Vol.2012-HPC-133 No.10
2012/3/26

ポストペタスケール高性能計算に向けた階層的プログラミングモデルの提案

池上 努[†1] 田中良夫[†1] 中田秀基[†1]
高野了成[†1] 関口智嗣[†1]

次世代スパコンで想定される百万コア越の大並列環境を対象に、高効率かつ頑健なアプリケーションを実装する上で必要となるアルゴリズムとプログラミング手法について考察する。まず、マスタ・ワーカ型の実装が可能であり、かつワーカの障害に対して寛容なアルゴリズムを紹介する。次いでワーカの動的な増減を可能にするプログラミング手法として、グリッド環境で実績のある、遠隔手続き呼出し (RPC) と MPI を組合せたハイブリッド型の手法を提案する。グリッド環境での経験を元に、従来型 RPC に対する改善案を検討する。

Hierarchical programming model for the post-petascale era

Tsutomu Ikegami ,[†1] Yoshio Tanaka ,[†1]
Hidemoto Nakada ,[†1] Ryousei Takano [†1]
and Satoshi Sekiguchi [†1]

Efficient and robust applications are required for the massively parallel computational environment expected in the post-petascale era. Requirements to develop such applications are discussed from both algorithm and programming model points of view. We will introduce master-worker type algorithms that are tolerant to worker failures, followed by a programming model that allows dynamic addition/removal of workers. The programming model is a hybrid of the remote procedure call (RPC) and MPI, which achieves a lot of success in the grid environment. Necessary improvements on RPC to be suited to the next generation super computers will be discussed.

1. はじめに

我が国の科学技術および産業の発展において、高性能計算は不可欠なものとなっている。高性能計算の象徴とも言えるスパコンの性能は、少くとも過去 20 年に亘りほぼ指数関数的な延びを示してきた。2011 年の時点で最速のスパコン「京」は 88,128 個の CPU を搭載し、Linpack ベンチマークで 10.51 PFlops を達成している。性能向上を外挿すると、2018 年頃にはスパコンの性能はエクサフロップスに到達すると予測される。今後あらわれるであろうこれらポストペタ〜エクサ級のスパコンは、大量の計算ユニットを相互接続した大規模並列機になることがほぼ確実視されており、そのシステムソフトウェアおよびプログラミング等については、国内外で盛んに議論が進んでいるところである。国際的には、International Exascale Software Project[1)] において、各国の高性能計算に関する研究者が集まってエクサスケール高性能計算の実現に必要な技術の洗い出しやそのロードマップ作成を行なっている。国内では戦略的高性能計算システム開発に関するワークショップにおいて、5 年後にセンター運用可能な高性能並列計算機システムとしてはどのようなシステム仕様が考えられ、そのために今後どういう研究開発をしていくべきなのか、アプリケーション開発者、数値計算ライブラリ、プログラミング言語、ミドルウェア、システムソフトウェア、ハードウェア開発者が集まって議論を進めている。こうした取り組みの中で、数百万規模の演算コアから構成される次世代スパコンでは、計算実行中に発生する異常への対処が重要な課題として挙げられた。すなわち、ポストペタ〜エクサスケール高性能計算では、大量の演算コアを使いこなす大規模並列性だけでなく、実行中の異常を検知して復旧し、計算を継続する頑健性が求められる。

次世代スパコンで必須とされる頑健性の実現には、二通りのアプローチが考えられる。ひとつは計算の多重化など、ハードウェアやシステムソフトウェア側で頑健性を担保するやり方で、実現コストは高くつくものの従来型のアプリケーションに対応できる。もうひとつはアプリケーションの段階から障害の発生を前提に設計するやり方で、低コストで実現できるが新規なアプリケーション開発が必要となる。本稿では後者のアプローチに焦点をあて、階層的な並列化を意識した耐障害性を中心にアルゴリズムとプログラミングモデルの両面から論じる。アルゴリズムとして、マスタ・ワーカ型の実装が可能でありかつワーカの障害

†1 産業技術総合研究所
AIST

に対して寛容なものを 2 例、紹介する。またアルゴリズムの実装手法として、遠隔手続き呼び出し (Remore Procedure Call, RPC) に基いたプログラミングモデルを提案する。提案手法はポストペタスケール高性能計算アプリケーションが必要とする数千万規模の演算コアを用いる大規模並列性と実行中に発生する異常を適切に検知して復旧し実行を継続する頑健性を満たし、ポストペタスケール高性能計算環境上で長時間安定して動作するアプリケーションを容易に開発・実行可能とするものである。要素技術およびミドルウェアは開発中であるが、ここでは階層的並列性を持つアプリケーションのプログラミング環境に対する要求要件を整理し、提案プログラミング手法の概要と、ポストペタスケール高性能計算を実現するために解決すべき課題およびその解決策について述べる。次節で頑健性を実現しうるアルゴリズムやアプリケーションの具体例を紹介する。3 節 ではこれを実現するためのプログラミングモデルならびに想定される課題を提議する。

2. アルゴリズム

この節では、アプリケーションレベルで頑健性を担保するアプローチの候補として、二つの例を見ていく。ひとつはブロック櫻井・杉浦法に基く固有値解法で、アルゴリズムに内在する冗長性を活用することにより、障害で失われた計算内容を補完する。もうひとつは半導体デバイス設計への応用で、アプリケーションに内在する並列性を引き出すことにより、自明な耐障害性の実現を目指す。

2.1 固有値解法

ブロック櫻井・杉浦 (Bloss) 法[2] は周回積分を用いた一般化固有値問題解法で、大規模疎行列の内部固有値問題を大並列で解くのに適している。A, B を $n \times n$ の複素行列、Γ を複素平面上の閉曲線とし、Γ 内部の領域を G と定義する。Bloss 法は一般化固有値問題 $(\lambda B - A)q = 0$ の固有対 (λ, q) の内、$\lambda \in G$ なるものを選択的に取り出す手法である。Bloss 法では以下で定義するモーメント演算子 M_k が中心的な役割を果す。

$$M_k = \frac{1}{2\pi i} \oint_\Gamma m_k(z)(zB - A)^{-1} dz \tag{1}$$

$m_k(z)$ は Γ 内で正則な変調関数で、一般には $m_k(z) = z^k$ が用いられる。モーメント演算子 M_k は $\lambda \in G$ なる内部固有空間への射影演算子として働く。対角化可能なケースを仮定すると、任意ベクトル v が右固有対 (λ_i, q_i) を基底ベクトルとして $v = \sum_i c_i q_i$ と展開されるとき、$M_k B v$ の固有スペクトルは

$$M_k B v = \sum_i f_k(\lambda_i) c_i q_i, \tag{2}$$

$$f_k(x) = \frac{1}{2\pi i} \oint_\Gamma m_k(z) \frac{1}{z - x} dz = \begin{cases} m_k(x), & x \in G, \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

のように変化する。すなわち、$M_k B$ の作用により v の内部固有空間外の固有成分が消滅するだけでなく、固有スペクトルが関数 $m_k(\lambda)$ に従って変調を受ける。この性質により、L 本のランダムな初期ベクトル $V \in \mathbb{C}^{n \times L}$ から出発して基底ベクトル

$$S_R = \{M_0 V, M_1 V, \ldots, M_{K-1} V\} \in \mathbb{C}^{n \times LK} \tag{4}$$

を構築すると、これは内部固有空間を張り、G 内の固有値の数を n_Γ として $LK > n_\Gamma$ ならば S_R 内の線形独立な成分は n_Γ 本となる。n_Γ 本の独立な基底ベクトル $U \in \mathbb{C}^{n \times n_\Gamma}$ は $S_R = U s W^H$ と特異値分解することで得られる。ここで上付きの H は複素共役転置を示す。この基底 U の下で Rayleigh-Ritz 法を適用すると、内部固有値を計算できる。まず $n_\Gamma \times n_\Gamma$ 行列 $\mathbb{A} = U^H A U$, $\mathbb{B} = U^H B U$ を構築し、小型の固有値問題 $(\lambda \mathbb{B} - \mathbb{A})\mathsf{q} = 0$ を解く。この固有値はもとの問題の内部固有値に一致し、固有ベクトルは $q = U\mathsf{q}$ で与えられる。

実際の計算では、式 (1) の周回積分は求積公式を用いて数値近似する:

$$\bar{M}_k = \sum_{j=1}^{M} w_j m_k(z_j)(z_j B - A)^{-1} \tag{5}$$

ここで z_j, w_j は求積公式の節点および重率である。$M_k V$ の評価には、一連の節点について線形方程式 $Y_j = (z_j B - A)^{-1} V$ を解く必要があるが、この部分が Bloss 法で最も計算量を要する箇所である。幸いなことに方程式の間に依存関係がなく、独立・並行に解くことができるので、線形方程式の並列解法とあわせて階層的な並列性を無理なく実現できる。

周回積分に数値近似を用いる影響は、射影演算子としての性能として式 (3) に反映される:

$$\bar{f}_k(x) = \sum_{j=1}^{M} w_j m_k(z_j) \frac{1}{z_j - x}, \tag{6}$$

数値近似の結果、射影演算子の特性は $\bar{f}_k(\lambda)$ で規定されるようになる。一般に $\bar{f}_k(x)$ は $f_k(x)$ に見られる明瞭な境界がぼやけた関数になり、G の外側でも有限な値をとる。この結果、G 外部の微小な固有成分が基底ベクトル空間に混入するようになり、生成した内部固有空間の実効次元 $\bar{n}_\Gamma$ は n_Γ よりも大きくなる。したがって L および K は、G 内部の固有成分だけでなく Γ 周縁部も含めた固有副空間を張れるよう、十分に大きくとらなければな

Vol.2012-HPC-133 No.10
2012/3/26

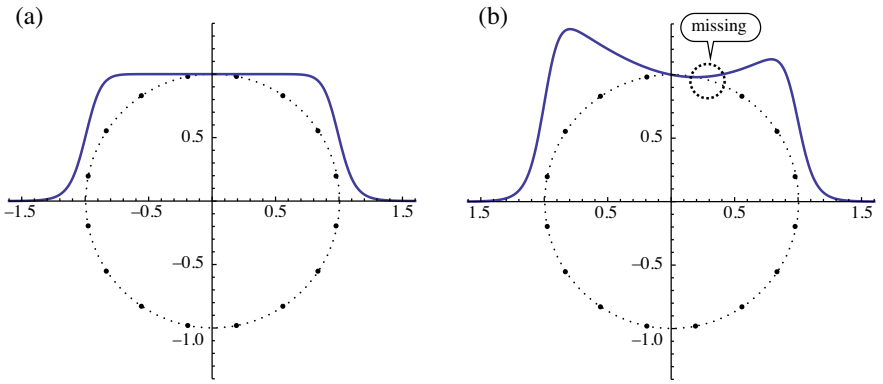

図 1 Bloss 射影演算子の特性関数。(a) 全節点が揃っている時の特性 $|\bar{f}_k(x)|$, (b) 節点が一つ欠けた時の特性 $|\tilde{f}_k(x)|$.

Fig. 1 Property functions of the Bloss projection operator. (a) $|\bar{f}_k(x)|$ with all nodes available, (b) $|\tilde{f}_k(x)|$ with one-node missing.

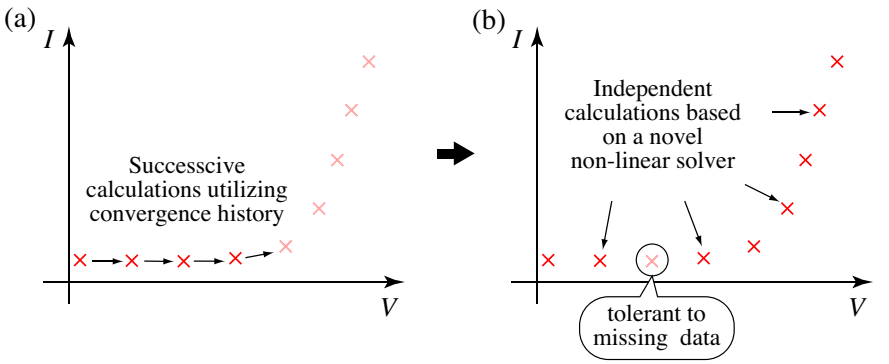

図 2 半導体デバイスの電流-電圧特性計算の例。(a) 従来型の手法 (b) 改良型非線形解法

Fig. 2 I-V property calculation in the semiconductor device simulations. (a) conventional method, (b) improved non-linear solver.

らない。この部分の冗長性を耐障害性の実現に活用することができる。すなわち、障害の影響で一部の線形解 Y_j が欠損した場合、Y_j を再計算することなく、欠損部を除いた節点を使って求積公式を再構成する。新たに定義される射影演算子 $\tilde{M}_k$ の特性 $\tilde{f}_k(x)$ は、$\bar{f}_k(x)$ に比べ Γ 周縁部での振幅がさらに増大するため、周縁固有値の混入の度合いが増す。しかしあらかじめ LK を十分大きくとってあるか、もしくは周縁部の固有値密度がそれほど高くない場合、射影演算子の特性悪化の影響を回避することができる。

Bloss の耐障害性に関する数値実験の例を **図 1** に図示する。Γ として原点を中心とした単位円をとり、変調関数は $m_k(z) = 1$, 求積公式は 16 点の台形公式を採用した。全節点が揃っている時の特性関数 $\bar{f}_k(x)$ の絶対値を実軸に沿ってプロットし、図 1(a) に示した。積分を数値近似した影響で、理想的なケース ($f_k(x)$) と比較すると $x = \pm 1$ における切断が若干、鈍っていることがわかる。節点のひとつを除き、重率 w_j を決めなおした時に得られるモーメント演算子 $\tilde{M}_k$ の特性関数 $|\tilde{f}_k(x)|$ を 図 1(b) に示す。関数全体の形状はかなり変化するものの $x = \pm 1$ 近傍の挙動は良く保たれており、$\tilde{M}_k$ が $\bar{M}_k$ 同様、射影演算子として有効に働くことがわかる。

2.2 半導体デバイス設計

半導体デバイスの設計では、その電気的な特性を試作することなく予測するため、デバイスシミュレーションが活用されている。デバイスシミュレーションでは、電極電圧など各種境界条件を設定した上で性格の大きく異なる複数の非線形方程式を連立させて解いている。非線形解法には Newton 反復法が用いられ、各反復ステップで線形方程式を解く必要がある。定量的な評価には二次元モデルや部分カットモデルを用いた計算では不十分で、素子全体の三次元構造を考慮したデバイスシミュレーションが不可欠となる。この結果、解くべき線形方程式が大規模化・複雑化するだけでなく、一般に非線形解法の収束性も悪化してしまう。このため、従来は境界条件を徐々に変えながら逐次的に計算を進め、直前の計算結果を利用することで強い非線形性に対処してきた (**図 2**(a))。

我々は非線形収束の問題を根本から見直し、収束履歴に頼らない非線形収束アルゴリズムの開発を進めている。これが実用化されると、複数の境界条件について独立・並行に計算を進めることが可能となるため、デバイスの特性曲線を計算する際、階層的な並列性を容易に実現できる (図 2(b))。さらに上位の並列階層としては、計算した特性曲線を評価し、遺伝的アルゴリズムなどを用いてデバイス構造を自動最適化するアプリケーションが想定される。このようなアプリケーションは、最下層の計算の一部、すなわちいくつかの境界条件における情報が欠損しても、上位層の計算を継続するよう実装することが可能である。さらに最適化の過程でより計算時間を要する (= 非線形性の強い) 境界条件を判別し、より多くの計算資源を配分することで負荷の均等化を図ることも可能となる。

3. プログラミングモデル

前節で見たアプリケーションはどちらもマスタ・ワーカ型の実装が可能であり、アルゴリズム上、下層ワーカの完全性を必要としないことから、耐障害性を担保することができる。マスタ・ワーカ型の実装では、ワーカ側のタスクを並列処理することで階層的な並列構造を無理なく実現できる。各ワーカの並列性を中規模程度に抑えることでワーカ単位の並列化効率を高く保つことができ、上位のマスタ階層で大量のワーカを束ねることで大並列の計算資源を効率良く活用することが可能になる。計算資源の大半はワーカで占められることから、障害発生箇所は統計的におおよそワーカ階層に限定される。したがってワーカの障害に対して寛容なアルゴリズムを採用すれば、アプリケーションレベルでの耐障害性を比較的容易に実現できる。

マスタ・ワーカ型アプリケーションの実装には様々な手段が考えられるが、耐障害性まで考慮すると、必ずしもその全てが適しているとは言えない。アプリケーションの実行を長時間安定かつ効率よく維持するためには、障害ワーカを動的に切り離すだけでなく、余剰計算資源の追加など計算資源を動的に按分する仕組みがあることが望ましい。並列プログラミング手法として、現在 MPI (Message Passing Interface) が広く利用されているが、MPI は障害発生時に全体の実行が停止してしまうため、大並列環境で長時間安定動作するようにアプリケーションを実装する事は困難である。階層的な並列構造を実現するプログラミング手法としては、MPI と OpenMP を組み合わせ、メッセージパッシングとマルチスレッド処理を併用するプログラミング手法が普及している。この手法では大量の計算資源を階層的に管理することで高い並列化効率が期待できるが、上位階層が MPI であるため、MPI の本質的な問題を解決する事は出来ない。

我々はこれまでグリッドミドルウェアの研究を通じ、遠隔手続き呼び出し (RPC) と MPI を組み合わせた階層的なプログラミングモデルを開発してきた。これははグリッドにおけるプログラミングモデルとして我々が提唱したもので、GridRPC として標準化すると共に、参照実装である Ninf-G[3] の開発を進めてきた。グリッド環境では計算資源の安定性を期待できないので、開発当初より耐障害性の担保が課題となっている。これら一連の研究では実際のグリッド環境上での検証を通じ、上位層に RPC を用いるプログラミング手法がスケーラブルかつ頑健なプログラムを実装する現実的な手段である事を示してきた。本研究ではこれらの研究成果や得られた知見を元に、ポストペタスケール高性能計算環境における RPC に基いたプログラム開発・実行環境について提案する。

提案手法は、RPC と MPI を組合せてマスタ・ワーカ型の階層的並列アプリケーションを実装するプログラミング手法である。この手法は Ninf-G の開発を通じてグリッド環境で十分な実績を積んでおり、シリコン表面への酸素注入過程を計算した量子古典マルチスケールシミュレーション[4] や生体高分子の量子化学計算[5] に応用されている。量子古典マルチスケールシミュレーションでは、日米 6 つの組織の 8 台のスパコンにより構成されるグリッドテストベッドを用い、20 日間にわたって合計 15 万 CPU・時間の計算を実施した。また生体高分子の量子科学計算では、環太平洋地域に分散した 10 台のクラスタ計算機からなるグリッドテストベッドを利用し、一連の計算を占有環境を作ることなく 70 日間で処理した。どちらの例も、元来マスタ・ワーカ型のアルゴリズムを採用していたが、元となるコードはモノリシックな構成で記述されていた。グリッド向けのコードはオリジナルコードから並列構造に注意しつつワーカ部分を切り出し、マスタ部分と RPC を介して接続する構成で開発した。その際、RPC とアプリケーション層の間にミドルウェアを構築し、耐障害性をここで担保することで長期間安定な計算処理を実現した。計算中に発生する障害の種類は様々で、ノードの単純故障や停電といった単純なものから原因不明のままワーカが応答しなくなるものまで多岐に亘る。これらの障害を適切に検出・通知するため、ミドルウェアの基盤となる RPC 部分に各種の障害検知機能を装備していった。通信経路の定期的なチェックと保全用に開発したハートビート機能や原因不明の障害まで補足するために用意したタイムアウト機能は、こうした実証実験を通じてその必要性が明かになったもので、ポストペタスケール世代に要求される耐障害性の実現にも応用可能である。また、グリッド環境ではワーカとして利用する計算資源の均質性が期待できないため、ワーカ側の並列数を適宜、調整することで、ワーカ単位での計算性能の均質化を図った。ポストペタスケール環境では計算資源の非均質性の心配はないが、ワーカ並列度の動的調整機能はワーカタスクが非均質である場合にも有効に活用できる。すなわち、並列度の高い高性能なワーカを適宜生成して多大な計算量を要求するタスクを割り付けることでタスク処理時間を均質化することが可能となり、負荷分散の改善を図ることができる。一方、グリッド環境ではその広域分散性から通信帯域の強化や通信遅延の削減に限界があり、通信性能の非均質性とあいまって、グリッドに対応可能なアプリケーションは自ずとタスク粒度の粗いものに限られてきた。ポストペタスケール環境ではこうした制約は大幅に緩和されるので細粒度のマスタ・ワーカ型実装も可能となり、適用範囲はより広くなると期待される。

以下、ポストペタスケール環境におけるマスタ・ワーカ型アプリケーションの開発を前提に、RPC やその上位ミドルウェア部分において今後取り組むべき課題を列挙する。

(1) マスタ側プロセスの強化

Ninf-G では単一のマスタプロセスがワーカとの通信を一手に引き受けるが、グリッド環境下では個々のマスタ・ワーカ通信の帯域が狭いので、マスタプロセスのマルチスレッド化で十分対応できた。ポストペタスケール環境では大量のワーカを生成できるだけでなく、通信性能の改善に伴いワーカタスクの粒度を低くすることが可能になるため、マスタ・ワーカ間の通信量が大幅に増大すると予想される。通信の輻輳を回避するためには、マスタプロセスの多重化や分散型データベースを介した入出力データの授受など、マスタ側プロセスの強化を検討する必要がある。こうした改良は、マスタの頑健性の担保も視野に入れて進める。

(2) 障害検知や復旧処理のサポート

Ninf-G では RPC 部分は障害検知に特化し、検知した障害はアプリケーションやミドルウェア層で対応した。前述の通り、障害の検出はワーカプロセスの死亡など明示的な異常に加え、ワーカの起動やタスク処理が所定の時間内に完了しないなど、ユーザが定義した異常状態をも障害として通知する機能を提供した。こうした障害はミドルウェアやアプリケーションなど上位層に API の返り値として通知され、障害への対応は全て上位層で明示的に記述する必要があった。ポストペタスケール環境では、障害の検出や障害からの復旧に関し、ハードウェアやシステムソフトウェアレベルでのサポートが期待されるが、これら機能を RPC 層でどのように活用していくか、検討する必要がある。たとえば障害からの復旧など、障害処理に時間を要する場合、2 節 で延べたようなアルゴリズムレベルでの耐障害性を有するアプリケーションでは負荷分散に悪影響を及ぼす可能性がある。こうしたアルゴリズムも視野に入れながら、障害発生時の対応処理を簡潔に記述する機能について研究を進める。

(3) 軽量な通信プロトコルの設計と実装

Ninf-G は広域分散環境での動作を前提としていたため、セキュリティ対策やエンディアン変換、マスタ・ワーカ間のハートビート送信などが必要となり、通信性能に対するオーバーヘッドが大きかった。この結果、単一システム上での通信性能を MPI と比較した場合、通信遅延はひと桁程度大きくなり、実効的な通信帯域も劣っていた。ポスペタスケール環境は均質な通信環境を前提とするため、より軽量なプロトコルの設計が可能となる。特にワーカの状態を確認するハートビートなど頑健性の実現に必要な機能について見直し、MPI と同程度の通信性能の実現を目指す。

(4) スケジューラとの連携

スパコンの運用は一般に共同利用の形態をとり、その計算資源はスケジューラを介して排他的に確保・利用する。Ninf-G はスケジューラに対して資源割当て要求を発行し、割当てられた資源を適宜分割してワーカを仕立てている。このため、一部のワーカで障害が発生した場合、これを切り離すためには割当てられた全計算資源を返却する必要があった。Ninf-G は広域に分散した複数のスケジューラを利用する都合上、各種スケジューラの最大公約数的な機能に絞った汎用のインタフェースを設計しており、スケジューラ固有の機能へのアクセスには限界があった。ポストペタスケール環境では単独のスケジューラがターゲットとなるため、きめ細かな資源管理が期待できる。障害の発生部分の切り離しや計算資源の途中追加など、マスタ・ワーカ型アプリケーションの特色を生かした資源管理を実現するため、スケジューラとの有機的な連携について検討する。

(5) ワーカ間の通信

Ninf-G の特徴として、ワーカプロセスが固有のステートを保持できる点が挙げられる。グリッド環境ではワーカは一般に広域に分散しているので、Ninf-G ではワーカ間の直接通信はサポートされておらず、ワーカの内部ステートを他ワーカから参照することはできなかった。ポストペタスケール環境ではワーカ間の通信環境が改善されるため、実用的なワーカ間通信が実現できる可能性がある。URI などを用いたワーカ上のデータの指定や非同期通信を用いた参照について検討を進める。

(6) デバッグ環境の整備

Ninf-G を用いた開発では、マスタ用のコードとワーカ用のコードを個別に用意する。このため、ワーカコードの開発にはテスト用に簡便化したマスタコードをあわせて用意する必要があり、効率的な開発の妨げとなってきた。ミドルウェアの中にはテストを簡便化するため、ワーカタスク毎に別個のプロセスを立ちあげる仕様とし、コマンドライン上でワーカタスクのデバッグを可能にしたものもあるが、ポストペタスケール環境ではタスク起動のオーバヘッドが無視できないため、現実的ではない。RPC に基くプログラミングモデルを普及させるためには、テストコードの自動生成やワーカに対する対話的なテストベッドの開発など、デバッグ環境の整備を進める必要がある。

Vol.2012-HPC-133 No.10
2012/3/26

参 考 文 献

1) Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M.S., Nagel, W.E., Nakashima, H., Papka, M.E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., vander Steen, A., Vetter, J., Williams, P., Wisniewski, R. and Yelick, K.: The International Exascale Software Roadmap, *Int. J. High Perf. Comp. App.*, Vol.25, pp.3–60 (2011).

2) Ikegami, T., Sakurai, T. and Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method, *J. Comp. Appl. Math.*, Vol.233, pp.1927–1936 (2010).

3) Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *J. Grid Comp.*, Vol.1, pp.41–51 (2003).

4) Takemiya, H., Tanaka, Y., Nakada, H., Sekiguchi, S., Ogata, S., Kalia, R. K., Nakano, A. and Vashishta, P.: Sustainable Adaptive Grid Supercomputing: Multiscale Simulation of Semiconductor Processing across the Pacific, *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (2006).

5) Ikegami, T., Maki, J., Takami, T., Tanaka, Y., Yokokawa, M., Sekiguchi, S. and Aoyagi, M.: GridFMO – Quantum Chemistry of Proteins on the Grid, *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pp.153–160 (2007).