GRIDコンピューティング環境における行列ライブラリ向け性能保証方式の検討

直野 健[†] 今村俊幸[††,☆] 恵木正史[†]

GRIDコンピューティング環境における行列ライブラリでは，課金に見合った高性能なライブラリ機能提供を求められるため，性能保証が重要な課題となる．この課題に対し，本報告では，金融工学で利用されることの多い下方リスクの概念を適用し，行列ライブラリ自体の性能保証値を算出するアルゴリズムを考案した．本アルゴリズムの妥当性を検証する初期実験として，日立SR8000/F1上で，3000次元から4000次元の固有値計算ループにおける性能保証を試みた．算出した性能保証値と1次元ごとの性能実測値を比較評価した結果，97.725%の確率で性能保証が実現できていることを確認した．

Research on a Performance Assurance Method for Matrix Libraries on the Grid

Ken Naono,[†] Toshiyuki Imamura[††,☆] and Masashi Egi[†]

Performance assurance for the Grid computing environment is a key factor in determining a charge for high performance library functions. For performance assurance of the matrix libraries, the concept of a down-side risk, which is generally used in financial engineering, is introduced to investigate the algorithm which computes a performance assurance value. Preliminary experiments to verify the validity of the algorithm is performed in the case of eigenvalue loops with the dimension of 3000–4000 on the Hitachi SR8000/F1. The results show that the performance value derived from the algorithm satisfies an assurance level with 97.725% probability from comparisons with performances actually measured in every dimension.

1. はじめに

近年，スーパーコンピューティングの利用環境は，スーパーコン単体の利用から，複数を連携させて利用するGRIDコンピューティングへとパラダイムが変わりつつある．GRIDコンピューティングによって，あるサイトのスーパーコンには存在しない行列ライブラリ機能を，別サイトのスーパーコンの行列ライブラリ機能によって補完し，より多くの行列ライブラリ機能をユーザに提供することも可能になってくる．

従来，行列ライブラリは，スーパーコン単体の性能を引き出すミドルウェアとして開発されてきた．しかし，上記の流れを受けて，この数年の間，産総研のNinf[1)]，テネシー大のNetSolve[2)]，SANS[3)]など，ネットワーク型のライブラリが研究されるようになっている．これらの研究では，手元のPCからどこのスーパーコンかを意識せずに利用可能なコンピューティング環境が提案されており，GRIDコンピューティングの発展に寄与してきた．

しかし，GRIDコンピューティング環境における行列ライブラリの実用化という観点では，ネットワーク型ライブラリのみでは，現在のスーパーコンセンタで稼動している一般的な行列計算機能の提供までサポートすることができない．その理由は主に次の2点が考えられる．1つは，スーパーコンのアーキテクチャが多種類かつ複雑なため，各コンピュータの性能を引き出すチューニングが困難になっており，多くの行列ライブラリ機能の提供が困難となっている点である．もう1つは，プログラムの性能値が，パラメータの少しの差異に敏感になるケースがあるため，性能保証が容易ではない点である．性能保証がない場合，課金に見合った高性能な行列ライブラリ機能を提供できない．

† 株式会社日立製作所中央研究所
Central Research Laboratory, Hitachi, Ltd.
†† 日本原子力研究所
Japan Atomic Energy Research Institute
☆ 現在，電気通信大学電気通信学部情報工学科
Presently with Department of Computer Science, The University of Electro-Communications

上記の1つ目の問題点に対しては，LAPACKを自動チューニングするテネシー大学のATLAS[4]，並列ライブラリを自動チューニングする東京大学のI-LIB[5),6)]，電気通信大学のABC-LIB[7)]などの研究プロジェクトによって解決が試みられている．ところが，2つ目の課題に対してはこれまでほとんど検討されておらず，詳細な数値実験[8),9)]において，ようやく課題が明確になってきたところである．この研究では，単体サーバ向けの高性能化方式であるループアンローリングが，行列サイズによって性能のばらつきを増大させる傾向を明らかにした．今村ら[9)]ではその対策として，性能安定化方式を提案しているが，一方で，性能保証の必要性を認識するに至った．

そこで，本研究では，GRIDコンピューティングにおいて重要な要求事項と予想される，性能保証方式を確立する第1段階として，GRIDコンピューティング環境における行列ライブラリの性能保証値算出アルゴリズムを検討する．なお，GRIDコンピューティング環境下での性能に関しては，ネットワークの品質（QoS）を鑑みた資源配分方法[10)]，あるいはスケジューリングアルゴリズム[11),12)]などが検討されているが，本報告では特定の計算サーバ内部に実装されるライブラリ自体の性能保証について検討する．

2. GRIDコンピューティング環境における行列ライブラリの課題

2.1 GRIDコンピューティング環境における行列ライブラリ利用例とその課題

GRIDコンピューティング環境では，ユーザが，インターネットを介して個々のスーパーコンにチューニングされて実装されている様々な行列ライブラリを呼び出し，利用することが可能である．本報告ではこのような行列ライブラリ利用環境をライブラリポータルと呼ぶことにする．

図1にライブラリポータルの利用例を示す．この例では，クライアントPCにあるユーザプログラムの中で処理時間が長い行列計算部分の処理を，ポータルサーバに実装されている行列ライブラリを呼び出し，高速実行する．ライブラリポータルによって，ユーザは，行列ライブラリが組み込まれているサーバに自分のプログラムをインストールすることなく，適切な行列ライブラリを選択し，利用できる．

このような計算環境を実現するうえで，次の4つの課題があげられる．

- **機能メニュー整備**
  ライブラリポータルには，実対称密行列用の固有値計算ソルバ，あるいは疎行列用の反復解法を実装した連立一次方程式ソルバなどの多種多様な機能サポートが必要である．従来から非常に多くのアルゴリズム開発，行列ライブラリ実装の研究がなされ，現在は幅広いユーザニーズに応えられるようになってきている．しかし，疎行列解法，反復法などの一部で未開発の機能も残っており，さらに開発されたプログラムが様々なプラットフォーム上で高速に稼動するようチューニングされる必要がある．
- **機能メニュー選択のインタフェース**
  ライブラリポータルには，ユーザがある機能を利用する際，どのポータルサーバにはどの行列ライブラリがあるかが容易に分かるインタフェースが必要である．
- **性能保証**
  ライブラリポータルは，単に速いというだけではなく，課金に見合った性能で，期待した時間以内に答えを返す必要がある．これまでの行列ライブラリの研究では，高速化の研究が長い間行われてきたが，報告者らの知る限り，限られたパラメータセットでのピンポイント的な性能向上を実現したものであって，性能保証の観点はほとんど研究されていない．
- **精度保証**
  ライブラリポータルは，ユーザの期待する精度で答えを返さなければならない．現在，精度保証計算の研究[13)]がさかんであり，その流れから，ライブラリ実装が始まったところであるが，精度保証された行列ライブラリ機能は少ない．

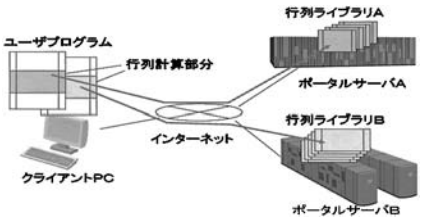

図1 ライブラリポータルの例
Fig. 1 Example of matrix library portal.

2.2 性能保証の課題

上記のとおり，ライブラリポータルには4つの課題があるが，以下，これまでほとんど検討されていなかったと考えられる性能保証の課題に関して検討する．

性能保証をするには，処理を実行する前に性能を見積もることになる．見積りは，おおよそのポイントでの実測性能を連続的につないで得られる数値と設定す

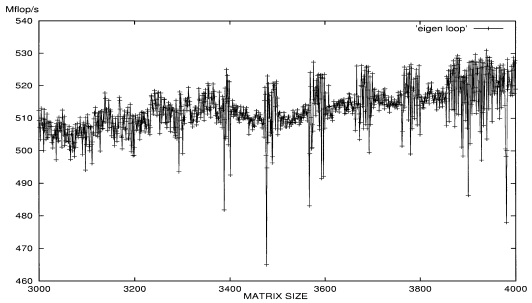

図 2 一部の実効性能が劣化する例（日立 SR8000/F1 の 1CPU，固有値ループ）
Fig. 2 Example of performance degradation (Eigensolve loop, on Hitachi SR8000/F1, 1CPU).

ることが多い．しかし，実際，詳細に性能を評価すると，不安定な実効性能となるケースがある．例として**図 2** に，日立 SR8000 での固有値ループの性能を示す．これは，3000–4000 次元で実測した結果である．この次元の区間は，性能がほぼ一定になると予想されていたが，実際には，性能のばらつきがあり，一部の次元では，1 割程度も性能が劣化している[8]．

性能保証では，このように一部の性能が劣化するようなケースについて考慮しなければならない．上記の性能不安定性は，キャッシュ競合によって引き起こされることが分かっている．これに対して，キャッシュのアライメントをもとに安定させる方法[9] もあるが，必ずしも安定化方法が存在するとも限らず，さらに，現在のサーバアーキテクチャではキャッシュがより多層になり，かつ複数 CPU でキャッシュを共有するケースもあるため，プログラム処理性能が不安定になる傾向が強まっている．

このようにライブラリポータルでは，処理性能のブレをあらかじめ考慮に入れた性能を見積もり，平均的な性能からそのブレ幅を差し引いた性能保証値を算出することが課題となる．

3. 行列ライブラリ向けの性能保証値算出アルゴリズム

3.1 性能保証に対するリスク管理の概念の導入

本研究では，性能保証に対しリスク管理の概念を導入する．これは，確率密度分布での下方の領域をリスクと見なすリスク管理の考え方である．リスクの指標として，本報告では，金融リスク管理で現在最もよく利用されている Value-at-Risk（バリュー・アット・リスク，以下 VaR）[14] を適用する．

図 3 に VaR の例を示す．図 3 では，金融機関の収益予測を確率密度分布で表現している．金融機関がさ

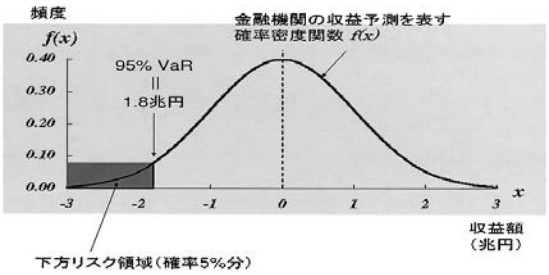

図 3 95%の VaR，および確率 5%の下方リスク領域
Fig. 3 95% VaR, 5% downside risk area.

らされているリスクを，たとえば，最悪 5%のケースを除いた範囲で起きうる最大損失額とし，これを 95%の VaR という．たとえば 95%の VaR が 1.8 兆円であったとすると，金融機関はその 1.8 兆円の自己資本を用意しておき，リスクに備えるといった施策をとっている．図 3 の濃いアミの領域で $f(x)$ の下側は，下方リスク領域と呼ばれる．

本研究では，ライブラリ自体の性能保証に対しこの考えを適用する．行列サイズなど，何らかの入力パラメータによって，実効性能がばらつき，金融機関の収益額のように何らかの分布を持つとする．そこで，性能保証値=VaR，たとえば「95%の性能保証値=95%の VaR」とする．すなわち，5%で起きる最悪のケースは除外し，残りの 95%のケースの中で最低の性能値を「95%の性能保証値」と定義する．

3.2 性能保証値算出アルゴリズム

VaR を計測するには，一般に，分布をできる限り正確に把握する必要がある．性能に影響を及ぼすパラメータがとるすべての値において性能を計測すれば，最も正確に分布が特定できる．しかし，一般に莫大な時間を要するため現実的ではない．そこで，分布の形状に仮定を置き，ランダムにサンプリングしたパラメータにおける性能を計測する．

分布の形状は性能測定対象に応じて種々検討すべきであるが，今回の研究の範囲では，アルゴリズムの初期検討という位置付けのため，(1) 性能の分布の平均がほぼ一定である領域を対象とする，(2) 性能の分布は正規分布に従うとする，という条件の下，性能を左右するパラメータとして行列次元（行列のサイズ）をとりあげ，性能保証値を算出するアルゴリズムを考案した．なお，正規分布としたのは，パラメータの取扱いの容易さと，報告 8) における評価データでは，性能分布の形状が比較的，正規分布に近かったためである（80%分の中心値データに対し，有意水準 10%の尖度，歪度の検定によって正規分布であるという仮説は棄却されなかった）．

1) パラメータ空間からランダムに L 個の数値を選択し，そのパラメータにおける性能を実測する．たとえば，3000 次元から 4000 次元の 1001 個のパラメータから，L=10 個のポイントをランダムに選択し，各次元での性能を測定する．
2) L 個の性能実測値の平均 M_L と標準偏差 σL を算出する．
3) 真の平均 M，真の標準偏差 σ との誤差項 ΔM_L，$\Delta\sigma_L$ に対し

$$\Delta M_L - 2\Delta\sigma_L > -q_L\sigma_L \quad (3)$$

を満たす q_L を定める．
4) 性能保証値 $VaR(2\sigma)(97.725\%)$ を $M_L - (2 + q_L)\sigma_L$ とする．理由は下記のとおり．

$$\begin{aligned} &VaR(2\sigma) \\ &= M - 2\sigma \\ &= (M_L + \Delta M_L) - 2(\sigma_L + \Delta\sigma_L) \\ &= (M_L - 2\sigma_L) + (\Delta M_L - 2\Delta\sigma_L) \\ &> M_L - (2 + q_L)\sigma_L \end{aligned}$$

図 4 性能保証算出アルゴリズム
Fig. 4 Algorithm of performance assurance value calculation.

正規分布は，サンプル点の実測値から，平均 M と標準偏差 σ を計算すれば分布が特定できるが，ここでは，2σ の VaR（=97.725%の性能保証値）を得る場合のみに限定して，**図 4** に未定数値 (3) の q_L) 付きの性能保証値算出アルゴリズムを示す．

未定数値 q_L の意味と具体的な定め方について以下，説明する．q_L とは，近似平均 M_L（近似標準偏差 σ_L）と真の平均 M（標準偏差 σ）との誤差 ΔM_L（誤差 $\Delta\sigma_L$）が，σ_L に対しどの程度の大きさになるかの指標である．図 4 の式 (1) を満たす q_L は，

$$-\Delta M_L + 2 \times \Delta\sigma_L < q_L \times \sigma_L \quad (1)$$

を満たす q_L であり，式 (2) をいつも満足する q_L を求めるには式 (2) の左辺が最大になり，かつ σ_L が最小の値をとっても，式 (2) を満たすよう q_L を求めることである．ここで式 (2) の左辺が最大になるのは，ΔM_L が負であり，かつ $\Delta\sigma_L$ が正の値の場合である．しかし，ΔM_L の最小値（負の値で，絶対値が最大）や $\Delta\sigma_L$ の最大値を実際に求めることはできない．

そこで，M_L と σ_L を複数のサンプルで計測し，M_L の標準偏差 $\omega(M_L)$，σ_L の標準偏差 $\omega(\sigma_L)$ を求め，ΔM_L を $3\omega(M_L)$ で，$\Delta\sigma_L$ を $3\omega(\sigma_L)$ で代用し，式 (2) の評価の代わりに

$$3 \times (\omega(M_L) + 2\omega(\sigma_L)) < q_L \times min(\sigma_L) \quad (2)$$

を満たす q_L を定めることにする．ここで「3 × 標準偏差分」にしたのは，性能保証値が低くなりすぎない程度であって，かつ，$VaR(2\sigma)$ での「2 × 標準偏差分」よりも詳細なレベルにしたためである．

4. 性能保証値算出アルゴリズムに対する初期評価実験

4.1 評価実験対象のプログラム

評価実験対象は，下記の固有値計算における 3 重ループである．これは，3 重対角化（リダクション演算）と呼ばれる部分である．

```
do M = 1, MB
 do J = 1, N
  do I = 1, J
    A(I, J) = A(I, J)
    - ( V(I, M)*U(J, M) + U(I, M)*V(J,M) )
  enddo
 enddo
enddo
```

ここで N は行列の次元である．また，MB はブロック幅であり，10〜40 程度の定数である．この 3 重ループは，アンロール段数によって性能不安定の度合いが異なると経験的に知られているため，3 種類のアンロールで評価する．アンロールは，外側から段数を表記することとし，たとえば (3-2-2) は，上記 M のループを 3 段，J のループを 2 段，I のループ 2 段とアンロールする展開を表す．実験対象のアンロール段数の組合せは，(1-1-1)，(3-3-2)，(5-5-2) の 3 種類とする．SR8000 でのソースコードの例として，(3-3-2) を**図 5** に示す．この例では，外側 2 段の展開は実際にコーディングを行い，最内側では*soption unroll(2) という指示文によってアンローリングを行う．

4.2 実際の性能値による未定係数値の決定

ここでは，(1-1-1) のアンローリングに限定し，SR8000/F1 の 1CPU での N=3000〜N=4000 の範囲でランダムサンプリングした実測性能を用い，図 4 の 3) における未定係数 q_L について決定する．

3000〜4000 次元におけるランダムサンプリングは以下のように実施した．まず，長さ 2000 の乱数パス（乱数列）を異なる種から 20 本生成した．この際，乱数生成には SR8000 の行列ライブラリ MATRIX/MPP の HSRUIM（合同乗算法による一様乱数）を利用し，乱数の種は，$(11 \times (J \times 2) + 19;\ J = 1, 2, \cdots, 20)$ とした．

次に，各乱数パスの 1001 番目から連続する 10 個，20 個，40 個，80 個，160 個の乱数値をサンプリングした．たとえばある乱数パスにおける 10 個の乱数値は，3021，3870，3678，3067，3504，3386，3489，3163，3908，3309 である．これに対する当該パス（10 個分）の近似平均，近似標準偏差は，3021 次元の性

```
      subroutine tunecore332(
     &            i$1, i$5, i$3, m0_mod6, m0, i$nnod, i$inod, l1$a, u, v,  nm
     &    )
C
      implicit double precision (a-h,o-z),integer(i-n)
      double precision l1$a(nm,*)
      double precision u(nm,*)
      double precision v(nm,*)
C
*voption vec
*soption nolooproreroll
        do i$1=i$5,i$3,3
*soption nolooproreroll
          do i0=m0_mod6+1,m0,3
            j$0=(i$1-1+0)*i$nnod+i$inod;   j$1=j$0+i$nnod;   j$2=j$1+i$nnod
*
            u0_0 = u(j$0,i0+0);     u1_0 = u(j$1,i0+0);     u2_0 = u(j$2,i0+0)
            u0_1 = u(j$0,i0+1);     u1_1 = u(j$1,i0+1);     u2_1 = u(j$2,i0+1)
            u0_2 = u(j$0,i0+2);     u1_2 = u(j$1,i0+2);     u2_2 = u(j$2,i0+2)
*
            v0_0 = v(j$0,i0+0);     v1_0 = v(j$1,i0+0);     v2_0 = v(j$2,i0+0)
            v0_1 = v(j$0,i0+1);     v1_1 = v(j$1,i0+1);     v2_1 = v(j$2,i0+1)
            v0_2 = v(j$0,i0+2);     v1_2 = v(j$1,i0+2);     v2_2 = v(j$2,i0+2)
*soption unroll(2)
*soption nolooproreroll
*voption vec
            do k=1,j$2
              l1$a(k,i$1+0)=l1$a(k,i$1+0)
     1          - u0_0*v(k,i0+0) - v0_0*u(k,i0+0)
     1          - u0_1*v(k,i0+1) - v0_1*u(k,i0+1)
     1          - u0_2*v(k,i0+2) - v0_2*u(k,i0+2)
              l1$a(k,i$1+1)=l1$a(k,i$1+1)
     1          - u1_0*v(k,i0+0) - v1_0*u(k,i0+0)
     1          - u1_1*v(k,i0+1) - v1_1*u(k,i0+1)
     1          - u1_2*v(k,i0+2) - v1_2*u(k,i0+2)
              l1$a(k,i$1+2)=l1$a(k,i$1+2)
     1          - u2_0*v(k,i0+0) - v2_0*u(k,i0+0)
     1          - u2_1*v(k,i0+1) - v2_1*u(k,i0+1)
     1          - u2_2*v(k,i0+2) - v2_2*u(k,i0+2)
            enddo
          enddo
*
        enddo
      return
      end
```

図 5 (3-3-2) 展開のリダクション演算のソースコード
Fig. 5 Source code of reduction with (3-3-2) unrolling.

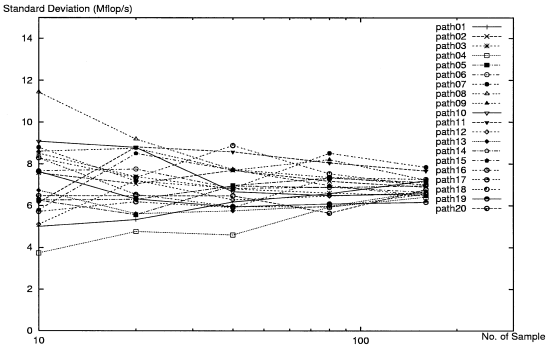

図 6 (1-1-1) 展開のランダムサンプリングしたポイントでの性能の近似標準偏差値
Fig. 6 Estimated standard deviation of performance with (1-1-1) unrolling from random sample points.

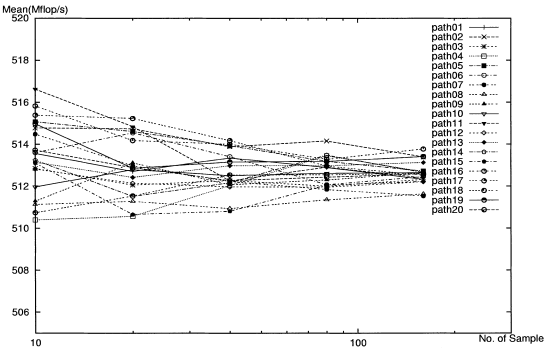

図 7 (1-1-1) 展開のランダムサンプリングしたポイントでの性能の近似平均値
Fig. 7 Estimated mean of performance with (1-1-1) unrolling from random sample points.

能値，3870 次元の性能値，…から算出される．すなわち，これらが，ある乱数パスでの 10 個のランダムサンプリングから算出した近似平均，近似標準偏差となる．

図 6 に (1-1-1) 展開での 10 個，20 個，40 個，80 個，160 個のサンプリング点によって算出した近似標準偏差を乱数パスごとに示す．**図 7** には同様のサンプリングにおける近似平均を示す．

図 6，図 7 から，ランダムサンプリングしたポイントでの性能の標準偏差，および平均が収束していく様子が確認できた．3000–4000 次元全体での標準偏差，平均は，それぞれ 7.042996 Mflop/s，512.6649 Mflop/s であり，それらの値の近傍に収束していることが確認できる．

20 パスでの近似平均値 M_L の標準偏差 $\omega(M_L)$，近似標準偏差 σ_L の標準偏差 $\omega(\sigma_L)$，近似標準偏差 σ_L の最小値 $min(\sigma_L)$ を算出した結果を**表 1** にまとめる．

表 1 20 パスでの近似平均値 M_L，近似標準偏差 σ_L の各統計量（L=サンプル数）
Table 1 Statistic values of estimated mean M_L and of estimated standard deviation σ_L from 20 paths (L=number of sample).

L	M_L 標準偏差	σ_L 標準偏差	σ_L 最小値
10	1.735529	1.768973	3.736653
20	1.418387	1.281866	4.758869
40	0.948270	1.017397	4.595856
80	0.686058	0.800070	5.643155
160	0.539976	0.422868	6.163444

また，表 1 の結果から 3.2 節の式 (3) を満たす q_L を求めた数値（簡単のため整数とする），および $VaR(2\sigma)$ の算出式は**表 2** のとおりとなった．ここで得られた $VaR(2\sigma)$ の算出式を次節，(1-1-1)，(3-3-2)，(5-5-2) の 3 ケースで検定評価する．

4.3 性能保証値算出アルゴリズムに対する評価結果

表 2 で得られた算出式の保証値 $VaR(2\sigma) = M_L - 3\sigma_L$，$M_L - 4\sigma_L$，$M_L - 5\sigma_L$，$M_L - 7\sigma_L$ を，実際の

表 2 表 1 の結果を満足する q_L の数値，および $VaR(2\sigma)$ 算出式（L=サンプル数）

Table 2 Values of q_L satisfying results in Table 1, and expression of calculaing $VaR(2\sigma)$ (L=number of sample).

L	$\frac{3\omega(M_L)+2\omega(\sigma_L)}{min(\sigma_L)}$	q_L 値	$VaR(2\sigma)$ 算出式
10	4.2338491	5	$= M_L - 7\sigma_L$
20	2.5103353	3	$= M_L - 5\sigma_L$
40	1.9472307	2	$= M_L - 4\sigma_L$
80	1.2153829	2	$= M_L - 4\sigma_L$
160	0.6744826	1	$= M_L - 3\sigma_L$

表 3 各保証レベル（M_L からの引き幅）での性能保証値（上段，単位 Mflop/s）および保証値以下の個数（下段）（L=サンプル数）

Table 3 Performance assurance value (upper side, in Mflop/s) and number of samples under the value (lower side) with each assurance level (L=number of sample).

L	展開	$3\sigma_L$	$4\sigma_L$	$5\sigma_L$	$7\sigma_L$
10	(1-1-1)	499.19	495.45	491.71	484.24
		27	11	6	4
	(3-3-2)	1017.07	996.25	975.43	933.80
		121	56	11	4
	(5-5-2)	1220.10	1214.84	1209.57	1199.05
		138	89	61	40
20	(1-1-1)	497.90	492.34	486.78	477.26
		16	7	5	1
	(3-3-2)	1017.23	995.63	974.04	930.84
		122	54	11	4
	(5-5-2)	1208.55	1200.54	1192.52	1176.49
		58	41	28	16
40	(1-1-1)	498.24	493.65	489.05	479.86
		17	9	5	2
	(3-3-2)	991.30	964.67	938.04	884.78
		43	8	4	0
	(5-5-2)	1202.73	1192.44	1182.16	1161.58
		44	28	17	8
80	(1-1-1)	495.63	489.86	484.21	472.93
		11	5	4	1
	(3-3-2)	985.68	957.03	928.37	871.06
		29	7	4	0
	(5-5-2)	1198.45	1187.41	1176.58	1154.93
		39	21	16	7
160	(1-1-1)	494.91	488.75	482.59	470.26
		11	5	3	1
	(3-3-2)	973.34	941.39	909.45	845.55
		10	4	0	0
	(5-5-2)	1198.43	1187.21	1175.99	1153.54
		39	21	16	7

1,001 個の性能数値で評価する．$VaR(2\sigma)$ は，通常起こりうる 97.725%中のケースを保証する．つまり，下方リスク領域は 2.275%である．よって，1,001 個のポイントのうち，$VaR(2\sigma)$ を下回る個数が 22 個までであれば $VaR(2\sigma)$ は保証値として合格である．**表 3** に，各パスから生成された性能保証値のうち最大のものが，実際に保証値を下回った性能の個数を，10 個，20 個，40 個，80 個，160 個のサンプル数，アンロールパタンごとに示す．

表 3 から，サンプル数 80 のケースでは算出式 $M_L - 4\sigma_L$ によって，97.725%の性能保証が可能であることが，3 つのアンロールパタンで確認できる．

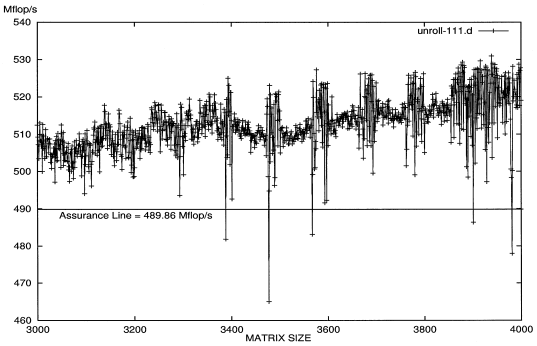

図 8 (1-1-1) 展開時の $M_L - 4\sigma_L$ 保証ライン

Fig. 8 $M_L - 4\sigma_L$ performance assurance line of (1-1-1) unrolling.

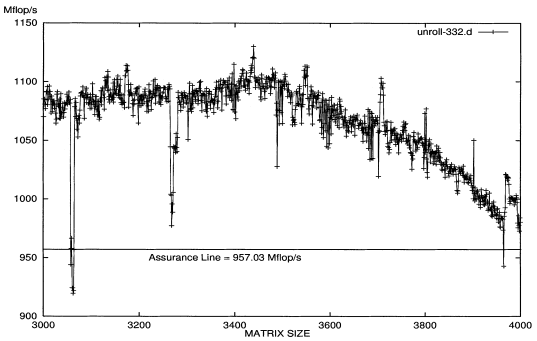

図 9 (3-3-2) 展開時の $M_L - 4\sigma_L$ 保証ライン

Fig. 9 $M_L - 4\sigma_L$ performance assurance line of (3-3-2) unrolling.

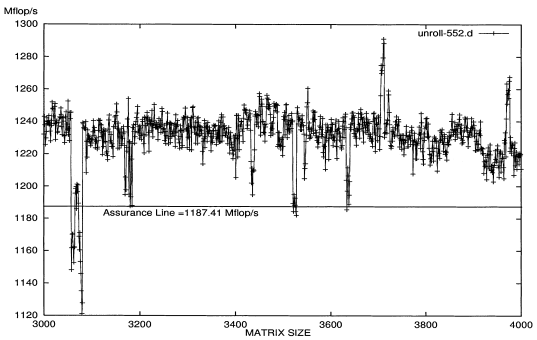

図 10 (5-5-2) 展開時の $M_L - 4\sigma_L$ 保証ライン

Fig. 10 $M_L - 4\sigma_L$ performance assurance line of (5-5-2) unrolling.

また，(1-1-1) よりも (3-3-2) が，(3-3-2) よりも (5-5-2) のほうがより多くのサンプル点を要することが分かる．これは，アンロール段数が増えるほどに性能のブレが大きくなることを反映している．

図 8，**図 9**，**図 10** に，サンプル数 80 のケースにおける 1001 ポイントの実際の性能と性能保証値のライン $M_L - 4\sigma_L$ を示す．これらの図から，(5-5-2) のアンロール時にはややライン以下の個数が多いが，いずれも 2σ の 97.725%としての性能保証ができていることが分かる．

5. まとめ

5.1 結論

GRIDコンピューティング環境における行列ライブラリ向けに，性能保証方式を検討した．特に，金融リスク管理で利用されている VaR（Value-at-Risk，バリュー・アット・リスク）というリスク指標を適用し，GRIDコンピューティングで必要とされる性能保証値を算出するアルゴリズムを提案した．今回提案の方式は，(1) 性能の分布の平均がほぼ一定である領域を対象とする，(2) 性能の分布は正規分布に従うとする，という条件の下，性能を左右するパラメータとして行列次元（行列のサイズ）をとりあげ，ランダムに選んだ行列次元のサンプル点から得られる近似標準偏差と近似平均値から性能保証値を算出する．

SR8000/F1 の 1CPU 上での，固有値計算ループにおける性能に対して，本保証性能値の妥当性を評価した結果，次元数 3000 から 4000 次元の中のサンプル数 80 のケースにおいて，標準偏差 2σ 分の $VaR(2\sigma)$ である 97.725%の性能保証が可能であることが，上記固有値計算ループの 3 つのアンロールパタンで確認でき，十分な性能保証値となることが明らかになった．

5.2 今後の課題

以下に今後の課題を列挙する．

第 1 に，本報告では，性能のばらつきが正規分布であるという仮定の下，下方リスクである VaR を性能保証値として算出したが，性能のばらつきは必ずしも正規分布とは限らない．今後，より実際の性能を反映する分布での性能保証値を検討していく．

第 2 に，今回用いた性能データは，キャッシュ競合で性能が安定していないことが分かっており，それを回避する安定化方法が開発されている[9]．今後，安定化方法を導入したうえでの性能保証値の評価を実施していく．

第 3 に，本報告では，性能保証値を VaR としていたが，他のリスク指標の候補もある．金融リスク管理でも，実際，CVaR（Conditional Value-at-Risk）という，VaR 値以下の性能の平均をとって，リスク値とする新しい評価手法が提案されている．今後，性能の保証値として VaR 以外の有効な指標を検討していく．

第 4 に，本報告では，ライブラリ自体の性能保証を検討したが，ライブラリポータルでは，ネットワークの QoS の検討を含めた性能保証が必要になる．今後，グリッド関連ソフトウェアを適用しつつ，統合的な性能保証を検討していく．

謝辞 研究の方向性に関し有益な助言をいただいた電気通信大学の弓場敏嗣教授，片桐孝洋助手に謝意を表します．また，平成 13 年度に実施された共同研究で施設利用させていただいた日本原子力研究所に謝意を表します．最後に，有益なコメントをいただいた査読者の各位に謝意を表します．

参考文献

1) Ninf project. http://ninf.etl.go.jp/
2) NetSolve project. http://www.cs.utk.edu/netsolve/
3) Dongarra, J. and Eijkhout, V.: Self-adapting numerical software for next generation applications, *The International Journal of High Performance Computing Applications*, Vol.7, No.2, pp.125–131 (2003).
4) ATLAS project. http://www.netlib.org/atlas/index.html
5) Kuroda, H., Katagiri, T. and Kanada, Y.: Knowledge Discovery in Auto-tuning Parallel Numerical Library, *Progress in Discovery Science, Final Report of the Japanese Discovery Science Project*, Lecture Notes in Computer Science 2281, pp.628–639, Springer (2002).
6) Kudoh, M., Kuroda, H. and Kanada, Y.: Parallel Blocked Sparse Matrix-Vector Multiplication with Dynamic Parameter Selection Method, *ICCS2003* (*International Conference on Computational Science 2003*), International Conference, Melbourne, Australia and St. Petersburg, Russia, June 2–4, 2003, Proceedings, Part III, Lecture Notes in Computer Science 2659, pp.581–591, Springer (2003).
7) 自動ブロック化・通信最適化ライブラリ ABCLIB. http://www.abc-lib.org/
8) 直野 健，今村俊幸：自動チューニング型固有値ソルバについて，情報処理学会研究報告，SWoPP2002, Vol.2002, No.91, pp.49–54 (2002).
9) 今村俊幸，直野 健：性能安定化を目指した自動チューニング型固有値ソルバーについて，*SACSIS* (*Symposium on Advanced Computing Systems and Infrastructures*) *2003*, pp.145–152 (2003).

10) Foster, I., Roy, A. and Sander, V.: A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, *8th International Workshop on Quality of Service* (*IWQOS 2000*), pp.181–188 (June 2000).
11) Takefusa, A., Casanova, H., Matsuoka, S. and Berman, F.: A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid, *10th IEEE International Symposium on High Performance Distributed Computing* (*HPDC-10*), pp.406–415 (2001).
12) Yang, L., Schopf, J.M. and Foster, I.: Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments, *Supercomputing 2003* (Nov. 2003).
13) 大石進一：精度保証付き数値計算（2）—線形方程式の高速精度保証とプログラミング技法，シミュレーション，Vol.19, No.1, 39/45 (2000).
14) 木島正明：金融リスクの計量化（上）バリュー・アット・リスク，金融財政事情研究会 (1997).

(平成 15 年 10 月 6 日受付)
(平成 15 年 12 月 11 日採録)

直野　健（正会員）

1968 年生．1994 年京都大学大学院理学研究科数理解析専攻修士課程修了．同年（株）日立製作所中央研究所入所．以来，並列計算機 SR2201，SR8000 向け行列計算ライブラリの研究開発に従事．現在の主たる研究テーマは，HPC の研究，並列固有値計算ライブラリ，HPC の金融工学への応用．日本応用数理学会，日本金融・証券計量・工学学会各会員．

今村　俊幸（正会員）

1969 年生．1996 年京都大学大学院工学研究科応用システム科学専攻博士後期課程単位認定退学．同年日本原子力研究所入所．計算科学技術推進センターにて途切れのない思考を支援する並列処理基本システム STA の開発に従事．2001 年から 2002 年までシュツットガルト大学 HLRS にて招聘研究員．2003 年より電気通信大学講師．現在に至る．HPC とその周辺ソフトウェア，数値計算における並列・分散処理の研究に従事．博士（工学）．平成 11 年日本応用数理学会論文賞，同年石川賞企業部門受賞．日本応用数理学会，SIAM 各会員．

恵木　正史

1971 年生．2000 年名古屋大学大学院理学研究科素粒子宇宙物理学専攻博士後期課程満了．同年（株）日立製作所中央研究所入所．コンピュータシステムの性能評価，経済物理学，集団遺伝学等，多岐にわたる分野において数理統計学を応用する研究に従事．