ハイパフォーマンス
コンピューティング 87−21
（２００１．７．２７）

実行時情報を用いて通信を最適化するPC クラスタ上の並列化コンパイラ

横田大輔[†] 千葉滋[††,†††] 板野肯三[†]

分散メモリ機ではプロセッサ間の通信がボトルネックになりがちである。このため、並列計算機によっては特定のパターンの通信を高速に処理できるハードウェアを持っているものがある。CP-PACS/Pilot3(日立SR2201) の通信機構の RDMA(Remote DMA) には、制限つきながらメモリ上の不連続なデータを一回の送信命令で送信する機能を持っている。インスペクタ・エグゼキュータ方式は実行時の情報を用いて、プログラムの通信パターンを特定する。我々はこの方式を通信の最適化に利用する。我々が開発した並列化コンパイラはCP-PACS/Pilot3 の RDMA に最適化したコードを出力する。前回の研究ではインスペクタの処理を通常の 1 プロセッサの PC 上で行なったため様々な制限があった。今回我々は、PC クラスタを用いて通信の最適化のための情報を解析するようにした。

A parallelizing compiler on PC clusters optimizing for communication devices with run-time information

Daisuke Yokota,[†] Shigeru Chiba[††,†††] and Kozo Itano[†]

A number of parallel comupters with distributed memory provide special hardware for boosting the speed of typical inter-processor communication. For example, CP-PACS/Pilot3 equip special communication hardware RDMA (Remote DMA). This device can send a data in regular discontinual memory address by only one operation. The inspector-executor method parallelizes programs with run-time information. We extended this method to optimization for inter-processor communications. Our compiler can generate parallel code optimized for RDMA. Once, we implemented inspector phase to analyze communication patterns on standard PC with 1 processor. That caused some restricts. Now, we developed the parallelizing compiler on PC cluster to analyze communication patterns with inspector-executor method.

1. はじめに

計算機を使うシミュレーションは並列計算機でも膨大な時間がかかる場合がある。このような計算に用いられる高速な並列計算機は共同利用で運営されている場合が多い。膨大な時間がかかるシミュレーションは、特に実行時間を短くすることが望ましい。シミュレーションはある特定の処理を繰り返し行うことによって、正確さを増したり、シミュレートする時間を伸ばすので、計算量そのものを減らすことは難しい。そこで、少しでも少しでも効果的なコードが必要になる。

並列計算では、通信時間を短くすることが望ましい。分散メモリ機を用いた並列計算では通信が必要であるが、時間がかかる。このため、通信を効果的に行うように、ハードウェアとソフトウェアの両方でサポートが行われている。

そこで、もしハードウェアが効果的な通信機構をもっている場合、これを使って高速なコードを得たい。例えば CP-PACS/Pilot3(日立 SR2201)[6] にはRDMA(Remote DMA)[7] という特殊な通信機構が実装されている。しかしながら、このような通信機構はアプリケーションから効果的に使われているとは必ずしもいえない。このような通信機構はアプリケーションから直接実行するのは低レベルである。そこでコンパイラの支援は必須であるが、並列化コンパイラはこのような通信機構をうまく活用できてこなかった。なぜなら、正確な通信パターンを解析できなければ、コンパイラは正しい実行コードを出すために性能を犠牲にした冗長なコードを出すからである。

そこで我々は前回の研究で、CP-PACS/Pilot3 が持つ RDMA を効果的に使うコンパイラを実装した。このコンパイラは HPF のサブセットを入力とし、RDMA

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 東京工業大学情報科学科
Information Science, Tokyo Institute of Technology
††† 科学技術振興事業団 さきがけ研究 21
PREST,Japan Science Technology Corp.

を用いたSPMDのFortranプログラムを出力とする。このコンパイラはインスペクタ・エグゼキュータ方式[1)]を拡張して実装している。コンパイラは最初にインスペクタだけを実行するコードを出力しこれを実行する。次にその実行結果を元に、通信を最適化したコードを出力する。しかし、このコンパイラには大きな制限があった。全てのプロセッサが同じパターンで通信することをプログラマが保証するという点である。

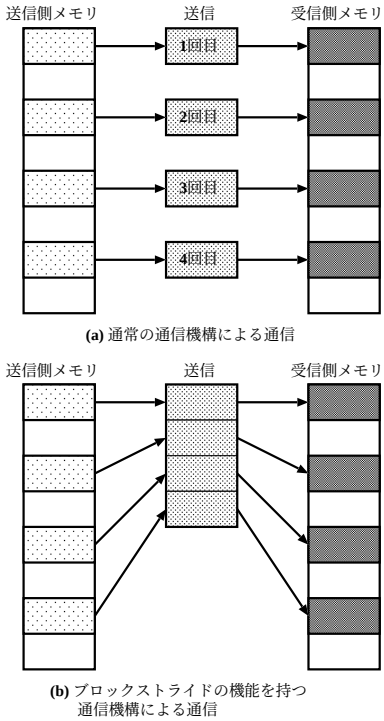
そこで本研究では、この問題を解決するコンパイラを実装した。このコンパイラは前回の研究が持つ制限を改良するためにPCクラスタ上で動作する。また、本研究はより効果的な通信の利用を目標としている。今回の研究では最適化する対象を前回の研究より増やした。

2. 特殊な通信機構の利用

並列計算では効果的な通信を行うことが望ましい。これは、並列計算においてプロセッサ間の通信がボトルネックになりがちだからである。このため、並列計算ではハードウェアやソフトウェアで効果的な通信をサポートしている場合が多い。そこで、もしハードウェアが効果的な通信機構をもっている場合、これを使った高速なコードを得たい。

例えば、CP-PACS/Pilot3が持っている通信機構にはRDMAがある。これはブロックストライドによる通信(図1)とTCW再利用型の通信をサポートしている。ブロックストライドとはある任意のバイト長連続し(ブロック:4〜1020Bytes)、ある任意の等間隔(ブロック:4〜65532Bytes)で繰り返されるデータである。RDMAはこれを一度の送信命令で送信できる。TCW再利用型の通信とは通信をあらかじめ設定しておくことで、レイテンシが少ない通信を高速におこなう機能である。これは、あらかじめ送信先のプロセッサ、送信先のメモリアドレスなどの送信権を確保しておく機能である。

このような特定の通信パターンを高速化するハードウェアを、一般のユーザが活用してプログラムすることは現実的には難しい。活用するためには、そのハードウェアの詳しい使い方、およびアプリケーションの実行中に現れるプロセッサ間の通信パターンを知っていなければならない。しかし、並列計算機の一般ユーザである物理学者や天文学者などの計算機の非専門家に、これらの知識をもってアプリケーションプログラムを書くように要求することは現実的ではない。一般のユーザはより抽象度の高いライブラリを使う傾向にある。CP-PACS/Pilot3でも、MPI,PVM,HPFなど



(a) 通常の通信機構による通信

(b) ブロックストライドの機能を持つ通信機構による通信

図1 ブロックストライド通信と通常の通信

のライブラリや言語がサポートされており、RDMAの機能はあまり使われていない。

このため、RDMAを活用するにはコンパイラの支援が必要であるが、従来の並列化コンパイラではあまり成功していない。しかし、最小限のデータを最小限の回数で、可能な限りブロックストライドとTCW再利用型の通信を利用するコードを出力するのが望ましい。このためにはコンパイル時に通信パターンを正確に分析する必要がある。

3. 通信パターンの解析

コンパイル時にプログラムが実行する通信パターンが分かっていると非常に効果的なコードを生成することができる。このため、並列化コンパイラは通信パターンを解析することが重要である。通信パターンの解析は静的な方法と動的な方法がある。

我々の実装した並列化コンパイラは通信の解析を動的に行う。これは、静的な解析に比べて解析にかかる時間が長くなるが、次のような利点がある。ソースコードに依存しない解析力と通信パターンが直接分かる点である。我々の開発しているコンパイラは、特殊な方法で、動的な解析結果をソースコードのコンパイルへフィードバックする。このため静的な解析のように、通信の解析結果を用いてコードを最適化できる。

この方式はインスペクタ・エグゼキュータ方式を拡張している。この方式については次章の実装で詳しく説明する。

以下では順に CP-PACS/Pilot3 の説明とインスペクタ・エグゼキュータ方式の説明をする。

3.1 CP-PACS/Pilot3

CP-PACS/Pilot3 は筑波大学計算物理学研究センターが所有する超並列計算機である。CP-PACS は 2048、Pilot3 は 128 個のプロセッサを持つ分散メモリ機であり、どちらも同じプロセッサユニットで構成されている。各プロセッサはクロスバーで接続されており、CP-PACS では三次元、Pilot3 では二次元に接続されている。これらの計算機はブロックストライドをサポートした RDMA 通信機構をもっている。

3.2 インスペクタ・エグゼキュータ方式

ループを分散メモリ機用に並列化する場合、それぞれの反復でどのような通信がプロセッサ間で必要になるか解析する必要がある。しかし、通信パターンをコンパイル時に静的に解析する事ができない、または困難な場合は、そのままでは並列化ができない。このような場合には、インスペクタ・エグゼキュータ方式を用い、実行時に通信の解析をおこなうことで並列化をすることができる。

インスペクタ・エグゼキュータ方式は不規則な通信パターンに強く、実際の通信パターンの複雑さに関係なく、並列化が容易である。この方式は通信パターンの解析を実行時におこなう。このため解析結果を用いて最適化をおこなう場合、コードを書き換えるような最適化は困難である。

インスペクタ・エグゼキュータ方式によって並列化されたコードは次の流れで処理をおこなう。まず、実行時に並列に処理されるループと同じ構造のループを実行し、通信が必要になる配列の添字をチェックして通信先を特定する。次に、計算に必要な配列要素を持っているプロセッサや配列要素のメモリ上の位置などの情報を各プロセッサ間で交換し合う。その後、自分の持っているデータを他のプロセッサが必要とする場合、これを送信する。また、自分が必要なデータを受信する。最後に本来のループを並列に実行する。インスペクタ・エグゼキュータ方式では通信パターンの解析とその結果をループの繰り返しの度に各プロセッサで交換し合う処理が必要であるが、通信パターンに変化がない事をユーザの指示などで保証できれば、このインスペクタの作業は一度で済む。

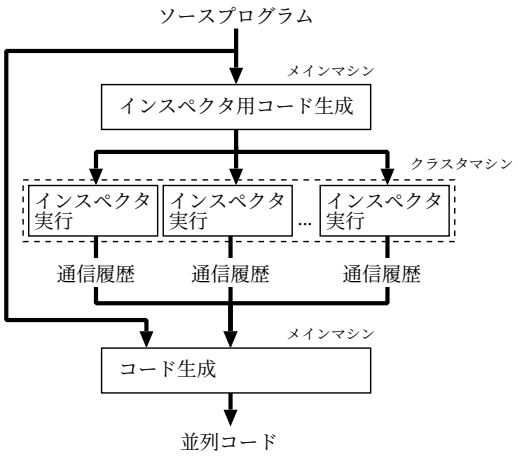


図 2 本方式の処理の流れ

4. 実　　装

われわれは動的な解析を用いて通信を最適化するコンパイラを実装した。このコンパイラは HPF のサブセットを入力とし、RDMA を用いた SPMD の Fortran プログラムを出力とする。また、このコンパイラはインスペクタ・エグゼキュータ方式を拡張したものである。処理の流れは図2のとおりである。我々のコンパイラは最初に入力されたソースプログラムを元に、インスペクタに相当するプログラムを出力する。このプログラムは PC クラスタ上で実行され、通信に関する情報だけを集めて記録する。この時、実際に通信はおこなわないので、通信で通信先が特定されるようなプログラムはコンパイルできない。次にこのデータをメインマシンに集めコードを生成する。理想的には、実行中に自己書き換えを行い最適化するべきだが、我々のコンパイラはオフラインで静的に最適化する。これは多くの並列計算機は PC やワークステーションによるクロスコンパイル環境を前提としているので、そのような運営形態と適合する。また、インスペクタに相当する部分を並列計算機ではなく PC クラスタで実行する。これは、多くの並列計算機は実行したいプログラムを投入しても、すぐには実行されず、他のユーザーのプログラムも含めた順番で実行されるからである。現実的な並列計算機の運用形態を考えるとコンパイル時には PC クラスタを用いる。

我々は、前回の研究ではこの処理に PC クラスタではなく 1 台の PC を用いた。1 台の PC で 1 プロセッサ分の通信パターンを解析して、全てのプロセッサが同じ通信パターンで通信すると仮定して最適化をおこなった。このため、全てのプロセッサが同じパターン

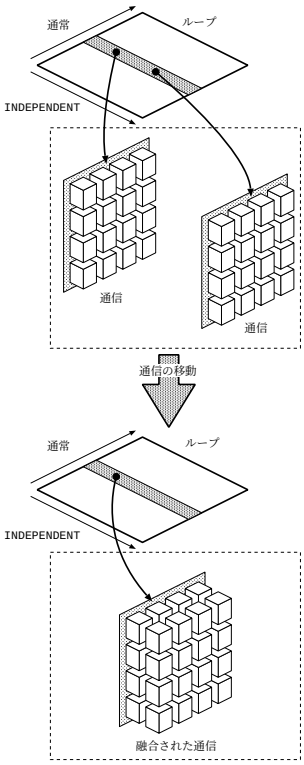


図 3　INDEPENDENT 命令を利用した通信の融合

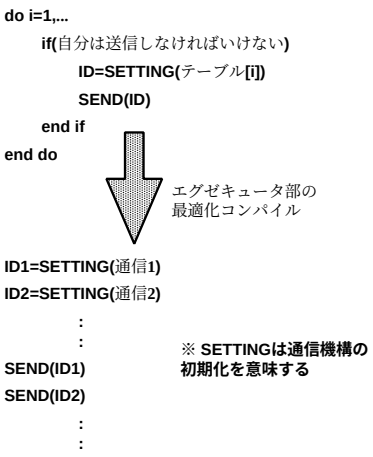


図 4　TCW 再利用型通信の利用

で通信するプログラムしかコンパイルする事ができなかった。今回の実装ではこの点が改良されている。

このコンパイラはがおこなう通信の最適化は 3 種類ある。ブロックストライドの利用と TCW 再利用型通信の利用とループの分割である。前回の研究ではループの分割に関する最適化を行なわなかった。ブロックストライドの利用は通信を可能な限りブロックストライドの塊にまとめ、通信回数を減らすことである。TCW 再利用型通信の利用は、通信のパラメータがループ不変である場合、TCW 再利用型通信を用いる通信命令を用いることである。ループの分割は、ループの反復を複数のプロセッサに配る際に、通信量が減るように配ることである。これらの処理はコスト高であるが、通信の解析はインスペクタに相当する部分を実行する時の１度で済むので、通信の最適化によるエグゼキュータ部の実行速度の向上の効果の方が支配的で全体として性能が改善される。また我々の開発したコンパイラは、EXECUTE 命令を提供して、インスペクタに相当する部分の実行時間を短縮するためのヒントをユーザが与えられるようにした。インスペクタ部の実行時には、この指定をされたループは 1 反復しか実行しない。この命令を使うことにより、反復解法を用いるシミュレーションの通信パターンの解析時間を短縮できる。

ブロックストライドの利用は次のように最適化される。本コンパイラは通信の回数を減らすために通信を HPF の INDEPENDENT 命令で指定された範囲でコードを移動し、可能な限りブロックストライドの単位にまとめるようにした。(図 3) 通信回数が減ることで通信に発生するレイテンシが減る。HPF の INDEPENDENT 命令とはプログラマがコンパイラに与えるヒントの一つである。INDEPENDENT と指定されたループは、ループが運ぶ依存がないかまたはそれが無視できるという事を、プログラマがコンパイラに保証するために使われる。したがってこの指定がされているループでは反復をインデックス変数の示す順番でおこなう必要がなく、このループ中でなされる通信も反復の順番に従っておこなう必要がない。この範囲では異なる反復でおこなわれるべき複数の通信をブロックストライドを単位とした 1 回の通信に置き換えて実行する事ができる。

TCW 再利用型通信は次のように最適化される。通信命令に必要なパラメータ（送信側の開始アドレス、ブロック幅、ストライド幅、受信側の開始アドレス、ストライド幅）がループ不変量である場合、コンパイラは TCW 再利用型通信が利用できると判断する。このとき、コールグラフをたどってその配列変数の実態が宣言されている関数までさかのぼる。コンパイラはそこに TCW 再利用型通信の設定命令を生成し、ID をパラメータで渡すようにする。(図 4)

ループの分割は次のように最適化する。PC クラスタの各ノードで通信に関する履歴を回収する際に、各ループの反復毎に発生した通信量を調べる。この情報をメインマシンに集める。メインマシンでは各反復を

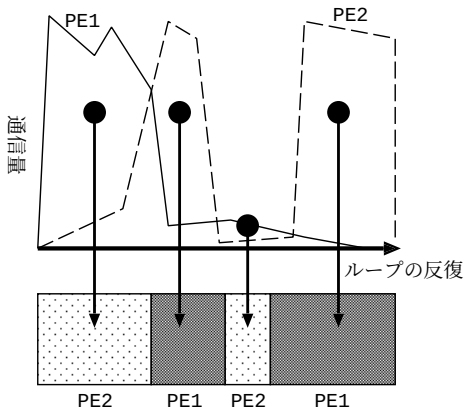


図 5　ループの分割

通信量の少なかったプロセッサに優先的に割り当てる。このとき、反復数は単純に均等になるようにする。(図 5)

我々が実装したコンパイラは現在チューニング中である。現在 NAS Parallel Benchmarks[4] を用いた実行時間の測定、商用コンパイラとの比較を予定している。

5. 予備実験

我々は実装したコンパイラを用いて予備実験をおこなった。測定の目的は、プロセッサ対実行時間を確認することと、コンパイル時間を確認することである。本方式はコンパイル中にプログラムの一部を実行するので、コンパイル時間が増加する。そこで、これを確認する。実行環境は Pilot3 上の 1〜8 プロセッサである。並列化コンパイルは PentiumIII733Mhz、メモリ 128Mbytes、LinuxRedHat7.1 上で構成される PC クラスタでおこなった。用いたノード数は並列計算機で実行時に用いるプロセッサ数と同じ台数である。なお、バックエンドには日立製の Fortran90 コンパイラ (バージョン 02-06-/C + 02-06-XF + 02-06-XJ) を用いた。実験に用いたベンチマークは NAS Parallel Benchmarks の HPF 版[5] から FT のクラス A である。

表 1　表 1. 実行時間とコンパイル時間 (NAS FT class-A)

プロセッサ数	1	4	8
実行時間 (sec)	5.0	5.3	3.7
コンパイル時間 (sec)	23.4	52.6	126.7

予備実験の結果は表 1 のようになった。ここでは本方式についてコンパイル時間と実行時間を測定しただけである。また、十分なチューニングが完了していないので、参考程度の実験である。プロセッサ台数 1 に関しては本方式のコンパイラで並列化する必要がない。このため、このコンパイル時間はバックエンドに用いたコンパイラの処理時間である。プロセッサ台数 4、8 に関しては本方式による並列化とバックエンドに用いたコンパイラの処理時間の両方を含む。

今回の予備実験では次の二点の問題が見られた。実行時間に比べコンパイル時間が長い点と、プロセッサ台数が増加する事でコンパイル時間が長くなる点である。現在の実装ではまだチューニングが進んでいない。これらの問題の原因は次の理由にある。

まず、実行時間に比べコンパイル時間が長いのは、ベンチマークの性質と実装上のチューニング状況に原因がある。用いたベンチマークは比較的大きな計算を 6 回繰り返す。このベンチマークは同じ形の通信パターンを 6 回繰り返す。今回の実験では、最適化にかかる時間を回収するのに繰返し回数が十分ではなかった。我々が前に行なった実験[8] ではコンパイル時間を実行時間で回収する事ができた。この時のベンチマークは pde1(GENESIS Distributed Memory Benchmarks) の HPF 版[5] を用いた。このベンチマークは FT に比べて軽い計算を 1000 回繰り返す。

次に、プロセッサ台数が増加する事でコンパイル時間が長くなる点である。これはメインマシンの処理の集中が原因である。(図 2) 我々の実装は、クラスタマシンが通信パターンを解析して、その結果をメインマシンに集める。この問題は、メインマシンの仕事の集中を避けることにより改善できると考えられる。このためにはメインマシンがおこなっている処理の一部を、クラスタマシンに協調的に処理させる必要がある。

6. 関連研究

我々は同様に動的な解析を用いた並列化コンパイラを実装した[8]。その研究では RDMA のブロックストライドの利用と TCW 再利用型通信の利用を行った。その研究ではインスペクタに相当する部分の実行を 1CPU の計算機で行った。このため、全てのプロセッサが同じパターンで通信するプログラムのみがコンパイルの対象だった。

TEA Expert[2] は動的な情報を用いて並列プログラムを最適化するツールである。我々の研究と同様に、実行時の情報を用い、あらためてコードを生成する方式である。TEA Expert の最適化の対象はメモリアクセスやデータ分割に関するものである。

Voss と Eigenmann[9] も実行時情報を用いた最適化の研究をおこなっている。本研究との違いは最適化の対象がことなる点と、実行時情報を用いて最適化

したコードをあらためて生成しない点である。Voss と Eigenmann の研究ではループ並列化の可否、tilling、serializing を対象にしている。また、Diniz と Rinard[11]の研究では実行時情報を用いた同期の最適化を取り扱っている。並列計算ではないが、ATLAS[12]や、窪田ら[13]による Java の実行時最適化では、実行時情報を用いたループの最適化を研究している。

7. まとめ

我々は、現在開発中の CP-PACS/Pilot3 用の並列化コンパイラについて述べた。このコンパイラは通信機構の RDMA の機能を効果的に使うために、動的な解析を用いた。この解析はインスペクタ・エグゼキュータ方式の基づく。また、解析結果をコンパイラに戻すため、生成されるコードには動的な手法に発生しがちなオーバーヘッドを発生しない。

現在の実装状況ではチューニングが完了していないため、今回は予備実験のみおこなった。現在より詳しい実験を行う準備をしている。より多くのベンチマークとより大規模なプロセッサ数で実験をする予定である。

我々が実装したコンパイラには適さないプログラムがある。通信パターンが通信によって明らかになるプログラムと反復解法で反復のたびに通信パターンが変る場合である。通信パターンが通信によって明らかになるプログラムは現在の実装ではコンパイルできない。これは、配列変数の添え字がプロセッサに分散配置される配列によって決定される場合に発生する可能性がある。例えば圧縮した大規模疎行列の掛算などが含まれる。この問題は、コンパイラが PC クラスタ上で動的な通信の解析をおこなう際に実際に通信を行うことで解決できる。反復解法で反復のたびに通信パターンが変る場合は、コンパイルは可能である。しかし、コンパイルに際してプログラムの全てを実行しなければならない。このため、実行時間とコンパイル時間を足すと処理時間の短縮が得られない。この状況に対処するには実行時にプログラムが自己書き換えする方法が考えられる。以上二点が今後の課題である。

参考文献

1) Charles Koelbel, Piyush Mehrotra: Compiling Global Name-Space Parallel Loops for Distributed Execution *IEEE Trans. on parallel and distributed systems* pp.440 − pp.451 (1991)
2) 佐藤 三久, 建部 修見, 関口 智嗣, 朴 泰祐: 自動適応並列プログラム性能最適化ツール TEA Expert 情報処理学会 *HPC* 研究会報告 pp.13 − 18 (1998)
3) High Performance Fortran Forum, 富士通株式会社, 株式会社日立製作所, 日本電気株式会社 訳: High Performance Fortran 2.0 公式マニュアル シュプリンガーフェアラーク東京 (1999)
4) NAS Parallel Benchmarks http://www.nas.nasa.gov/Software/NPB/
5) Portland Group ftp://ftp.pgroup.com/pub/HPF/examples/
6) 計算物理学研究センター http://www.rccp.tsukuba.ac.jp
7) 岩崎 洋一, 中澤 喜三郎, ほか: 計算物理学と超並列計算機 - CP-PACS 計画 情報処理 *vol.37, no.1, pp.10 − 42 (1996)*
8) 横田 大輔, 千葉 滋, 板野 肯三: 通信機構に合わせた最適化をおこなう並列化コンパイラ 情報処理学会論文誌 *vol.42, no.4, pp.860 − 867 (2001)*
9) Michael Voss, Rudolf Eigenmann: Dynamically adaptive parallel programs *In proc.of Int'l Symposium on Highperformance Computing*, springer, May. LNCS1615, pp.109 − 120 (1999)
10) Daniel Weise, Roger F. Crew, Michael Ernst, Bjarne Steensgaard: Value Dependence Graphs: Representation Without Taxation *ACM SIGPLAN POPL* pp.297 − 310 (1994)
11) Pedro Diniz, Martin Rinard: Dynamically feedback:An effective technique for adaptive computing *ACM SIGPLAN PLDI* pp.71 − 84 (1997)
12) R.Clint Whaley, Jack J.Dongarra: Automatically tuned linear algebra software *In proc. of SC*, Novenmber. Orlando, FL. (1998)
13) 窪田 昌史, 坂口 陽祐, 津田 孝夫: 実行時情報を用いた最適化手法 情報処理学会 *HPC* 研究会報告 *pp.23 − 28 (1999)*