ハイパフォーマンス
コンピューティング 73−11
（1998. 10. 9）

メタ計算系上のコンパイラインターフェイス

佐藤周行
東京大学 大型計算機センター

要旨:

メタコンピューティングとは分散する計算リソースを統合して全体として一つにみせる計算形態である。このためには従来の分散計算技術に加えて、プログラマから見た統一ビューを提供する技術が必要になる。
本研究では数値計算ライブラリを計算リソースとして分散環境のなかで利用することを考える。具体的に外部環境に存在する高速計算機資源を数値計算ライブラリのインターフェイスでトランスペアレントに利用できるメタ言語処理系を構築した。ここでは外部分散環境とのインタラクションを実行時環境の提供とコンパイラディレクティブの形で提供している。

MetaCompiler: A Compiler System on a MetaComputing Envionment

SATO Hiroyuki
Computer Centre, The University of Tokyo.

Abstract:

Metacomputer is an environment of computing in which a number of distributed resources are organized as the uniform computing resource. In addition to the conventional technologies of distributed environment, a unified viewpoint from a programmer is required in order to fully utilize the resource of metacomputer. In this paper, we construct a metacomputing environment of numerical library routines together with the C and Fortran interfaces for transparently utilizing the numerical library routines scatterd in the distributed environment. In our implementation, we provide the directory service of library names, the runtime interface, and the compiler directives for the interaction with the metacomputer environment.

1 Introduction

メタコンピューティングとは分散する計算リソースを統合して全体として一つにみせる計算形態である。このためには従来の分散計算技術に加えて、プログラマから見た統一ビューを提供する技術が必要になる。近年の分散計算技術の発展により、これらメタコンピューティングのための要素技術としては十分なものが揃ってきている。

分散環境の中で利用可能なものをリソースとして抽象化することは分散計算の概念で始めて可能になった。リソースとしてはファイルシステムが代表的なものである。近年、リソースの概念が拡大され一般的な計算リソースが分散環境の中で「資源」として利用可能になった。この場合分散環境の中での名前空間をローカルな環境にどう統合するかは重要な問題である。解決方法は大きく分けて2種類ある。ひとつは全体のなかでユニークに定まる名前つけのコンベンションを決めることである。インターネットドメインネームやWeb SystemでのURLではこの方法がとられている。この方法はリソースを外部のものと認識しさえすれば全体の中でユニークに定まる名前によってアクセスできる利点がある。もうひとつは外部のリソース名をローカルな環境の中にマッピングすることである。NFSではこの方法がとられている。この方法で

は従来のローカルな環境でトランスペアレントに外部リソースが利用できる利点がある。

分散計算技術のうち、リソースのトランスペアレンシは現在重要な要素になっている。これによりリソースの使用者は分散環境のどこにリソースが存在しているかを意識せずにすますことができる。この実装のためにディレクトリサービスが標準的なインターフェイスとなった。ディレクトリサービスの標準としてはISOが定義されている他、実装固有のものが数多く存在するが、本質がリソースのデータベースとその検索手段の提供にあることはどの実装でもかわらない。

この研究では 数値計算ライブラリを計算リソースとして分散環境のなかで利用することを考える。数値計算ライブラリは、並列計算機など通常の意味でいう高速計算のための計算リソースの、プログラマから見たインターフェイスになることが多い。従来のRPCのインターフェイスの切り方の伝統からいっても数値計算ライブラリでインターフェイスを切るのは良い選択である。また、数値計算ライブラリとして各プラットフォームで高性能なものが提供されていること、さらにBLASやLAPACKなど、いわゆるNetlibに関係する共通インターフェイスを持つ数値計算ライブラリが開発され続けていることもこの選択の良さを支持するものである。また各所に点在する高性能計算機をLANやインターネットで接続した環境はまさしく分散計算環境である。高速計算のための計算機を計算リソースとして理解することが近年重要視されてきた背景には、ますます増大する計算需要をますます多く設置されてきた高速計算機を結合してまかなうことが充分リーズナブルであると認識されてきたことがある。NetsolveやNinfはまさしくこの環境をターゲットとして分散計算環境を提供するものである。

ここでは、特に言語処理系(コンパイラ)をターゲットとして計算リソースをトランスペアレントに利用できる体系を提案する。これによって、ローカルな環境で書いたプログラムを数値計算ライブラリのあるプラットフォームを意識せずにそのまま分散環境で実行することができる。今回提案するシステムを以後 メタコンパイラという。メタコンパイラのためには、外部の計算リソースを認識すること、さらにそれらをローカルなプログラミングビューの中に埋め込むこと、コンパイル、リンクの各フェーズで外部環境とインタラクションをとること、また、時間によって変化する計算リソースの能力に対応するために外部分散環境の状況を監視することも求められる。従来のシステムのうち、NetsolveやNinfはこの要請をほとんど満たしているということができる。ただし、外部計算リソースの利用のためには陽に関数を呼び出さなければならず、この点でトランスペアレンシに欠ける。これはローカルな環境で開発したプログラムの書き換えを意味し、メタコンパイラの基本的な要請を満足しない。

我々は本論文でメタコンパイラのソフトウェアアーキテクチャについて検討するとともに、今回の実装について述べる。今回のシステム構築のためにNetSolveのサーバのバージョン管理機能およびagentのディレクトリサービス機能を強化し、リンカレベルで外部リソースを取り込む言語処理系を開発した。さらにコンパイラにも性能チューニングのための#pragmaディレクティブを追加した。

以下本論文の構成を述べる。2節は関連研究を述べる。3節では今回の実装に用いたメタ計算系NetSolveをサーベイする。4節ではメタコンパイラのソフトウェアアーキテクチャについて述べる。概念とディレクトリサービス、性能のチューニング、並列性について検討する。5節では今回実装したメタコンパイラの概略について述べる。6節では簡単なまとめを与える。

2 Related Work

メタコンピューティングの技術や概念はRPCの開発とともにあったが、具体的な形として現れたのは最近の話である。メタコンピューティングとして概念が整理される前にはheterogeneous環境でのプログラミングビューをを統合化する研究がさかんになされた。Jade[4]はその中で言語レベルのインターフェイスを提供した代表的なものである。また、PVMを使って環境下のマシンのリソースを使用する研究がさかんだった。PVMはheterogeneous環境の統合ビューを提供する初期のインターフェイスとして理解することができる。

NetSolve[1]やNinf[9]は分散している計算リソースを統合するソフトウェアである。Foxらのグループは HTTPを使って計算リソースをまとめる仮想的なマシンWWVM[3]を提案した。村岡らのグループはJavaを用いて計算リソースを移動可能なものにし、計算サーバに割り当てることのできるメタコンピュータシステムMCを提案した[5]。さらに最近のプロジェクトとしてLEGION[7]やGLOBUS[8]が大規模に展開されている。また、メタコンピュータシステムではな

いが問題解決のためにネットワークを利用してリソースを統合するプロジェクトにNEOS[10]などが知られている。

NetSolveとNinfは計算サーバに計算依頼をする点で概念を同じくするものであるが、計算リソースの特定の点で異なるものが一部にある。NetSolveは 関数名を拡張した概念である Problem と呼ばれるインターフェイスを、Ninfは関数名とロケーションを提供している。ロケーションの指定がNetSolveでは不要でAgentとよばれるデーモンがProblem名からロケーションを引くディレクトリサーバとして機能している。一方、Ninfではロケーションの指定にURLが使えるなど、より一般的なロケーション指定ができるようになっている。

3 Survey of NetSolve

メタコンパイラのソフトウェアアーキテクチャは充分一般的なものであるがRPCとして今回用いたメタ計算系 NetSolve のソフトウェアアーキテクチャに依存するところがある。本節では NetSolve のアーキテクチャを簡単にまとめる。メタコンパイラの求めるサービスとの齟齬の議論は次節に含まれる。

3.1 プログラミングビュー

プログラムではインターフェイスの関数として次を意識するだけで良い。

```
netsl(char *problem,...)
netslnb(char *problem,...)
```

`netsl`はブロッキング、`netslnb`はノンブロッキングである。プログラム言語としてはC, Fortran, Java, Matlabが用意されている。

計算リソースはProblemで記述される。具体的には関数名を文字列で記述したものである。計算サーバのロケーションはリソースを構成する要素として入っていない。

3.2 システムの概要

プログラミングビューとしては関数 `netsl` をインターフェイスとして持つ。`netsl`はクライアントを呼びだし、次の処理を行なう。

1. Agentと呼ばれる特殊なサーバに Problem をキーとして検索リクエストを出す。2. AgentはProblemに対して利用可能なサーバのリストを返す。3. クライアントはサーバのリストから一つを選んで計算をリクエストする。4. サーバから結果が返る。

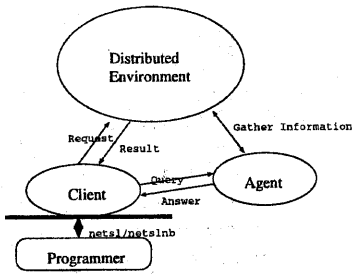

図 1: Software Architecture of NetSolve

前半はAgentがディレクトリサービスを行なっていることを意味している。後半はクライアントがNetSolveに対してRPCをかけていることを意味している。

以上から、NetSolveは標準的な分散技術の概念を用いてシステムを構築していることがわかる。

NetSolveではサーバの選択にマシンの計算能力の情報を利用しているが、この分野でもマシンの計算能力とネットワークの状態を総合して選択を行なう研究がある[6]。これと比べるとNetSolveのアルゴリズムは素朴である。さらにNetSolveで考慮しているのはFault Torelanceであるが、これもpingをかけているだけでどちらも技術的に他と大きく変わるものがあるわけではない。つまり、今回のシステム構築ではNetSolveを利用したが、特にこれに依存しているわけではなく、汎用的なインターフェイスを用意することが可能であることがわかる。

4 Software Architecture

4.1 MetaCompiler

ここでいうメタコンパイラは以下の性質を持つメタコンピューティングシステムである。

- ユーザからはローカルなコンパイラシステムとして見える。
- ライブラリ関数に代表される計算リソースとして外部分散環境にあるものを利用できる。
- コンパイラシステムや実行時ルーチンは外部分散環境とインタラクションをとって、外部計算リ

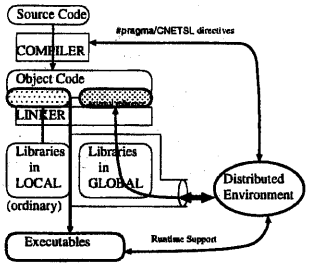

図 2: Software Architecture of MetaCompiler

ソースを利用する。

メタコンパイラの構築のためには以下が具体的に問題になる。以下、今回構築したメタコンパイラの概要を述べ、問題点の解決方法を具体的に述べる。

1. 計算リソースを利用する時のパラメタ渡し
2. 計算リソースの名前の管理と外部とのインタラクションのためのディレクトリサーバ
3. パフォーマンスチューニングのためのコンパイラの動作と外部環境とのインタラクション
4. コンパイラでの並列性のサポート
5. サーバの選択を代表的なものとするスケジューリング

4.2 Overview

われわれの構築したメタコンパイラシステムの概略を図 2に示す。

提供するのは C と Fortran のメタコンパイラシステムである。ローカルな環境でプログラミングしたコードを変更なしで外部分散環境のもとでコンパイルできる。大問題になっている dusty deck のプログラムが数値計算ライブラリをインターフェイスとしているならば、変更なしに外部計算リソースを利用することができる。

今回、外部環境とのインタラクションのために NetSolve のサーバと agent を利用した。パフォーマンスチューニングとして、サーバの選択のためのヒントと NetSolve 自身が提供する外部計算のブロッキング計算とノンブロッキング計算の切替えが directive でできるようにした。さらに、外部リソースと同じものがローカルなライブラリシステムとして提供されている場合、どちらを優先するかはコンパイラのコマンドレベルで制御できる。

4.3 Parameter Passing

今回のメタコンパイラは NetSolve の上に構築されるので、引数渡しは NetSolve の方法に従う。

外部計算リソースを利用する時に引数渡しの方法は、間接参照されているオブジェクトの扱いが大きな問題にある。プリミティブなものしか許さない従来型の RPC から参照されているオブジェクトグラフ全体を渡す JAVA RMI まで多くの方法が提案されている。その 2 つの中間に限られた扱いをゆるす方法が多く提案されている。NetSolve を RPC の提供であると考えると、そこでの解決方法は 1 次元ベクトルと 2 次元配列を許すものである。これは数値計算ライブラリを対象にしたことを考えると現実的な解である。

4.4 Name Resolution and Directory Service

NetSolve で提供されるライブラリ関数が外部リソースと認識される。具体的にはリンクのフェーズで外部環境と接続される。ライブラリ関数のうち、何を接続するのかは Agent をディレクトリサーバとして用いて判断する。Agent からとってくる情報は以下の通りである。

Name	Parameter Type	Result Type	Server List

これをもとにして、メタコンパイラ側ではインターフェイススタブを生成する。

4.5 Directory Server

メタコンパイラでは Agent をディレクトリサーバとして利用しつつ、計算性能とネットワーク距離の 2 つのサーバの性質をパラメタとしてクライアントからサーバを陽に選択できるようにした。また Agent からの情報を内部にキャッシングできることも重要である。Agent を利用する時にメタコンパイラ側のリクエストとの齟齬が大別して 2 種類あり、プロトタイプ実装を発展させるときに NetSolve の本格的な組み直し、または独自コーディングを必要とする。

NetSolve で計算リソースは Problem と呼ばれるライブラリ関数であり、サーバの位置はリソースとし

て認識されない。立場を変えていえば、認識させないのがNetSolveのユーザビューであると言うことができる。これを実際の環境に実装するためにagentではProblemに対してそれを持っているサーバの情報を内部で持ち、クライアントからの問題をkeyにしたqueryに対してサーバのリストを返すようになっている。その中で最良とagentが考えるサーバがfirst-serverとなっている。この選択はagentが内部の論理で行なっている。

さらに、リソースやサーバの種類の変更もAgentの内部で処理され、クライアントには(関係ないものとして)通知されないが、メタコンパイラのインターフェイスではこれも知る必要がある。具体的に言えば、Problemとサーバのバージョン管理情報が必要である。

以上の意味でNetSolveのagentのもつディレクトリサービスはメタコンパイラの求めるものと微妙に食い違う。このためにagentとサーバの機能強化を行なった。

4.6 Interface with Conventional Compiler System

MetaCompilerにおいて、リソースのロケーションをトランスペアレントにするために従来のコンパイラシステムとリンカでインターフェイスを切った。

ここではリソースは関数名の形で表現されるので、外部環境の提供する関数名を集めてアーカイブの形にしておく。これはAgentとのやりとりをローカルなシステムのアーカイブに保存することでありリソースのキャッシングである。

4.7 Caching

外部環境から提供されるライブラリ関数数は一般に手に追える範囲を超えている。したがって外部環境のライブラリ関数すべてをアーカイブの形でのインターフェイスで提供することはリンカの性能を大きく超える。すなわち、ある範囲で区切ってその中はアーカイブの形でのインターフェイスを提供し、その外にあるものはベースのメタ環境システムと実行時にインタラクションをとらなければならない。

アーカイブの形でのインターフェイスは、ディレクトリサーバの情報から部分評価によってメタ計算系のリソース名の解決(directory search)と、サーバに対してのRPCを行なうコードを生成しておくものである。一方、実行時にインタラクションをとる場合は、メタ計算系上での計算を動的に実行するものである。

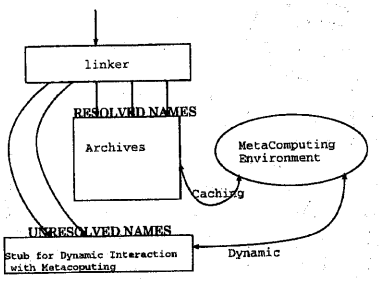

図 3: Caching of Resource Names

これらから内部に対してのアーカイブ形式のインターフェイスは外部に対するメタ計算系のインターフェイスに対するキャッシングであると結論づけることができる。キャッシングはさまざまなソフトウェアシステムでの有効性が確認されているが、特にWebシステムで効果があることが実証されている。

MetaCompilerでの動的なメタコンパイラとのインタラクションのために、リンカにフックをかけておき、ライブラリ名が解決不可能なものに対してメタ計算系での計算を行なうコードを自動的に挿入することにした(図3参照)。

4.8 Performance Tuning

前述の情報と関係するが、サーバの選択のロジックはメタコンパイラで制御したい場合がある。今回 ディレクティブで以下の機能を提供する。

heavy	関数が重い計算を必要とするかどうか
block	関数をブロッキングするか、 ノンブロッキングで実行するか
server	サーバの特定

heavyかどうかはサーバの選択のヒントとして使える。ブロッキング処理するかどうかは[2]によれば並列処理に有益である。

4.9 Parallelism

ブロッキング/ノンブロッキング呼び出しによる並列性を利用することができる。ブロックするかどうかはおおもとのスレッドをブロックするかどうかで解決できる。インターフェイスが関数/サブルーチンレベ

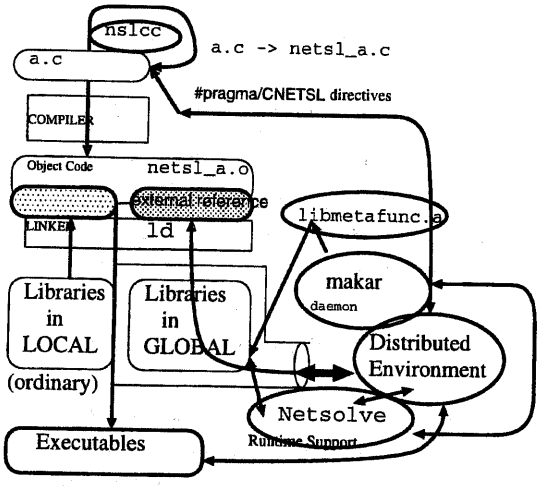

図 4: 実装の概要

ルで切られていることからコンパイラ側の解析に問題があるので NetSolve で提供する並列性を残す。

4.10 Server Selection and Scheduling

NetSolve の提供するサーバ選択の方針に関係するパラメタは性能、負荷状態、および接続去れているネットワークの bandwidth と latency である。

我々はそれに加えて 計算そのものが重い場合に性能に比重をおくこと、計算が軽い場合により近くのネットワークに接続されていることを重視することの2つのオプションを用意した。

4.11 Hierarchical Directory of Resources

Directory を管理している agent で、名前が resolve できなかった場合、外部の agent に接続してさらに resolve する。NetSolve は resolution が階層的にはできないので、通常の agent の他に外部のより大きなデータベースを持っている agent に接続できることでこれを解決している。

5 Implementation

図 4は前節で述べたソフトウェアアーキテクチャにしたがって今回の実装を図示したものである。

6 Conclusion

本論文ではメタコンパイラのソフトウェアアーキテクチャについて述べた。概念とディレクトリサービス、性能のチューニング、並列性について検討した。

本研究は 一部 日本学術振興会未来開拓事業 (JSPS-RFTF96P00505) の援助を受けた。

参考文献

[1] Casanova, H., and Dongarra, J.: "Netsolve: A network server for solving computational science problems," Supercomputing 96, 1996.

[2] Casanova, H., Dongarra, J. and Seymour K.: "Client User's Guide to NetSolve," draft, 1997.

[3] Dincer, K. and Fox, G.C.: "Building a World-Wide Virtual Machine Based on Web and HPCC Technologies," Supercomputing 96, 1996.

[4] Rinard, M.C., Scales, D.J. and Lam, M.s.: "Jade:a high-level, machine-independent language for parallel programming," Supercomputing 92, 1992, pp. 245–256.

[5] 首藤一幸, 菅原健一, 浜中征志郎, 村岡洋一: "分散環境を対象とした並列プログラミング環境 MC," 97-HPC-67-16, 1997.

[6] Wolski, R., Spring, N. and Peterson, C.: "Implementing a Performance Forecasting System for Metacomputing: The Network Weather service," Supercomputing 97, 1997.

[7] http://www.cs.virginia.edu/legion

[8] http://www.globus.org

[9] http://www.ninf.etl.go.jp

[10] http://www-c.mcs.anl.gov/home/otc