の判定式は不変値 dlmin を vmax で除算しているため, dtmin は vmax が更新されたときのみ dtmin が更新される. そのため, DO ループで vmax の最大値を求め, DO ループ終了後に一回 dtmin の判定を行うことで同じ結果が得られる. 図 3-7 にチューニング後のコードを示す. 前述したとおり, MAX 関数で求める vmax を MIN 関数で参照することが依存関係であるため, MIN 関数の演算を DO ループの外に移動することで依存関係が解消され, vmax を求める処理はマクロ演算が適用されてベクトル化が行われる.

```
!チューニング後
23: +------>          do iy = 1,ny
24: |V----->            do ix = 1,nx
 :
37: ||      A              absv = max(absvxi,absvxj,absvyi,absvyj)
38: ||                     cf   = sqrt(cs2(ix,iy)+ca2(ix,iy))
39: ||      A              vmax = max(vmax,absv+cf))
40: |V-----              enddo
41: +------            enddo
42:                    dtmin = min(dtmin,dlmin/vmax)
```

図 3-7 部分ベクトル化回避後のコード

(3) 性能分析

図 3-8 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
821	26.020(7.2)	31.693	1664.6	496.5	**67.53**	255.9	2.017	0.001	1.503	0.003	1.113	【チューニング前】
821	2.273(0.8)	2.769	14592.6	4215.6	**99.19**	255.9	1.971	0.000	0.124	0.019	1.029	【チューニング後】

図 3-8 部分ベクトル化回避前後の FTRACE 情報

部分ベクトルが解消しベクトル化率が 67.5%から 99.2%に改善したことで実行時間は 26.0 秒から 2.3 秒に短縮することができた.

4. 平均ベクトル長の拡大の事例

4.1. ループに関する情報の追加による最適化の促進

(1) チューニング方針

多重ループの最内にあるループが自動ベクトル化の対象となるため, 最内ループのベクトル長が短い場合, 十分な性能を発揮できない. 図 4-1 のチューニング前のコードではループの中に配列式や配列関数を使用しており, その部分が最内ループとなりベクトル化の対象となる. 図 4-2 の FTRACE 情報では平均ベクトル長が 67.0 であり十分なベクトル長でないことが分かる. ここでは, コンパイラがループ長を判断してベクトル長を伸ばす方法で最適化を行う. ループの演算処理量が多いなどの問題からこのような最適化が行われない場合もあるが, ループに関する情報をコンパイラに明示することでベクトル長を伸ばす最適化が行われる場合がある. このような高速化の事例を以下に示す.

```
!チューニング前
 5:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 6:            real*8:: loglhTP(Np), dy(Nch)
 :
15: V------>  do ip=1,Np
16: |V===== A   dy = y - xp(1,:,ip)  &
17: |                          - xp(Mu+1,:,ip) - xp(Mu+Ms,:,ip)
18: |
19: |V===== A   loglhTP(ip) = - 0.5d0 * ( &
20: |                          dble(Nch) * log( 2.0d0 * pi )            &
21: |                        + log( product( xp(Ndim,:,ip) ))           &
22: |                        + sum( dy * dy / xp(Ndim,:,ip) )           &
23: |                          )
24: V------   enddo
```

図 4-1 ループに関する情報を追加する前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
60	73.899(73.1)	1231.647	2404.9	576.8	89.76	**67.0**	63.853	0.000	0.000	0.001	41.961	【チューニング前】

図 4-2 ループに関する情報を追加する前の FTRACE 情報

(2) チューニング内容

ベクトル化対象となる配列式と配列関数のループ長(変数 Nch)は 3, 外側 ip の DO ループの長さは 1,280,000 であり, この情報を追加することでコンパイラはより効率の良い最適化を実施する. 図 4-3 にチューニング後コードを示す. PARAMETER 文を追加することでコンパイラは最内 DO ループのループ長が 3 であることを認識するため, ループの展開を行い, 外側のループでベクトル化を行う. ただし, このチューニング内容は変数 Nch, Np の値が不変であることを前提としている場合のみ使用することが出来るので注意が必要となる.

```
!チューニング後
 6:            parameter(Nch=3,Np=1280000)
 7:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 8:            real*8:: loglhTP(Np), dy(Nch)

15: V------> do ip=1,Np
16: |*=====    dy = y - xp(1,:,ip)
17: |               - xp(Mu+1,:,ip) - xp(Mu+Ms,:,ip)
18: |
19: |*=====    loglhTP(ip) = - 0.5d0 * ( &
20: |                         dble(Nch) * log( 2.0d0 * pi )            &
21: |                       + log( product( xp(Ndim,:,ip) ))           &
22: |                       + sum( dy * dy / xp(Ndim,:,ip) )           &
23: |                         )
24: V------  enddo
```

図 4-3 ループに関する情報を追加後のコード

(3) 性能分析

図 4-4 にチューニング前後の性能値を示す.

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

       60   73.899( 73.1)  1231.647   2404.9    576.8 89.76  67.0     63.853   0.000   0.000    0.001    41.961 【チューニング前】
       60    0.289(  1.0)     4.813  23763.3  12232.7 99.60 256.0      0.267   0.000   0.000    0.017     0.111 【チューニング後】
```

図 4-4 ループに関する情報を追加する前後の FTRACE 情報

外側のループをベクトル化対象とすることで平均ベクトル長が 67.0 から 256.0 となり, 十分な性能を発揮することで, 実行時間が 73.9 秒から 0.3 秒に高速化された.

4.2. 部分配列を DO ループに書き替え, ループ交換を実施

(1) チューニング方針

チューニング前コードおよびチューニング前の FTRACE 情報は前項と同じ図 4-1 と図 4-2 である. 前項では, 変数 Nch, Np の値が不変であるという条件のもと PARAMETER 文を追加することでコンパイラの最適化を促進させた. ここでは変数 Nch, Np の値が不定の場合のチューニング方法を示す.

(2) チューニング内容

変数 Nch, Np の値が不定の場合のチューニング方法を示すが, ここで Nch の値が小さく, Np の値が十分に大きいことが分かっているものとする. 図 4-5 にチューニング後コードを示す. 配列式と配列関数を DO ループの形に変形し, ループ分割を行っている. このとき作業配列を新たに用意し使用する. また, Nch＜Np であるため, Np の DO ループがベクトル化の対象となるように最内にループの入れ換えを行っている.

```
!チューニング後
 5:            real*8:: y(Nch), xp(Ndim,Nch,Np)
 6:            real*8:: logIhTP(Np), dy(Nch)
 8:            real*8:: wrk_dy,wrk
 9:            real*8:: wrk_sum(Np),wrk_prd(Np)

24: V------>   wrk_prd=1.d0
25: V------    wrk_sum=0.d0
26: +------>   do j=1,Nch
27: |V----->   do ip=1,Np
28: ||      A    wrk_dy=y(j)-xp(1,j,ip)                    &
29: ||                 -xp(Mu+1,j,ip)-xp(Mu+Ms,j,ip)
30: ||      A    wrk_prd(ip)=wrk_prd(ip)                   &
31: ||                    * xp(Ndim,j,ip)
32: ||      A    wrk_sum(ip)=wrk_sum(ip)                   &
33: ||                  +(wrk_dy**2/xp(Ndim,j,ip))
34: |V-----    end do
35: +------    end do
36:            wrk=dble(Nch)*log(2.0d0*pi)
37: V------>   do ip=1,Np
38: |       A    logIhTP(ip)=-0.5d0*(wrk                  &
39: |                +log(wrk_prd(ip))+wrk_sum(ip))
40: V------    enddo
```

図 4-5 部分配列を書き換え後のコード

(3) 性能分析

図 4-6 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
60	**73.899**(73.1)	1231.647	2404.9	576.8	89.76	**67.0**	63.853	0.000	0.000	0.001	41.961	【チューニング前】
60	**0.314**(1.1)	5.232	25804.5	11742.5	99.54	**256.0**	0.287	0.000	0.000	0.026	0.101	【チューニング後】

図 4-6 部分配列を書き換え前後の FTRACE 情報

外側のループをベクトル化対象とすることで平均ベクトル長が 67.0 から 256.0 となり, 実行時間が 73.9 秒から 0.3 秒に短縮された.

5. メモリ競合の回避の事例

5.1. ADB によるメモリアクセス低減

(1) チューニング方針

0.(3)で述べているように, ADB を利用することにより, より効率的なメモリアクセスを行うことができる. その事例として, 図 5-1 を挙げる.

図 5-1 において最内側の DO ループがベクトル化されているが, 配列 b はこのベクトル化されたループに入る度にメモリからのロードを行うため, メモリ競合が発生する. 図 5-2 の FTRACE 情報が示すように, チューニング前のコードでは, 実行時間 26.6 秒に対してバンクコンフリクト時間が 18.1 秒を占めている. この配列 b に着目してみると, ベクトル化されている DO ループの外側の DO ループ(DO 変数 l のループ)による繰り返しによって配列の参照位置が変わることはなく, 最内 DO ループにおいては, 同じ添え字 j のデータを毎回メモリからロードすることが分かる. 従って, 配列 b は最内 DO ループにおいて再利用性があることが分かる.

通常, コンパイラはループに対する最適化処理の中で配列の再利用性解析を行う. しかし, 本事例のように, ループからの飛び出しがあるような場合にはコンパイラはループの最適化処理を行わない. そのため, 配列の再利用性解析も行われず, 再利用性がある配列に対してもコンパイラは自動でADBに載せるための処理を行わない. このような場合, コンパイラ指示行により明示的に配列を ADB に載せることを指示し, ADB 経由でのメモリアクセスを行うことにより, メモリに直接アクセスする頻度を削減し, バンクコンフリクト時間を短縮する.

```
!チューニング前
335: ||+----->              do l=1,lmax
336: |||V--->                 do i=is,imax(j)+1
337: ||||                        if(a(l).lt.b(i,j)) exit
338: |||V---                   end do
339: |||                       is    =i
340: |||                       iwk(l)=i
341: ||+-----               end do
```

図 5-1 ADB によるメモリアクセス低減前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
126	26.574(100.0)	210.906	1166.8	342.1	88.72	114.4	22.488	0.000	0.000	0.012	**18.072**	【チューニング前】

図 5-2 ADB によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

ON_ADB 指示行を用いて，配列 b を ADB に載せることを指示する．チューニング後のコードを図 5-3 に示す．これにより 1 回目のロード命令はメモリから行われるが，2 回目以降のロード命令は ADB から行われることになる．この結果，メモリへのアクセスを削減し，バンクコンフリクトによる実行時間の増加を解消する．

```
!チューニング後
335: ||+----->              do l=1,lmax
336: |||          !cdir on_adb(b)
337: |||V--->                 do i=is,imax(j)+1
338: ||||    A                    if(a(l).lt.b(i,j)) exit
339: |||V---                   end do
340: |||                       is    =i
341: |||                       iwk(l)=i
342: ||+-----                 end do
```

図 5-3 ADB によるメモリアクセス低減後のコード

(3) 性能分析

図 5-4 にチューニング前後の性能値を示す．

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
126	26.574(100.0)	210.906	1166.8	342.1	88.72	114.4	22.488	0.000	0.000	0.012	**18.072**	【チューニング前】
126	9.547(99.9)	75.771	3247.7	952.3	88.72	114.4	5.455	0.000	0.000	0.012	**1.036**	【チューニング後】

図 5-4 ADB によるメモリアクセス低減前後の FTRACE 情報

メモリへのアクセスが減少することにより，バンクコンフリクト時間が短縮され，実行時間が 26.6 秒から 9.5 秒に短縮することができた．

5.2. ループ融合によるベクトルロード・ストアの削減

(1) チューニング方針

通常，コンパイラによる最適化では，同じループ構造の DO ループが連続していると自動でループ融合を行う．ループ融合を行うことで同一の配列データに対するメモリアクセス(ロード，ストア)回数を減らすことができる．図 5-5 にチューニング前のコードを示す．チューニング前コードでは同じループ構造の DO ループが連続しているが OpenMP 指示行を含んでいるためループ融合が行われない．コンパイラは PARALLEL リージョン単位で最適化を行うため，ループ融合が行われず連続する複数の DO ループで同じ配列データのメモリアクセスが発生し，配列 sm3dh はメモリロード 2 回，メモリストア 2 回の命令が発生している．図 5-6 の FTRACE 情報を確認すると実行時間 13.1 秒に対してバンクコンフリクト時間が 7.4 秒を占めているため，ループ融合によりメモリアクセス回数を減らしバンクコンフリクト時間が短縮される．

```
!チューニング前
7414:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7419:              !$OMP DO
7420: P------>           do iz=1,lz
7421: |+----->           do iy=1,ly
7422: ||V---->           do ix=3,lx-2
  :
7437: |||     A             sm3dh(ix,iy,iz)=sm3d(ix,iy,iz)
7438: |||               &             +sdltxinv*( du(ix-1,iy,iz)-du(ix,iy,iz) )
  :
7443: ||V----            enddo
7444: |+-----            enddo
7445: P------            enddo
7446:              !$OMP END DO NOWAIT
7447:              !$OMP END PARALLEL
  :
7454:                    sdltthalfgx=0.5d0*adltt*sgravx
  :
7457:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7462:              !$OMP DO
7463: P------>           do iz=1,lz
7464: |+----->           do iy=1,ly
7465: ||V---->           do ix=3,lx-2
7466: |||     A             sm3dh(ix,iy,iz)=sm3dh(ix,iy,iz)
7467: |||               &                 +sdltthalfgx*sr3d(ix,iy,iz)
  :
7473: ||V----            enddo
7474: |+-----            enddo
7475: P------            enddo
7476:              !$OMP END DO NOWAIT
7477:              !$OMP END PARALLEL
  :
7498:              !$OMP PARALLEL SHARED (sm3dh,sn3dh,sl3dh,sr3dh,se3dh
  :
7502:              !$OMP DO
7503: P------>           do iz=1,lz
7504: |+----->           do iy=1,ly
7505: ||V---->           do ix=3,lx-2
7506: |||     A             sr3dhinv=1.0d0/sr3dh(ix,iy,iz)
7510: |||     A             su3dh(ix,iy,iz)=sr3dhinv*sm3dh(ix,iy,iz)
  :
7519: ||V----            enddo
7520: |+-----            enddo
7521: P------            enddo
7522:              !$OMP END DO NOWAIT
7523:              !$OMP END PARALLEL
```

図 5-5 ループ融合によるメモリアクセス低減前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
200	**13.111**(2.6)	65.557	36092.3	16377.9	99.26	129.0	13.102	0.002	0.005	**0.802**	**6.589**	【チューニング前】

図 5-6 ループ融合によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

コンパイラによる自動ループ融合が行われない場合，ユーザチューニングによるループ融合が効果的である．チューニング後コードを図 5-7 に示す．連続する複数の DO ループは同じループ構造であり，一つの DO ループに修正することでコンパイラが最適化を行う．ループ融合後の配列

sm3dh に注目すると，演算で求めた結果をレジスタに格納しその後の演算ではレジスタ内のデータを参照することになるため，メモリアクセス回数はメモリストアの 1 回になり，メモリアクセス回数を削減することができる．表 5-1 にチューニング前後のメモリアクセス回数の一覧を示す．

```
!チューニング後
7858:                    sdltthalfgy=0.5d0*adltt*sgravy
7859:              !$OMP PARALLEL PRIVATE (ix,iy,iz,sr3dhinv)
7860:              !$OMP DO
7861: P------>           do iz=1,lz
7862: |+----->           do iy=1,ly
7863: ||V---->           do ix=3,lx-2
  :
7868: |||     A             sm3dh(ix,iy,iz)=sm3d(ix,iy,iz)
7869: |||               &             +sdltxinv*( du(ix-1,iy,iz)-du(ix,iy,iz) )
  :
7875: |||                   sm3dh(ix,iy,iz)=sm3dh(ix,iy,iz)
7876: |||               &                 +sdltthalfgx*sr3d(ix,iy,iz)
  :
7881: |||                   sr3dhinv=1.0d0/sr3dh(ix,iy,iz)
7882: |||                   su3dh(ix,iy,iz)=sr3dhinv*sm3dh(ix,iy,iz)
  :
7891: ||V----            enddo
7892: |+-----            enddo
7893: P------            enddo
7894:              !$OMP END DO NOWAIT
7895:              !$OMP END PARALLEL
```

図 5-7 ループ融合によるメモリアクセス低減後のコード

表 5-1 ループ融合によるメモリアクセス低減前後のメモリアクセス命令数

	チューニング前	チューニング後
メモリロード	36	16
メモリストア	17	9

(3) 性能分析

図 5-8 にチューニング前後の性能値を示す．

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU PORT	CONFLICT NETWORK	PROC.NAME
200	**13.111**(2.6)	65.557	36092.3	16377.9	99.26	129.0	13.102	0.002	0.005	0.802	**6.589**	【チューニング前】
200	**8.791**(1.8)	43.953	52951.2	24427.6	99.47	129.0	8.770	0.002	0.003	0.769	**3.745**	【チューニング後】

図 5-8 ループ融合によるメモリアクセス低減前後の FTRACE 情報

メモリアクセス回数が減少することでバンクコンフリクト時間が削減され，実行時間が 13.1 秒から 8.8 秒に短縮することができた．

5.3. 内側ループの展開によるメモリアクセスの削減

(1) チューニング方針

図 5-9 にチューニング前のコードを示す．最内側 n の DO ループの長さ(変数 nmax)は PARAMETER 文で 5 と宣言されており，ベクトル化の対象ループとなるが 121,122 行目の変数 vcx1，vcy2 はベクトル化の阻害要因となるため，コンパイラは最内側 n の DO ループでベクトル化を行わない．一つ外側 ic の DO ループの長さは 4 のため，コンパイラは更に外側 i の DO ループ

でベクトル化を行う. i の DO ループのベクトル化に伴い, ループ分割とループ入れ換えを行う. ループ分割では, i の DO ループのみに依存する演算と i, ic, n の DO ループに依存する演算に分割し, 演算結果を作業配列に格納する. 次に i, ic, n の DO ループの演算では, i の DO ループを内側に移動し事前に求めた作業配列の値を使用してベクトル処理する. このように外側の DO ループをベクトル化することで作業配列を用いたループ分割とループ入れ換えを行うためメモリアクセスの回数が増大する. 図 5-10 の FTRACE 情報では, 実行時間 267 秒の内バンクコンフリクト時間が 196 秒を占めるため, メモリアクセス回数を減らしてバンクコンフリクト時間を短縮することを検討する.

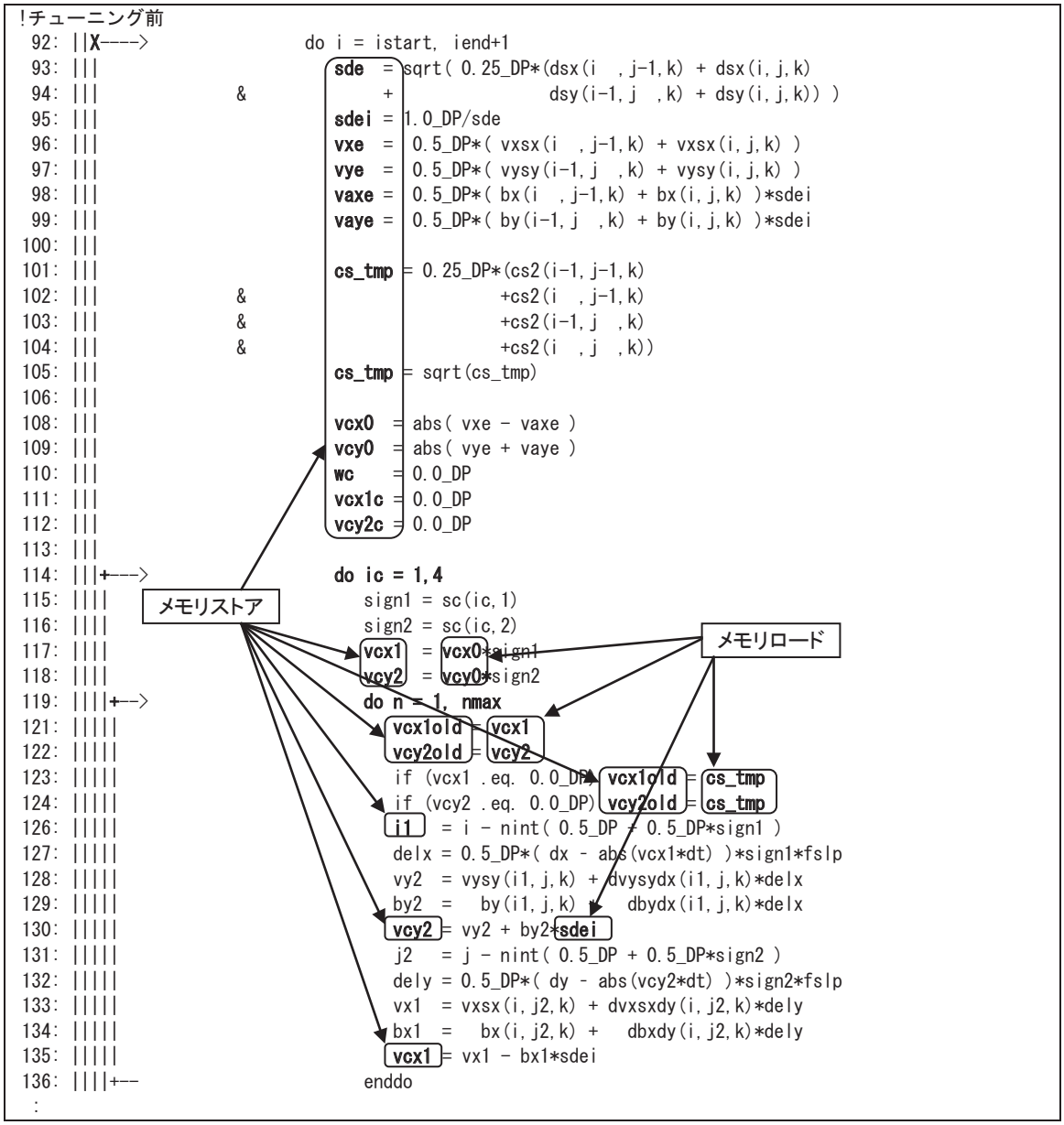


図 5-9　内側ループの展開によるメモリアクセス低減前のコード

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
            TIME[sec](  %)    [msec]                   RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

         50  266.912( 17.9)  5338.239  22400.9  6851.1 99.59 213.8   265.674   0.002   0.002  17.916  178.408 【チューニング前】
```

図 5-10 内側ループの展開によるメモリアクセス低減前の FTRACE 情報

(2) チューニング内容

内側の DO ループの長さがコンパイル時に判明しているため，EXPAND，UNROLL 指示行を挿入しループの展開を行う．内側ループを展開することで単純なベクトルループとなるため，ループ分割とループ入れ換えのための作業配列を使用する必要が無く，レジスタのデータで演算を行うことが可能になる．作業配列を使用しないためメモリアクセスの回数を削減することができる．図 5-11 にチューニング後コード，表 5-2 にチューニング前後のメモリアクセス回数の一覧を示す．

```
!チューニング後
 93: ||V---->               do i = istart, iend+1
 94: |||                       sde  = sqrt( 0.25_DP*(dsx(i  ,j-1,k) + dsx(i,j,k)
 95: |||             &              +              dsy(i-1,j  ,k) + dsy(i,j,k)) )
 96: |||                       sdei = 1.0_DP/sde
 97: |||                       vxe  =  0.5_DP*( vxsx(i  ,j-1,k) + vxsx(i,j,k) )
 98: |||                       vye  =  0.5_DP*( vysy(i-1,j  ,k) + vysy(i,j,k) )
 99: |||                       vaxe =  0.5_DP*( bx(i  ,j-1,k) + bx(i,j,k) )*sdei
100: |||                       vaye =  0.5_DP*( by(i-1,j  ,k) + by(i,j,k) )*sdei
101: |||
102: |||                       cs_tmp = 0.25_DP*(cs2(i-1,j-1,k)
103: |||             &                          +cs2(i  ,j-1,k)
104: |||             &                          +cs2(i-1,j  ,k)
105: |||             &                          +cs2(i  ,j  ,k))
106: |||                       cs_tmp = sqrt(cs_tmp)
107: |||
109: |||                       vcx0  = abs( vxe - vaxe )
110: |||                       vcy0  = abs( vye + vaye )
111: |||                       wc    = 0.0_DP
112: |||                       vcx1c = 0.0_DP
113: |||                       vcy2c = 0.0_DP
114: |||
115: |||         !cdir unroll=4
116: |||*--->                  do ic = 1,4
117: ||||                        sign1 = sc(ic,1)
118: ||||                        sign2 = sc(ic,2)
119: ||||                        vcx1  = vcx0*sign1
120: ||||                        vcy2  = vcy0*sign2
121: ||||        !cdir expand=nmax
122: ||||*-->                   do n = 1, nmax
124: |||||                        vcx1old = vcx1
125: |||||                        vcy2old = vcy2
126: |||||                        if (vcx1 .eq. 0.0_DP) vcx1old = cs_tmp
127: |||||                        if (vcy2 .eq. 0.0_DP) vcy2old = cs_tmp
129: |||||                        i1   = i - nint( 0.5_DP + 0.5_DP*sign1 )
130: |||||                        delx = 0.5_DP*( dx - abs(vcx1*dt) )*sign1*fslp
131: |||||                        vy2  = vysy(i1,j,k) + dvysydx(i1,j,k)*delx
132: |||||                        by2  =   by(i1,j,k) +   dbydx(i1,j,k)*delx
133: |||||                        vcy2 = vy2 + by2*sdei
134: |||||                        j2   = j - nint( 0.5_DP + 0.5_DP*sign2 )
135: |||||                        dely = 0.5_DP*( dy - abs(vcy2*dt) )*sign2*fslp
136: |||||                        vx1  = vxsx(i,j2,k) + dvxsxdy(i,j2,k)*dely
```

```
137: |||||                          bx1  =   bx(i,j2,k) +   dbxdy(i,j2,k)*dely
138: |||||                          vcx1 = vx1 - bx1*sdei
139: ||||*--                      enddo
 :
```

図 5-11　内側ループの展開によるメモリアクセス低減後のコード

表 5-2　内側ループの展開によるメモリアクセス低減前後のメモリアクセス命令数

	チューニング前	チューニング後
メモリロード	471	332
メモリストア	329	4
間接参照	160	0

(3)　性能分析

図 5-12 にチューニング前後の性能値を示す.

```
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS    MFLOPS V.OP  AVER.     VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

      50  266.912( 17.9)  5338.239  22400.9   6851.1 99.59 213.8   265.674   0.002   0.002  17.916  178.408 【チューニング前】
      50   82.086(  8.5)  1641.720  46808.5  22759.7 99.46 214.4    70.789   3.235   0.002   0.086   14.105 【チューニング後】
```

図 5-12　内側ループの展開によるメモリアクセス低減前後の FTRACE 情報

メモリアクセス回数が減少することでバンクコンフリクト時間が短縮され, 実行時間が 266.9 秒から 82.1 秒に高速化することができた.

5.4. 複素数のバンク競合回避

(1)　チューニング方針

図 5-13 に FTRACE 情報を示す. FTRACE 情報では, 実行時間に対してバンクコンフリクトの占める割合が大きいことが分かる. 該当のサブルーチンでは複素数型配列を利用しているために飛びアクセスが発生していると考えられる. このため, 複素数を実数部と虚数部に分け実数型配列に保存し, メモリ上の連続する領域へのアクセスとすることにより, 速度低下やバンクコンフリクトを招く要因となりうる飛びアクセスの削減が期待できる.

```
FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS    MFLOPS V.OP  AVER.     VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                      RATIO V.LEN      TIME    MISS    MISS CPU PORT  NETWORK

   44000 1453.723( 41.5)    33.039  31198.3  13481.2 99.62 255.9 1451.882   0.380   0.709 255.084  821.097 【チューニング前】
```

図 5-13　複素数のバンク競合回避前の FTRACE 情報

```
!チューニング前
   951: V------>            DO 101 JG=1,NXYZ
   952: V------       101   RHO2(JG)=(0.D0,0.D0)
    :
   955:             *VDIR NODEP(RHO1)
   956: V------>            DO 100 IG=1,NXYZ
   957: |                   JG=J2G(IG)
   958: V------       100   RHO1(JG)=P(IG)
    :
   991: V------>          do ig=1,nxyz
   992: |                 VG(ig)=VGG(ig)+Vloc(ig)
   993: V------           enddo
    :
  1009: V------>            DO 300 I=1,NXYZ
  1010: |                   fac=dt*dreal( vg(i) )
  1011: V------       300   RHO2(I)=dcmplx( dcos(fac),-dsin(fac) )*RHO1(I)
    :
  1028:             *VDIR NODEP(RHO2)
  1029: V------>            DO 110 IG=1,NXYZ
  1030: |                   JG=J2G(IG)
  1031: V------       110   P(IG)=RHO2(JG)
```

図 5-14 複素数のバンク競合回避前の前コード

(2) チューニング内容

図 5-14 にチューニング前の複素数型配列を利用したコード, 図 5-15 に最適化後の複素数を実数部と虚数部に分け実数型配列にしたチューニング後コードを示す. チューニング前コードでは倍精度複素数型を使用し配列 RHO1, RHO2 に演算を行っている. この該当箇所では倍精度実数型を使用し実数部は RHO1_R, RHO2_R とし, 虚数部は RHO1_I ,RHO2_I にそれぞれ分け演算を行い処理するようにする.

```
!チューニング後
   953: V------>            DO  JG=1,NXYZ
   954: |                     RHO2_R(JG)=real(0.D0)
   955: |                     RHO2_I(JG)=aimag((0.D0,0.D0))
   956: V------              ENDDO
    :
   968:             *CDIR NODEP
   969: V------>            DO  IG=1,NXYZ
   970: |                     JG=J2G(IG)
   971: |                     RHO1_R(JG)=dble(P(IG))
   972: |                     RHO1_I(JG)=aimag(P(IG))
   973: V------              ENDDO
    :
   984: V------>          do I=1,nxyz
   985: |                   VG_R(I)=VGG(I)+Vloc(I)
   986: |                   fac=dt*VG_R(I)
   987: |                   RHO2_R(I)=(dcos(fac)*RHO1_R(I))-((-dsin(fac))*RHO1_I(I))
   988: |                   RHO2_I(I)=(dcos(fac)*RHO1_I(I))-((dsin(fac))*RHO1_R(I))
   989: V------           enddo
    :
   995:             *CDIR NODEP
   996: V------>            DO  IG=1,NXYZ
   997: |                     JG=J2G(IG)
   998: |                     P(IG)=dcmplx(RHO2_R(JG),RHO2_I(JG))
   999: |                     VG(IG)=dcmplx(VG_R(IG),0.0d0)
  1000: V------              ENDDO
```

図 5-15 複素数のバンク競合回避後のコード

(3) 性能分析

図 5-16 は, チューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC. NAME
44000	**1453.723**(41.5)	33.039	31198.3	13481.2	99.62	255.9	1451.882	0.380	0.709	255.084	821.097	【チューニング前】
44000	**996.058**(34.9)	22.638	45680.0	19675.5	99.63	255.9	993.704	0.676	1.039	234.856	466.970	【チューニング後】

図 5-16 複素数のバンク競合回避前後の FTRACE 情報

メモリ上の連続する領域へのアクセスすることにより, 不連続なメモリアクセスが軽減され, 実行時間が 1,453 秒から 996 秒に短縮することができた.

6. ベクトル演算の効率化の事例

6.1. 総和演算の効率化

(1) チューニング方針

ベクトルループ内の総和演算は一回前の繰り返しの結果を参照するためベクトル化の阻害要因となる. しかし, コンパイラは総和演算を特別なパターンであることを認識し, 専用のベクトル命令(総和型マクロ演算)を用いることによりベクトル化を行う. 図 6-1 はチューニング前のコードである. 配列 Wn は i の次元を持たないため i の DO ループをベクトル化すると, コンパイラは総和型マクロ演算を用いてベクトル化を行う. この総和型マクロ演算により実行を高速化できるが, 総和演算を含む多重ループの場合, ループを入れ替え, 総和型マクロ演算をベクトル演算にすることによりさらなる高速化が可能となる場合がある. このような事例を以下に示す.

```
!チューニング前
11: +------>       do j=1,ny
12: |V----->       do i=1,nx
13: ||
14: ||              Wn(j) = Wn(j) + Wm(i,j)
15: ||
16: |V-----        end do
17: +-------       end do
```

図 6-1 総和演算の効率化前のコード

(2) チューニング内容

図 6-2 にループ入れ換え後のコードを示す. ループ入れ換えにより j の DO ループでベクトル化することで配列 Wm のメモリアクセスがストライドになることに注意されたい.

```
!チューニング後
12: +------>       do i=1,nx
13: |V----->       do j=1,ny
14: ||
15: ||              Wn(j) = Wn(j) + Wm(i,j)
16: ||
17: |V-----        end do
18: +-------       end do
```

図 6-2 総和演算の効率化後のコード

(3) 性能分析

図 6-3 にチューニング前後の性能値を示す.

```
 FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT   PROC.NAME
           TIME[sec](  %)     [msec]                    RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

         1  232.086( 71.8)232086.438  17813.0  8620.3 99.30 173.6  232.083   0.001   0.002    0.000   32.327 【チューニング前】
         1   91.148( 28.2) 91148.138  36054.1 20772.5 99.11 191.0   91.147   0.000   0.001   27.536    4.160 【チューニング後】
```

図 6-3 総和演算の効率化前後の FTRACE 情報

チューニングを行うことで実効性能が向上し, 実行時間が 232.1 秒から 91.1 秒に高速化することができた.

6.2. IF 文の括り出しによる演算数の削減

(1) チューニング方針

ベクトルループ内に IF 文がある場合, SX-9 では IF 文の真偽に関わらず全ての演算を行い最後に IF 文の真にあたる結果のみを採用する方法でベクトル処理を行う. 図 6-4 にチューニング前のコードを示す.

```
!チューニング前
12: +------>       do j=1,my
13: |V----->       do i=1,mx
14: ||               if(itbl(i)==1)then
15: ||                   Wn(i,j) = Wn(i,j) + Wm(i,j)
16: ||               end if
17: |V-----        end do
18: +------        end do
```

図 6-4 IF 文の括り出しによる演算数削減前のコード

(2) チューニング内容

図 6-5 にチューニング後のコードを示す. IF 文の判定式である配列 itbl はベクトル化対象である i の DO ループの次元しか持たないため, IF 文を含めたループ入れ換えを行うことでベクトルループ内に IF 文の処理が無くなる. ベクトル計算機では, ベクトルループ内に IF 文がある場合, 条件式の真偽に関わらず両方の演算を行い, 最後に条件式が真となる場合の結果を選択する. 両方の演算を行うため演算数は増加するが, ベクトル演算を行うことにより性能が向上する. ループ入れ換えを行うことにより, ベクトル演算の効率を低下させることなく演算数を削減し, 実行時間を短縮することができる場合がある. このような高速化の事例を以下に示す. なお, ループの入れ換えにより, 配列 Wn, Wm のメモリアクセスがストライドとなるため, メモリアクセスの観点から性能が低下する可能性があり, 実際に高速化できるかについては確認が必要である.

```
!チューニング後
12: +------>       do i=1,mx
13: |                if(itbl(i)==1)then
14: |V----->       do j=1,my
15: ||                   Wn(i,j) = Wn(i,j) + Wm(i,j)
16: |V-----        end do
17: |                end if
18: +------        end do
```

図 6-5 IF 文の括り出しによる演算数削減後のコード

(3) 性能分析

図 6-6 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	**98.349**(68.2)	98349.331	36691.7	**20319.8**	98.99	191.0	98.348	0.000	0.001	22.750	4.486	【チューニング前】
1	**45.849**(31.8)	45848.683	36026.1	**20756.3**	99.11	191.0	45.847	0.000	0.001	13.683	2.121	【チューニング後】

図 6-6　IF 文の括り出しによる演算数削減前後の FTRACE 情報

ベクトル演算量(実行時間×実効性能)が約 1/2 となるため, 実行時間が 98.3 秒から 45.9 秒に短縮することができた.

6.3. ライブラリの置き換えによる高速化

(1) チューニング方針

SX-9 では, HPC 用に高度に最適化された数学ライブラリ集の Mathkeisan が利用できる. BLAS/LAPACK も最適化されており, プログラムの高速化が容易に行える.

図 6-7 にチューニング前のコードを示す. このコードでは, LAPACK のサブルーチン zheev が呼び出されている. 図 6-8 のチューニング前の性能情報を確認すると, ベクトル化率が 98.17%となっている. zheev は QR 法を用いてエルミート行列の固有値・固有ベクトルを求めるが, 分割統治法でこれらを求めるライブラリ zheevd に置き換えることでベクトル化率の向上と計算時間の短縮を図る.

なお, 分割統治法は QR 法より高速と知られているが, QR 法の演算量が行列の次数の 3 乗のオーダーであるのに対し, 分割統治法の演算量は 2 乗から 3 乗のオーダーとばらつきがあることから, 場合によっては, 有意な性能差がみられないケースもありうる. また, 必要となるメモリ領域の大きさについては, QR 法は次数の 1 乗のオーダーであるのに対し, 分割統治法は 2 乗のオーダーとなる点も考慮する必要がある.

```
!チューニング前
983:        call zheev('V','U',nb*2,v2,nb*2,eig2,work2,2*(nb*2)-1,rwork2,  info)
```

図 6-7 ライブラリ置き換え前のコード

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
4	78.111(5.0)	19527.870	7546.6	4231.4	98.17	210.4	27.766	0.022	23.009	3.657	18.205	【チューニング前】

図 6-8 ライブラリ置き換え前の FTRACE 情報

(2) チューニング内容

図 6-9 にチューニング後のコードを示す. チューニング前のコードで呼び出されていた LAPACK のサブルーチン zheev を zheevd に置き換えている. zheev と zheevd は呼び出しの際の引数が異なり, 作業用の配列も 2 つ増える. 作業配列のサイズは計算前に事前に一度 zheevd を呼び出すことで, 計算に最適なサイズを取得することができる. 取得したサイズの作業配列を確保し, zheevd の呼び出す.

```
!チューニング後
184:                    lwork2= -1
185:                    lrwork2= -1
186:                    liwork2= -1
187:                    call zheevd('V','U',nb*2,v2,nb*2,eig2,work2,lwork2,rwork2,
188:           &             lrwork2,iwork2,liwork2,info)
189:                    lwork2=work2(1)
190:                    lrwork2=rwork2(1)
191:                    liwork2=iwork2(1)
192:                    deallocate (work2,rwork2,iwork2)
193:                    allocate (work2(lwork2),rwork2(lrwork2),iwork2(liwork2))
 :
986:              call zheevd('V','U',nb*2,v2,nb*2,eig2,work2,lwork2,rwork2,
987:           &           lrwork2,iwork2,liwork2,info)
```

図 6-9 ライブラリ置き換え後のコード

(3) 性能分析

図 6-10 はチューニング後の性能情報である. 適切なライブラリを選択することでベクトル化率が 98.17%から 99.38%へ向上し, 78.11 秒から 6.954 秒に短縮することができた. また, zheev から zheevd に置換することによる計算結果への影響もみられなかった.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
4	78.111(5.0)	19527.870	7546.6	4231.4	98.17	210.4	27.766	0.022	23.009	3.657	18.205	【チューニング前】
4	6.954(0.5)	1738.465	53869	31038.0	99.38	221.5	6.729	0.014	0.059	1.026	2.812	【チューニング後】

図 6-10 ライブラリ置き換え前後の FTRACE 情報

ここではライブラリルーチンの置き換えによる高速化の事例を挙げたが, 置換する際には, 置換によるメリット・デメリットを把握した上で置換を行い, 置換後の計算結果の妥当性も必ず確認する必要がある.

6.4. ファイルアクセスの高速化

(1) チューニング方針

図 6-11 に FTRACE 情報を示す. FTRACE 情報では, ベクトル化率が 1.2%となっている. 詳細な解析により, 図 6-12 に DO ループの中でファイルからデータを読み込んだ直後, 読み込んだデータに対して演算が実行されていることが分かった. そこで, ファイルの読み込みと演算を別々に処理するよう最適化を行う.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	219.190(12.5)	219189.905	469.7	4.1	1.20	248.1	0.082	34.416	5.627	0.395	0.057	【チューニング前】

図 6-11 ファイルアクセス高速化前の FTRACE 情報

```
!チューニング前
  2776: +------>          DO 450 IK = 1, NUMK
   :    |
  2783: |                 if ( my_rank.eq.0 ) then
  2784: |+----->           do icpu=0,ncpu
  2785: ||                 nbleng=nend(icpu)-nbegin(icpu)+1
  2786: ||+---->           do 451 ib=1,nbleng
  2787: |||                read(22)( rho1(ig),rho2(ig),ig=1,nxyz )
  2788: |||V--->            do ig=1,nxyz
  2789: ||||                coef0(ig,ib,ik)=dcmplx( rho1(ig),rho2(ig) )
  2790: |||V---             enddo
  2791: ||+----       451   continue
  2792: ||                 if ( icpu.eq.0 ) then
  2793: ||+---->            do ib=1,nbleng
  2794: |||V--->             do ig=1,nxyz
  2795: ||||                 coef(ig,ib,ik)=coef0(ig,ib,ik)
  2796: |||V---              enddo
  2797: ||+----             enddo
  2798: ||                 else
  2799: ||                  call MPI_Send(coef0(1,1,ik),nxyz*nbleng,
  2800: ||                &  MPI_DOUBLE_COMPLEX,icpu,tag,MPI_COMM_WORLD,ierr)
  2801: ||                 endif
  2802: ||
  2803: |+-----            enddo  ! end of icpu loop
  2804: |                 else
  2805: |                  nbleng=nend(my_rank)-nbegin(my_rank)+1
  2806: |                  call MPI_Recv(coef0(1,1,ik),nxyz*nbleng
  2807: |                & ,MPI_DOUBLE_COMPLEX,0,tag,MPI_COMM_WORLD,status,ierr)
  2808: |+----->           do ib=1,nbleng
  2809: ||V---->            do ig=1,nxyz
  2810: |||                 coef(ig,ib,ik)=coef0(ig,ib,ik)
  2811: ||V----             enddo
  2812: |+-----            enddo
  2813: |                 endif
  2814: +------       450 CONTINUE      ! ik roop
```

図 6-12 ファイルアクセスの高速化前のコード

(2) チューニング内容

図 6-12 にチューニング前コード, 図 6-13 にチューニング後コードを示す. チューニング前コードでは read(22) が DO ループの中にあり, その中でさらに演算を行っている. この該当箇所に対して read(22) を DO ループの外へ移動し, 演算に関連する部分の配列を 1 次元配列から 2 次元配列に変更しファイルの読み込みと演算を別々に処理するようにする.

```
!チューニング後
  2776: +------>          DO 450 IK = 1, NUMK
   :    |
  2783: |                 if ( my_rank.eq.0 ) then
  2784: |+----->            do ib=1,nend(ncpu)
  2785: ||                    read(22)(tmp_22(ig,ib),ig=1,nxyz)
  2786: |+-----             enddo
  2787: |                   icnt=0
  2788: |+----->            do icpu=0,ncpu
  2789: ||                    nbleng=nend(icpu)-nbegin(icpu)+1
  2790: ||+---->              do ib=1,nbleng
  2791: |||V--->                do ig=1,nxyz
  2792: ||||                     coef0(ig,ib,ik)=tmp_22(ig,ib+icnt)
  2793: |||V---                 enddo
  2794: ||+----                enddo
  2795: ||                     icnt=icnt+nbleng
  2796: ||                     if ( icpu.eq.0 ) then
  2797: ||+---->                do ib=1,nbleng
  2798: |||V--->                  do ig=1,nxyz
  2799: ||||                       coef(ig,ib,ik)=coef0(ig,ib,ik)
  2800: |||V---                   enddo
  2801: ||+----                  enddo
  2802: ||
  2803: ||                     else
  2804: ||                     call MPI_Send(coef0(1,1,ik),nxyz*nbleng,
  2805: ||                &      MPI_DOUBLE_COMPLEX,icpu,tag,MPI_COMM_WORLD,ierr)
  2806: ||
  2807: ||                     endif
  2808: |+-----              enddo  ! end of icpu loop
   :    |
  2811: |                   else
  2812: |                   nbleng=nend(my_rank)-nbegin(my_rank)+1
  2813: |                   call MPI_Recv(coef0(1,1,ik),nxyz*nbleng,
  2814: |                &    MPI_DOUBLE_COMPLEX,0,tag,MPI_COMM_WORLD,status,ierr)
  2815: |+----->            do ib=1,nbleng
  2816: ||V---->              do ig=1,nxyz
  2817: |||                     coef(ig,ib,ik)=coef0(ig,ib,ik)
  2818: ||V----               enddo
  2819: |+-----              enddo
  2820: |                   endif
  2821: +------       450 CONTINUE      ! ik roop
```

図 6-13 ファイルアクセスの高速化後のコード

(3) 性能分析

図 6-14 にチューニング前後の性能値を示す.

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	PROC.NAME
1	219.190(12.5)	219189.905	469.7	4.1	**1.20**	248.1	0.082	34.416	5.627	0.395	0.057	【チューニング前】
1	0.095(0.0)	94.761	13163.1	0.0	**98.99**	248.0	0.087	0.001	0.001	0.017	0.061	【チューニング後】

図 6-14 ファイルアクセスの高速化前後の FTRACE 情報

ファイルの読み込みと演算を別々に処理することにより, ベクトル化率が 1.2%から 98.99%に改善したことで実行時間が 219.190 秒から 0.095 秒に短縮することができた.

［利用相談室便り］

平成27年度の利用相談について

今年度も５月よりサイバーサイエンスセンター本館利用相談室で利用相談を行っています。日程等詳細は次頁をご覧ください。相談内容によってはメーカ等に問い合わせる場合や、時間を要する場合もありますが、利用者の問題解決にむけて努めております。直接面談のほかに、メールや電話での相談も受けておりますのでお気軽にご相談ください。

・プログラムを高速化するにはどうしたらいいの？
・プログラムを並列化してもっと速く計算したい！
・スパコンでプログラムを動かしても速さがPCと変わらないんだけど、どうして？
・研究室のコンピュータではメモリが足りない！
・研究室の電気代高騰で困っている。
・コンピュータの管理は面倒。研究に専念したい。
・サービスしているアプリケーションを研究室から利用するにはどうすればいいの？

このような、スーパーコンピュータ利用に関する疑問や問題をお持ちの方、これから利用してみたいとお考えの方、一度相談してみてはいかがでしょうか。また、サイバーサイエンスセンター本館相談室には、各種マニュアル、書籍も揃えています。相談室での閲覧、貸し出し（一部の書籍、マニュアルを除く）も可能ですのでご活用ください。

なお、弘前大学、秋田大学、山形大学で行っていた相談業務は、昨年度末をもって終了いたしました。今後、本館利用相談室ですべての相談を受け付けます。

東北大学サイバーサイエンスセンター本館1階 利用相談室

所在地：　仙台市青葉区荒巻字青葉 6-3　（もよりのバス停は「情報科学研究科西」）
Tel　：　022-795-6153　学内内線　92-6153
　　不在の場合は 022-795-3406(3 階共同利用支援係)
e-mail：　sodan05@cc.tohoku.ac.jp
　　e-mail の場合、曜日、時間帯によっては返信に少しお時間をいただくことがあります。あらかじめご了承ください。

サイバーサイエンスセンター
左側新棟、右側本館

利用相談室

平成 27 年度利用相談日程と主な担当分野

曜日・時間		テクニカルアシスタント	主な担当分野
月	2－4時	佐々木大輔（情報基盤課共同研究支援係）	・スーパーコンピュータ ・並列コンピュータ ・Fortran ・大判プリンタ
火	2－4時	齋藤　敦子（情報基盤課共同研究支援係）	・大判プリンタ ・可視化システム ・スーパーコンピュータ ・並列コンピュータ
水	2－4時	中村　公亮（理学研究科）	・アプリケーション（Gaussian）
金	2－4時	小松　一彦（サイバーサイエンスセンター） 森谷　友映（情報基盤課共同研究支援係）	・スーパーコンピュータ ・並列コンピュータ ・高速化（ベクトル化、並列化） ・Fortran　・C/C++ ・大判プリンタ
事前予約		山下　毅　（情報基盤課共同利用支援係）	・アプリケーション全般 ・高速化（ベクトル化、並列化） ・Fortran ・負担金
*上記以外の時間帯に面談・電話での相談を希望の方は、3 階窓口（共同利用支援係）まで相談内容をお申し出ください。センター内担当者に取り次ぎます。			

新テクニカルアシスタントの自己紹介

中村　公亮（なかむら　こうすけ）
東北大学 大学院理学研究科化学専攻　博士課程3年

新しくサイバーサイエンスセンターの利用相談員として、Gaussian、GRRM について担当させていただくことになりました、中村公亮です。担当は、水曜日(14～16 時)です。

Gaussian は著名な計算化学プログラムとして計算化学の研究者のみならず実験化学の研究者の方達にも広く使われておりますが、その使い方は必ずしも簡単とはいえません。必要なインプットなくして十分な計算結果を得ることは難しいのです。また GRRM は本大学で開発された非常に有効な反応解析プログラムですが、まだ使い方をご存じない方も多いでしょう。このような両プログラムについて相談員として、計算方法、出力の解析までご支援をいたします。

１年間どうぞよろしくお願いいたします。

［報 告］

第 116 回サイエンスカフェ「スーパーコンピュータの驚異的な力」の開催報告

東北大学サイバーサイエンスセンター　小林広明

平成 27 年 5 月 29 日金曜日の午後 6 時から仙台メディアテークにおいて，第 116 回サイエンスカフェ「スーパーコンピュータの驚異的な力」が約 100 名の一般市民の皆様をお迎えして開催されました．

サイエンスカフェでは，まず，スーパーコンピュータの高い計算能力が生み出される仕組みから，その能力を活用するシミュレーション技術，そしてその技術が私達の生活にどのように役立っているかについて講演を行いました．スーパーコンピュータが研究者のための研究基盤としてではなく，私達の生活の安全・安心を支える社会基盤として活用されつつある事例として，私達がサイバーサイエンスセンターのスーパーコンピュータを活用して研究開発を行っている「リアルタイム津波浸水被害予測システム」を，南海トラフ大規模地震を想定してのデモを行いながら，紹介しました．また，参加者の皆様に簡単な計算を解いていただき，参加者全員による並列処理を実体験していただきました．

ディスカッションの時間では，大学院生を中心に 10 人程度のグループに分かれて議論が行われ，「半導体技術の限界とその解決法」といった専門的なことからスーパーコンピュータの排熱利用の身近なアイディアなど、幅広い話題で大いに盛り上がりました．

当日の様子は，YouTube　東北大学チャンネル
(https://www.youtube.com/user/tohokuuniversity)で公開される予定です．

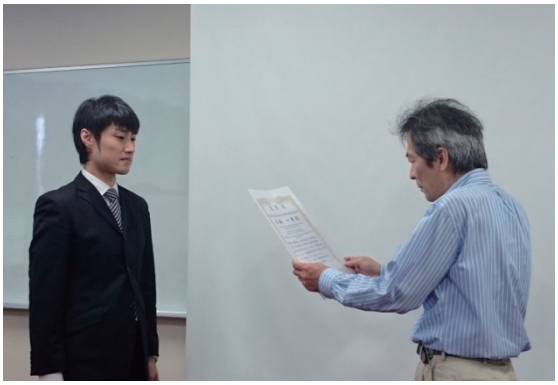
［報 告］

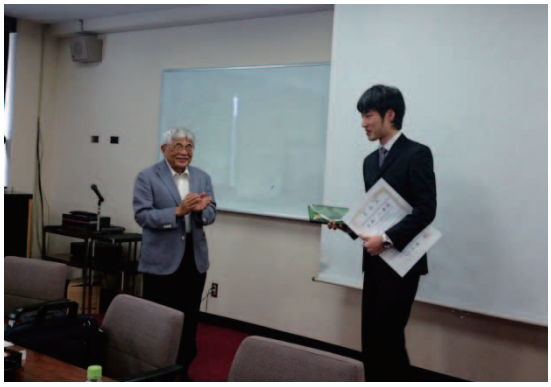
小松助教が情報処理学会東北支部「第 10 回東北支部野口研究奨励賞」を受賞しました

サイバーサイエンスセンターの小松助教が、情報処理学会東北支部「第 10 回東北支部野口研究奨励賞」を受賞しました。

この野口研究奨励賞とは、我が国を代表する情報処理分野のパイオニアであり東北地方の当分野の発展に多大に貢献された野口正一先生から寄贈された資金により設立されたもので、優秀な学術論文を出版した東北支部会員の若手研究者に対して贈られるものです。

学術雑誌に掲載された小松助教の論文が高く評価され、今回の受賞となりました。

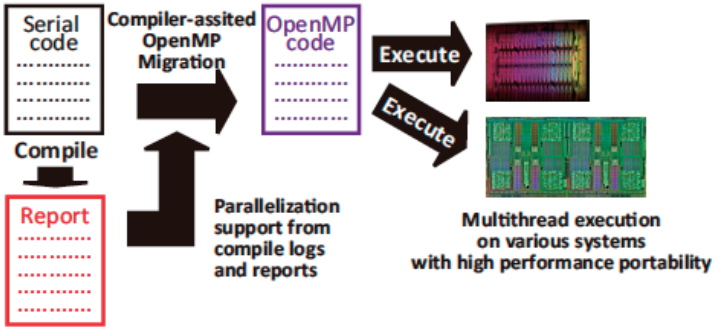




研究の概要および受賞の感想

［研究の概要］

高性能計算(HPC)システムを必要とするアプリケーションは、その性能を最大限に引き出すために、特定の HPC システムを強く意識したコードの開発が行われている。近年多様な HPC システムが登場しており、特定の HPC システムだけでなく様々な HPC システムにおいても高い性能を引き出すことができるコードが強く求められている。しかしながら、そのようなコードの最適化には大幅な修正が必要となっている。



今回受賞対象となった論文では、コンパイル情報を活用して、様々な HPC システムにおいても性能を引き出すことができる OpenMP 並列コードを容易に開発する手法を提案している。

特定の HPC システムを意識したコードのコンパイル情報が、他の HPC システムにおいても有用であることに着目し、提案する手法ではコンパイラの並列化情報から並列化可能な場所を特定している。これにより、並列化の際のプログラマの負担を抑えつつ、他の HPC システムにおいても効果的な並列コードを開発することができる。 実験により、提案手法による並列コードが様々な HPC システムにおいて高い性能を達成できることを明らかにしている。

[受賞の感想]

この度、野口研究奨励賞という大変名誉ある賞を受賞することができ、野口正一先生ならびに情報処理学会東北支部の皆様に深く感謝申し上げます。また、本研究を遂行するにあたり、小林広明先生、滝沢寛之先生、江川隆輔先生には、多大なるご指導をいただきました。厚く御礼申し上げます。今後も情報処理分野の発展のため、尽力してまいります。

［展示室便り⑭］

スーパーコンピュータ SX-9

今回は、日本電気（株）製のスーパーコンピュータ SX-9 です。この計算機は展示室便り⑥で紹介した「スーパーコンピュータ SX シリーズ」の後継機であり、ベクトル並列型のスーパーコンピュータです。16 ノードが 2008 年に導入、2 ノードが 2010 年に追加され、2015 年まで稼働しました。

SX-9 の 1 ノードは 16 個の CPU と 1TB のメモリで構成されており、全システムの総演算性能は 29,491GFLOPS、総メモリ容量は 18TB でした。図 1「CPU の内部構成」は CPU の構成を表しています。CPU はベクトルユニット部、スカラユニット部により構成されています。スカラユニットは命令の解読、ベクトルユニットへのベクトル命令の供給・起動およびスカラ命令の実行を行います。ベクトルユニットは 8 セットのベクトルパイプラインを備え、各ベクトルパイプラインは乗算器×2、加算／シフト演算器×2 および除算／平方根演算器×1、論理演算器×1 の 6 種類のそれぞれ独立に動作可能な演算パイプライン、マスク演算パイプライン、ロード／ストアパイプライン、マスクレジスタおよびベクトルレジスタにより構成されています。オペレーティングシステムは UNIX 系の SUPER-UX です。利用者はネットワークを経由し会話処理で直接計算機を利用することができました。また、長時間を要する計算は、バッチ処理で実行されました。

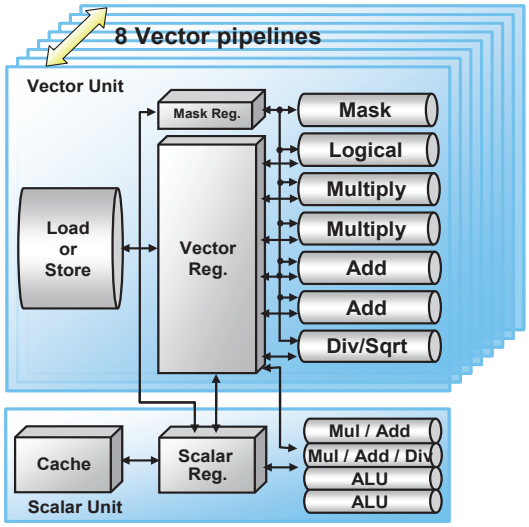


図 1　CPU の内部構成

展示品 1　SX-9 の正面と側面

展示室には SX-9 の 1 ノード筐体が展示されています。展示品 1 はその正面部と側面から見た CPU、メモリ、電源部が組み込まれているところです。ここでは、筐体内での CPU、メモリ、

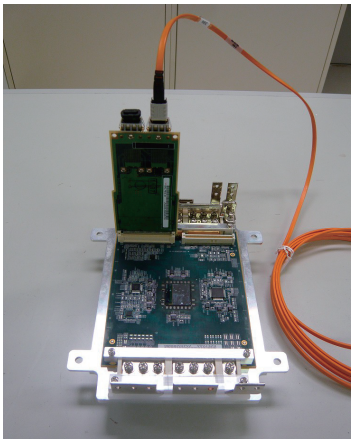
各種のケーブルが実装されていた状態を見ることが出来ます。展示品 2 のミニモックアップは 16 ノードの全景を現しています。4 ノードを 1 列にまとめ、新幹線のような形を構成しています。

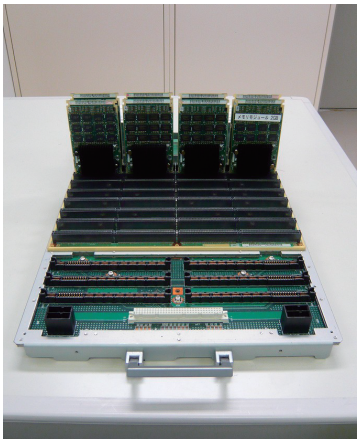
展示品 2　SX-9 筐体のミニモックアップ

写真 1　計算機室の SX-9

計算機室には写真 1 のように設置されていました。

展示品 3 は 1 つの CPU ボードとメモリーボードです。CPU ボードに付いているオレンジ色のケーブルは光ファイバーです。全ての CPU ボードは光ファイバーにより専用の超高速のノード間接続装置(Inter-node Crossbar Switch、IXS)に接続され、IXS を経由して CPU ボード間のデータ通信がされました。MPI プログラムでのノード間を跨るプロセス間通信はこのような機構で送受信が行われました。





展示品 3　SX-9 の CPU ボード（左）とメモリボード(右)

表 1「スーパーコンピュータの変遷」は、センターで導入した計算機の約 30 年間の演算処理能力とメモリ容量を表したものです。30 年間にわたる、スーパーコンピュータの驚異的な性能向上がお分かりいただけると思います。なお、SX-9 によるサービスは 2008 年から 2015 年と、他の計算機よりも長い期間行われました。

表 1　スーパーコンピュータの変遷

機種	演算性能 GFLOPS	メモリ容量 GB
SX-1　1968-1988 年	0.6	0.1
SX-2N　1989-1993 年	1.1	0.3
SX-3R　1994-1997 年	25.6	4.0
SX-4　1998-2002 年	256.0	32.0
SX-7　2003-2007 年	2,119.0	1,920.0
SX-9　2008-2015 年	29,491.0	18,432.0

センターでは、設立以来最先端の計算機を導入し共同利用施設として研究者への研究支援を行っております。また、導入した計算機の能力を研究に大いに活用してもらうため、センター職員と利用者による研究活動にも取り組んでいます。一

つは展示室便り⑥で紹介した「高速化推進研究活動報告」です。報告書では、プログラムの高速化技法や利用者プログラムの高速化事例が紹介されています。また、共同研究も行っており、その成果はこの広報誌 SENAC に掲載されています。これらの内容から、センターでの計算機サービスは様々な研究のため大いに活用されていること知ることができます。最後にそのタイトルをいくつか紹介いたします。

・降着円盤中での磁気乱流生成過程に関する計算機実験(SENAC　Vol.46,No.4)
・数GHzの周波数帯における負の透磁率を示す構造の開発とその広帯域化に関する研究(SENAC Vol46,No4)
・プラズモニック構造体による光エネルギー利用の効率化(SENAC　Vol.46,No.4)
・気候モデルデータのダウンスケーリングによるヤマセの将来変化(SENAC　Vol.47,No.2)
・民間航空機開発における大規模空力弾性解析シミュレーションの適用(SENAC　Vol.47,No.2)
・航空機エンジン排気ジェットと後流渦の相互作用の解析(SENAC　Vol.47,No.3)
・起電力法を用いた低姿勢な大規模リフレクトアレーの設計(SENAC　Vol.47,No.4)
・東北地震に伴う固有地震活動の揺らぎから推測された摩擦特性と余効すべり伝播過程(SENAC Vol48,No1)

「高速化推進研究活動報告」、「広報誌SENAC」は以下URLよりご覧いただけます。
http://www.ss.cc.tohoku.ac.jp/report/speed-up.html

[Web 版大規模科学計算システムニュースより]

大規模科学計算システムニュースに掲載された記事の一部を転載しています。　http://www.ss.cc.tohoku.ac.jp/tayori/

利用負担金の表示コマンドについて　(No.197)

本センター大規模科学計算システムでは、利用者の利用額とプロジェクトごとに集計した負担額、請求情報を表示するためのコマンドとして ukakin, pkakin があります。これらのコマンドは、並列コンピュータ(front.cc.tohoku.ac.jp)にログインして使用します。

コマンド名	機　　能
ukakin	利用者ごとの利用額を各システム、月ごとに表示
pkakin	プロジェクトごとに集計した負担額、請求情報を表示

いずれも、前日までご利用いただいた金額を表示します。コマンド使用例は大規模科学計算システムウェブページをご覧ください。

負担金の確認
http://www.ss.cc.tohoku.ac.jp/utilize/academic.html#負担金の確認

(共同利用支援係)

— SENAC 執筆要項 —

１．お寄せいただきたい投稿内容

次のような内容の投稿のうち、当センターで適当と判定したものを掲載します。その際に原稿の修正をお願いすることもありますのであらかじめご了承ください。

・一般利用者の方々が関心をもたれる事項に関する論説
・センターの計算機を利用して行った研究論文の概要
・プログラミングの実例と解説
・センターに対する意見、要望
・利用者相互の情報交換

２．執筆にあたってご注意いただく事項

(1)原稿は横書きです。
(2)術語以外は、「常用漢字」を用い、かなは「現代かなづかい」を用いるものとします。
(3)学術あるいは技術に関する原稿の場合､200 字～400 字程度のアブストラクトをつけてください。
(4)参考文献は通し番号を付し末尾に一括記載し、本文中の該当箇所に引用番号を記入ください。
　・雑誌：著者, タイトル, 雑誌名, 巻, 号, ページ, 発行年
　・書籍：著者, 書名, ページ, 発行所, 発行年

３．原稿の提出方法

原稿のファイル形式は Word を標準としますが、PDF での提出も可能です。サイズ*は以下を参照してください。ファイルは電子メールで提出してください。

－Word の場合－
　・用紙サイズ：A4
　・余白：上＝30㎜　下＝25㎜　左右＝25㎜　綴じ代＝0
　・標準の文字数（45 文字 47 行）

＜文字サイズ等の目安＞
　・表題＝ゴシック体 14pt 中央　・副題＝明朝体 12pt 中央
　・氏名＝明朝体 10.5pt 中央
　・所属＝明朝体 10.5pt 中央
　・本文＝明朝体 10.5pt
　・章・見出し番号＝ゴシック体 11pt～12pt
　　*余白サイズ、文字数、文字サイズは目安とお考えください。

４．その他

(1)執筆者には、希望により本誌*（10 部以内の希望部数）と本誌 PDF 版を進呈します。
　*2014 年末で、別刷の進呈は終了しました。
(2)投稿予定の原稿が 15 ページを超す場合は共同利用支援係まで前もってご連絡ください。
(3)初回の校正は、執筆者が行って、誤植の防止をはかるものとします。
(4)原稿の提出先は次のとおりです。

東北大学サイバーサイエンスセンター内　情報部情報基盤課共同利用支援係
e-mail　uketuke@cc.tohoku.ac.jp
TEL　　022-795-3406

スタッフ便り

東北大学と NEC の産学連携拠点として「高性能計算技術開発（NEC）共同研究部門」が、昨年 7 月センター内に開設されました。NEC 社員である私も本研究部門の一員としてスーパーコンピュータによる科学技術計算のさらなる高速化・高精度化を目指し、研究に取り組んでいます。

スーパーコンピュータの演算性能向上は、この 30 年で 10 億倍です。この熾烈な研究開発競争をリードするためには、先端技術研究からものづくりである量産性検討までを同時に行っていく必要があります。共同研究部門には産学のハードウェア、ソフトウェア、アプリケーションの様々な分野の研究者・技術者が揃っており、協調設計により東北大学発の先端スーパーコンピュータを生み出していきたいと考えています。

話は変わりますが、私は東北大学大学院情報科学研究科の卒業生です。大学を卒業してからちょうど 10 年、卒業時は再度東北大学の所属になるとは考えてもいませんでした。今、企業でスーパーコンピュータのエンジニアとして働いていることの基礎は東北大学で学んだものであり、自分を育ててくれた大学に共同研究の成果という形で恩返しができるよう、引き続き高い目標のもと研究に取り組んでいきたいと考えています。（S. M）

この号が皆様のお手元に届いている頃は、梅雨も明け暑い夏到来、といったところでしょうか。夏といえば、東北大学オープンキャンパス。今年も 7/29 と 7/30 の両日開催されます。最新鋭のスーパーコンピュータや、学内ネットワークシステム、そしてこれらの機器を活用した研究成果を公開します。飛行機、自動車などの"ものづくり"から気象・災害予測など、実は身近なところでスパコンは役に立っているんです。普段は、なかなか見ることができないスパコンの迫力を、間近で体感いただければ幸いです。

今回タイミングを逃してしまっても、見学は随時受け付けています。（K. O）

整備中の青葉山新キャンパス

SENAC 編集部会

小林広明　曽根秀昭　水木敬明　後藤英昭
江川隆輔　佐藤恵美子　高杉佳奈　大泉健治
小野　敏　斉藤くみ子

平成 27 年 7 月発行
編集・発行　東北大学
サイバーサイエンスセンター
仙台市青葉区荒巻字青葉 6-3
郵便番号　980-8578
印　刷　東北大学生活協同組合
プリントコープ

システム一覧

計算機システム	機　種
スーパーコンピュータ	SX-ACE
並列コンピュータ	LX 406Re-2

サーバ名

フロントエンドサーバ	front.cc.tohoku.ac.jp
SSH アクセス認証鍵生成サーバ	key.cc.tohoku.ac.jp

サービス時間

利用システム名等	利用時間帯
スーパーコンピュータ	連　続　運　転
並列コンピュータ	連　続　運　転
可視化機器室	平日 9:00～21:00
館 内 利 用	平日 8:30～21:00

スーパーコンピュータ（SX-ACE）の利用形態と制限値

利用形態	利用ノード数 ※1	実行時間制限 （経過時間）	メモリサイズ制限	-q オプション	-b オプション
通常	1～256	規定値：2 週間 最大値：1 ヶ月	60GB×ノード数	sx	利用ノード数
	257～1,024	規定値：1 ヶ月 最大値：1 ヶ月			
無料	1	1 時間	60GB		f
デバッグ	1～16	2 時間	60GB×ノード数	debug	利用ノード数
	17～32	24 時間			

並列コンピュータ（LX 406Re-2）の利用形態と制限値

利用形態	利用ノード数 ※1	実行時間制限 （経過時間）	メモリサイズ制限	-q オプション	-b オプション
通常	1～24	規定値：1 ヶ月 最大値：1 ヶ月	128GB×ノード数	lx	利用ノード数
アプリケーション	1	なし	128GB		a
会話型	1（6 コアまで）	1 時間 （CPU 時間合計）	8GB	-	-

※ 1. 2ノード以上を利用した並列実行にはMPIの利用が必用

目次

東北大学サイバーサイエンスセンター
大規模科学計算システム広報　Vol.48 No.3 2015–7