1P07　耐障害性を考慮した並列 FMO プログラムの実装

○稲富雄一[1]、梅田宏明[2]、辻美和子[4]、村井均[3]、南一生[3]、横川三津夫[3]、佐藤三久[2]、青柳睦[1](1;九大、2;筑波大、3;理研、4;ベルサイユ大)

【はじめに】京をはじめとしたスパコンは、多数のプロセッサやメモリ、あるいは、ネットワークインターフェイスなどの部品で構成されている。スパコンの高性能化に伴って、今後、さらなる部品点数の増加や計算機構成の複雑化が想定される。そのようなスパコンを用いて超並列処理を行う場合には、実行途中にいずれかの箇所（計算ノード）が故障して使えなくなる可能性が非常に高くなる。現在の大規模並列プログラムのほとんどは MPI で並列化されていることが多いが、通常の MPI プログラムでは、実行途中にいずれかのプロセスがノード故障により停止すると、プログラム全体が異常終了する。その場合には、最初からその計算をやり直すしか方法がなく、故障するまでに行った計算が無駄になる。このような状況を回避するために、OS や MPI などのミドルウェアレベル、あるいは、アプリケーションプログラムのレベルにおいて、計算途中でチェックポイントファイル（以降 CPF）を出力して、異常終了した場合には、その CPF を用いて、途中から計算を再開する、という方法が考えられ、実際に利用されている。しかし、CPF の利用では、一般に CPF を出力するコストがかかること、ジョブの再投入のためにユーザーの手が必要になること、また、ジョブスケジューラによる待ち時間がかかること、などといった欠点がある。一方、マスタ・ワーカ型のプログラミングモデルは、マスタプロセスが複数のワーカに対して計算を割り当てながら大規模な計算を行う並列処理手法であるが、このモデルでは、ワーカが異常終了したことをマスタプロセスが検知できれば、計算途中で、あるワーカが異常終了しても、そのワーカに割り当てていた計算を他のワーカに割り当てることで計算を進めることができるため、ノード障害に対応しやすい（耐障害性がある）。そこで現在、（１）京コンピュータのジョブ管理機構と MPI ライブラリの機能を強化する（具体的にはワーカが異常終了しても、プログラム全体が停止しないようにするなど）、（２）マスタ・ワーカプログラミングが容易なモデルである Remote Procedure Call（RPC）の参照実装である OmniRPC[1]を上記の拡張機能を用いて実装する、という２つの方法で、耐障害性を考慮したマスタ・ワーカプログラミングを実現するという試みがなされている[2]。この２つの方法の有効性を確認するために、今回、マスタ・ワーカ型の実行モデルを持つ並列フラグメント分子軌道（FMO）法プログラムをターゲットとして、その性能評価や耐障害性への対応手法について検討を行うことにした。その手始めとして、並列 FMO プログラム OpenFMO を MPI の動的プロセス生成機能を用いたマスタ・ワーカプログラムに変更する作業を行ったので、その手法などについて報告する。

【並列 FMO プログラムの基本構造】

FMO 法は、巨大な分子を小さなフラグメントに分割して、各フラグメント（モノマー）とそのペア（ダイマー）に対する電子状態計算を行うことで分子全体の電子状態を近似する計算手法である。複数のモノマー・ダイマーに対する計算を並列に処理できること、なら

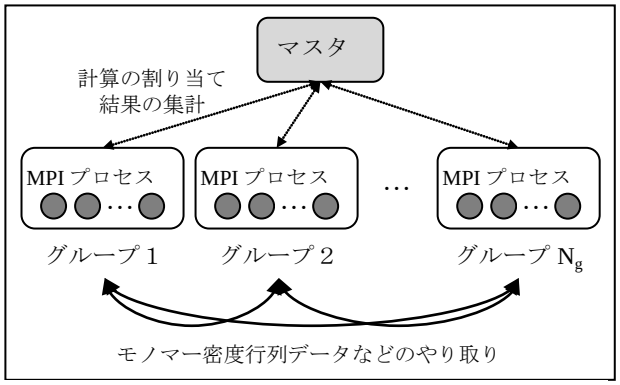

図 1：並列 FMO プログラムの基本構造

びに、各モノマー・ダイマーの計算もさらに並列化可能であることなどから、大規模並列処理向きの計算手法である。並列 FMO 計算を行うプログラムは、図 1 のような構造をとる。まず、並列処理に関わるすべてのプロセスを 1 つのマスタプロセスと、いくつかのグループ（複数プロセスで構成される）に分割する。マスタプロセスは、各グループに対して、どのモノマー・ダイマーの計算を行うべきかを指示し、計算が終了したグループからは計算結果を受け取る。各グループではマスタプロセスから割り当てられた計算をグループ内に存在するプロセスを用いて並列に処理し、割り当てられた計算が終了したら、その結果をマスタプロセスに渡して、次の計算を割り当ててもらう。このように、並列 FMO 計算は、マスタ・ワーカ型の実行形態をとっている。しかしながら、FMO 計算で必要となるモノマー密度行列データのやり取りのためにグループ間通信が必要となるため、純粋なマスタ・ワーカ型ではない、という特徴を持つ。

【MPI の動的プロセス生成を用いたマスタ・ワーカ FMO プログラム】

動的プロセス生成機能とは、あるプログラムから別のプログラムを起動する、という機能である。MPI-2 規格で提供されている動的プロセス生成には、別 MPI プログラムを起動する機能があるだけでなく、MPI で通信を行う際に用いられているコミュニケータの概念が、新規に起動された MPI プログラムと、その起動元の MPI プログラムとの間の通信で使える、という特徴がある。今回はこの MPI-2 の動的プロセス生成機能を用いて、最初はマスタプロセス 1 つだけを MPI プログラムとして起動して、その後、グループプロセス群（ワーカ）を、マスタプロセスから起動する、というプログラムにした。そこで 1 つ問題になるのが、モノマー密度行列授受に伴うワーカ間通信である。一般に、マスタ・ワーカ型プログラムでは、ワーカはマスタプロセスとしか通信しないため、ワーカ間での情報のやり取りはマスタプロセスを経由して行う必要がある。しかしながら、FMO 計算で必要となるモノマー密度行列データに関わる通信量は膨大になるため、そのすべてをマスタプロセスを経由させることは性能的に困難である。そこで、今回は、モノマー密度行列データを保持して、ワーカからの参照・更新要求に応答するだけのメモリサーバをマスタプロセスから生成して、データのやり取りをワーカ・メモリサーバ間のプログラム間通信で行うことにした（図 2 を参照）。

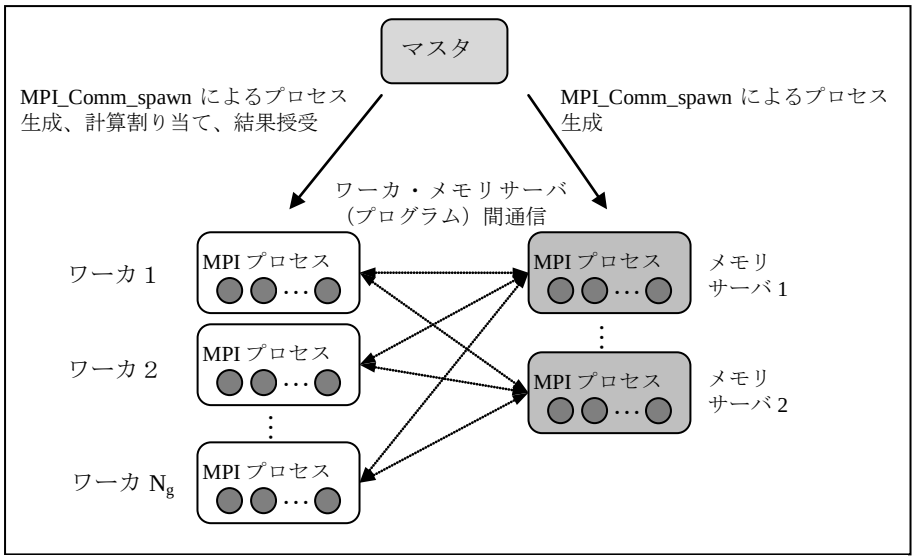

図 2：動的プロセス生成を用いたマスタ・ワーカ FMO プログラムの構造

プログラムの詳細、および、性能評価の結果については、当日報告する。

参考文献

[1] OmniRPC: a Grid RPC system for Parallel, M.Sato et. al, http://www.omni.hpcc.jp/OmniRPC/index.html.ja

[2] 村井均など、情報処理学会研究報告 Vol. 2013-HPC-138. No.26