ハイパフォーマンス
コンピューティング 90－10
（２００２．５．２７）

OpenGR コンパイラの設計および開発

平野基孝[†] 佐藤三久[††]
田中良夫[†††] 関口智嗣[†††]

我々は Grid 環境に対応した Ninf-G などの RPC システムを並列実行エンジンとする並列化指示文機構 OpenGR と，その処理系である OpenGR コンパイラを，OpenMP コンパイラ Omni をベースとして開発している．本報告書では，OpenGR 指示文機構の概要と，OpenGR コンパイラの現状の実装に関して述べる．

Design and implementation of OpenGR compiler

Motonori Hirano,[†] Mitsuhisa Sato,[††] Yoshio Tanaka[†††]
and Satoshi Sekiguchi [†††]

We introduce a compiler pragma set called OpenGR, using Grid-enabled RPC system such as Ninf-G as parallel execution mechanism, and are developping its compiler system OpenGR complier, based on Omni OpenMP compiler system. In this paper, we refer an overview of the OpenGR pragma and developping status of the compiler system.

はじめに

Grid 環境の普及に伴い，従来 LAN 等の狭い環境で使用されてきた RPC を，Grid 環境に対応させようという研究が行われてきている．Grid 環境に対応した RPC システムは GridRPC[5] システムと呼ばれ，Ninf-G[10]，NetSolve[1] 等のシステムが実装されている．

GridRPC システムは，通常の RPC システム同様，PRC 呼び出し側と RPC 被呼び出し側の 2 つの実行モジュールを持つ．特に GridRPC システムでは，被呼び出し側モジュール (以下サーバスタブモジュール) が被呼び出し側サイトに既にインストール済である場合には，呼び出し側モジュールは GridRPC システムのクライアント側 API を使用して，ユーザが呼び出したいモジュールを指定するだけで，GridRPC を使うプログラムを作成することが可能である．

一方，ユーザが呼び出したいモジュールが被呼び出しサイトにインストールされてない場合，ユーザは自分自身の手で被呼び出しモジュールを作成し，さらにそのサイトの GridRPC システムの利用手順に従ってインストール (以下サーバスタブシッピング) しなければならない．特にユーザが既存のコードを GridRPC 化する場合，ユーザはサーバスタブにしたい部分をオリジナルのソースプログラムより抽出し，GridRPC システムの提供する手段でサーバスタブモジュール化する必要がある．更に，Grid 環境において被呼び出しサイトにサーバスタブシッピングするためには，被呼び出しサイトプラットフォームに適応したオブジェクトフォーマットでサーバスタブモジュールを生成，リモートサイトへのネットワーク (特に WAN) を介したモジュールのコピー等，使用するための手順が煩雑であり，ユーザの負担が小さいものであるとは言いがたい．

そこで我々は，OpenMP コンパイラのような，compiler directive(pragma) ベースで，既存のプログラムをほとんど改編することなく GirdRPC システムを容易に使用できる環境の提供を目的として，コンパイラ指示文機構 OpenGR とその処理系である OpenGR コンパイラを設計開発している．OpenGR コンパイラおよび OpenGR 指示文により，ユーザは，

- ソースコード内のどの関数をサーバスタブモジュールにするかを指定する．指定された関数，部位は，OpenGR コンパイラにより，サーバスタブシッピングモジュールとして生成され，OpenGR コンパイラシステムが提供する周辺コマンドにより，容易にサーバスタブシッピングできるようになる．
- 指定したサーバスタブモジュールを，どこで使用するかを指定する．OpenGR コンパイラは，指定された部位 (ブロック) 内

† 株式会社 SRA
Software Research Associates, Inc.
†† 筑波大学 電子・情報学系 計算物理学研究センタ
Institute of Information Science and Electronics / Center for Computational Physics, University of Tsukuba
††† 産業技術総合研究所 グリッド研究センタ
Grid Technology Research Center, National Institute of Advanced Industrial Science and Technology

に存在する，サーバスタブモジュールとして定義のある関数を，GridRPC システム API への呼び出しに変換する．

の作業をおこなうだけで GridRPC システムを使用でき，直接 GridRPC システムを使用した開発を行う場合に比べ，負担の軽減が図れる．

本稿の構成

本稿ではまず **1** 節で Ninf-G を例にとり GridRPC システムについて概説する．次に **2** 節で，OpenGR 指示文およびコンパイラの設計方針について述べる．次に **3** 節で OpenGR 指示文について概説する．次に **4** 節で OpenGR コンパイラの開発状況および今後の予定に関して報告する．最後に **5** 節で関連研究を紹介する．

1. GridRPC システム

GridRPC システムとは，特に Grid 環境での使用に対応すべく，広域に分散した計算資源への RPC を安全に行うための，通信相手 (peer) 間の相互認証，通信内容の秘匿化・暗号化などの，いわゆるセキュリティに関して充分配慮のなされた RPC システムである．Ninf-G は，旧実装である Ninf[6] を基に，通信層および計算資源管理部分等を Globus toolkit[3] を利用して再設計実装したものである．

Ninf-G では，ユーザは IDL と呼ばれる言語でサーバスタブプログラムを記述し☆，IDL コンパイラを用いてサーバスタブモジュールを生成する．サーバスタブモジュールには，実際に RPC 呼び出しされるサーバ関数スタブ以外に，XDR を含む RPC 引数授受のための情報も格納される．図 **1** に Ninf-G IDL 記述の例を示す．生成したサーバスタブモジュールは，IDL コンパイラの生成する Makefile 内の install ターゲットにより，通常ユーザが "`make install`" のコマンドを発行すれば，そのサイトの適切な場所にあるサーバスタブモジュールデータベースに登録され，RPC 呼び出し側からの使用が可能になる．ユーザは，使用する RPC 被呼び出しサイト毎に，IDL をコピーし，IDL コンパイラでサーバスタブモジュールを生成，"`make install`" を実行する，という手順を踏む必要がある．

任意のサイトに登録されたサーバスタブモジュールを使用するために，ユーザは Ninf-G API を用いてプログラムを開発する．図 **2** に例を示す．図 **2** 中，Ninf-G の提供する API は `grpc_*()` と記述されたもので，ランタイムライブラリの初期化，RPC 関数ハンドルの生成，RPC 呼び出し，同期機構等が用意されている．特に Sun RPC と比較して，Sun RPC の `rpcgen` がサーバ，クライアントの両サイドのスタブを生成するのに対し，Ninf-G では IDL コンパイラが生成するのはサーバスタブのみである，という特徴がある．

☆ NetSolv では PDF と呼ばれる．本稿では，これらを総称して IDL と呼ぶことにする．

```
Module pi;

Define pi_trial(IN int seed, IN long times,
                OUT long * count)
"monte carlo pi computation"
Required "pi_trial.o"
{
  long counter;
  counter = pi_trial(seed, times);
  *count = counter;
}
```

図 1: Ninf-G IDL 記述例

2. 設　　計

本稿冒頭で述べたように，OpenGR 指示文およびコンパイラは，既存のプログラムをほとんど改編することなく GridRPC 化を行える環境をユーザに提供することを目的としている．この目的のために，ベースとなる GridRPC システムが提供する機能を，

- スタブ生成および呼び出し等の RPC 基本機能
- スケジューリングメカニズム，フォルトトレランス機能，統計情報取得機能，その他 GridRPC システムとして共通もしくは特定の実装に固有な機能

の 2 種類に分けて，別のレイヤで実現することにした．この 2 種類をそれぞれ，

- RPC 基本機能は OpenGR 指示文で使用できるようにする．これにより，ユーザの既存プログラムソースコード内に追加するのは RPC システムとして一般性の高い，RPC 基本機能の使用を指定する指示文だけになり，プログラムの可視性を高く保てる．
- GridRPC システム固有の機能は，ランタイムライブラリで吸収することにする．各々の GridRPC システム用のランタイムライブラリを，OpenGR コンパイラシステムがサポートすることで，一つの OpenGR 指示文を挿入したソースプログラムは，様々な GridRPC システムをベースとして使用できるようになる．

のように実現する設計とした．この設計方針に関し，元来その GridRPC システムの提供する機能詳細に指示文でアクセスできない，という問題が考えられる．本システムの設計では，目的である容易な GridRPC 化を提供することに重きを置き，ユーザが使用する指示文規格を最小にとどめることにした．必要が生じれば，OpenMP コンパイラシステムのような，並列実行制御用低レベル API をユーザプログラム内で使用できるようにすると

```
#include "grpc.h"
#define NUM_HOSTS 2
char * hosts[] = {"brain.a02.aist.go.jp",
                  "brain.a02.aist.go.jp"};
grpc_function_handle_t handles[NUM_HOSTS];
int port = 4000;

main(int argc, char ** argv){
  double pi;
  long times, count[NUM_HOSTS], sum;
  int i;
  times = atol(argv[2]) / NUM_HOSTS;
  /* Initialize GRPC runtimes. */
  if (grpc_initialize(argv[1]) != GRPC_OK){
    exit(2);
  }
  /* Initialize handles. */
  for (i = 0; i < NUM_HOSTS; i++) {
    grpc_function_handle_init(&handles[i],
      hosts[i], port, "pi/pi_trial");
  }
  for (i = 0; i < NUM_HOSTS; i++) {
    /* Parallel non-blocking remote
       function invocation. */
    if (grpc_call_async(&handles[i], i,
          times, &count[i]) == GRPC_ERROR) {
      grpc_perror("pi_trial");
      exit(2);
    }
  }
  /* Sync. */
  if (grpc_wait_all() == GRPC_ERROR){
    grpc_perror("wait_all");
    exit(2);
  }
  /* Handler destruction. */
  for (i = 0; i < NUM_HOSTS; i++) {
    grpc_function_handle_destruct(
      &handles[i]);
  }
  /* Compute and display pi. */
  for (i = 0, sum = 0; i < NUM_HOSTS; i++) {
    sum += count[i];
  }
  pi = 4.0 *
    (sum / ((double) times * NUM_HOSTS));
  printf("PI = %f\n", pi);
  /* Finalize GRPC runtimes. */
  grpc_finalize();
}
```

図 2: Ninf-G API を用いたプログラム例

いう方式で, 詳細機能を使用するための手段を提供できる.

2.1 OpenGR 指示文の機能

指示文規格設計時には, 対象とするプログラムとして, パラメタサーチを意識したが, 特にパラメタサーチのみに特化した設計にはしておらず, 前述のように, GridRPC システムの提供する RPC 基本機能を処理系に反映することを, 指示文規格設計の指針とし,

- サーバスタブ関数の記述およびサーバスタブモジュールの生成のための付加的情報 (例えば外部リンケージ情報等) の記述
- サーバスタブ関数の同期・非同期呼び出し
- 非同期呼び出ししたサーバスタブ関数の任意の時点での同期

の機能を提供する指示文を用意することにした.

2.2 OpenGR コンパイラランタイムの機能

RPC 基本機能の実現方法 (および API) は, 各々の GridRPC システムによって異なり得り, 前述のように, GridRPC システム毎に提供する機能の詳細も異なり得る. OpenGR コンパイラランタイムは, これらの差異を吸収するためのレイヤとして機能し, 特にコンパイラフロントエンド, バックエンドに対し, 各々の GridRPC システムに依存しない, GridRPC システムライブラリ初期化, RPC 呼び出し, 同期機構のための関数を提供する. スケジューリングメカニズム, フォルトトレランス機能, 統計情報取得機能等, GridRPC システム間で実現方法が大きく異なっていたり, 実装がない機能等は, OpenGR コンパイラランタイムの開発者が, 各 GridRPC システムの実装に合わせて適切に処理されなければならないものとした.

2.3 OpenMP との協調動作

GridRPC システムを容易に使用可能にするという目的の中には, 既に OpenMP 指示文によって主に SMP 環境での並列化がなされているソースプログラムも対象としたいという要求が含まれている. OpenMP 指示文によりある関数が並列化されていた場合, その関数を OpenGR 指示文で GridRPC 用サーバスタブ化できれば, SMP 環境向けにチューニングされた関数を Grid 環境から RPC 呼び出しできるようになる. これを容易に実現する最善の方法は, OpenMP 指示文と OpenGR 指示文の双方を解するコンパイラシステムの開発にあると考え, OpenGR 指示文による RPC ベースの並列化と, OpenMP 指示文による並列化が相互に悪影響を与えないことも設計方針の一つとした. また、実装のベースとして、オープンソースで公開されている OpenMP コンパイラシステム Omni OpenMP コンパイラを使用することにした。

3. OpenGR 指示文

以下, C 言語用 OpenGR 指示文について概説する.

OpenGR 指示文は, 以下の形式で使用する.

```
#pragma ogr 指示文 指示文引数 ...
```

OpenGR 指示文は, 以下の種類に分けられる.

- サーバ関数スタブ定義指示文 define_func
- サーバスタブモジュール生成指示文 gen_stub, start_func_body, end_func_body
- RPC 呼び出し指示文 call, call_async
- 非同期 RPC 用同期指示文 wait_async

図 3 に, OpenGR 指示文, OpenMP 指示文を使用したソースプログラム例を示す.

3.1 define_func 指示文

define_func 指示文は, サーバ関数スタブの定義に使用する. 以下に使用例を示す.

```
#pragma ogr define_func foo(IN double a[N], IN double b[N], OUT double c[N], IN int N)
```

この例では,foo という関数を,double a[N], b[N], int N を入力用引数, double c[N] を出力用引数として定義している. C 言語でのプロトタイプ宣言に, Ninf-G IDL 用のサーバスタブ引数仕様を加えた形式であり, 主に GridRPC システム用 IDL でのサーバ関数スタブ定義部 (Ninf-G においては Define 文に相当) の生成に使用され, 以降の call, call_async 指示文でのサーバ関数スタブ呼び出し時の OpenGR コンパイラによる引数型チェックにも使用される.

3.2 gen_stub 指示文

gen_stub 指示文は, define_func 指示文により定義されたサーバ関数スタブを, サーバスタブモジュール生成するために, 言語的にはユーザに見えないが, 特にリンケージ等に必要な情報を与えるために使用する. 以下に使用例を示す.

```
#pragma ogr gen_stub foo(Module bar, ...)
```

この例では, foo(前述の例) という関数のサーバスタブモジュールを生成するための情報を指定している. "()" の内部には, 使用する GridRPC システムに応じ, GridRPC システムがサーバスタブモジュール生成に必要な情報を記述する. Ninf-G の場合, IDL に記述する Module 文, Required 文等が記述できる.

また, 既存のサーバスタブモジュールを使用する場合のために, C 言語での外部リンケージであることを示す

```
#pragma ogr define_func \
  pingpong(IN char src[N], OUT char dst[N],
           IN int N)
#pragma ogr gen_stub pingpong(Module exam)
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#pragma ogr start_func_body pingpong
#include <memory.h>
static void
pingpong(char *src, char *dst, int n);

static void
pingpong(char *src, char *dst, int n)
{
    memcpy(dst, src, n);
}
#pragma ogr end_func_body pingpong

int
main(int argc, char *argv[])
{
    char *src = NULL;
    char *dst = NULL;
    int n = -INT_MAX;
    int i;

    if (argc < 2) return 1;
    if ((n = atoi(argv[1])) <= 0)
        return 1;

    src = (char *)malloc(n);
    dst = (char *)malloc(n);
#pragma omp parallel for
    for (i = 0; i < 10; i++) {
#pragma ogr call_async pingpong
        {
            pingpong(src, dst, n);
        }
    }
#pragma ogr wait_async
    return 0;
}
```

図 3: OpenGR 指示文使用例 (含 OpenMP 指示文)

extern 文に相当する external 指示子を記述することもできる.

3.3 start_func_body, end_func_body 指示文

start_func_body, end_func_body 指示文は, プログラムソースコード内のどの部分をサーバスタブモジュールに含めるかを指定するのに使用する. 以下に使用例を示す.

```
#pragma ogr start_func_body foo
```

```
#include <stdio.h>

int bar(...)
{
    ...
}

void foo(double *a, double *b,
         double *c, int N)
{
    ...
    bar(...);
    ...
}
#pragma ogr end_func_body foo
```

OpenGR C コンパイラ (特に C 言語フロントエンド部) として, 上記 foo をサーバスタブモジュール化するためには, foo 内の外部参照をコンパイラで全て解決すれば良いので, start_func_body, end_func_body のような仕組みは必要ない. が, 特にサーバ関数スタブが C プリプロセッサを利用して, システムインクルードヘッダ内のプラットフォーム依存なマクロを参照している場合, C 言語として完結したソースコードをスタブ化するだけでは, ヘテロジニアスな環境へのサーバスタブシッピングが行えないことになる. そのため, C プリプロセッサ指令を含む, プログラムソースコード内の任意の範囲をユーザに指定してもらうという手法を取ることにした.

3.4 call, call_async 指示文

call, call_async 指示文は, define_func 指示文等で定義されたサーバ関数スタブの実際の呼び出しを指示するために使用する. call は同期呼び出し, call_async は非同期呼び出しである. 以下に使用例を示す.

```
#pragma ogr call foo
{
    if (x == ...) foo(X, Y, Z)
    else foo(X1, Y1, Z)
}
```

call, call_async 指示文は, 挿入位置直下の節 (clause) 内の, 指示文引数で示された関数名 (上記例では foo) に関し, 全ての呼び出しを, RPC 呼び出しに置換する.

call_async 指示文による非同期 RPC を行った場合, wait_async 指示文 (**3.5** 節) を明示的に使用して, ユーザの望む時点で同期をとることが可能である.

3.5 wait_async 指示文

wait_async 指示文は, call_async 指示文によって非同期呼び出しした RPC の同期を取るために使用する. 現状の仕様では, wait_async 指示文の挿入位置で, それまでに同期していない全ての非同期 RPC 呼び出しが同期される. 非同期呼び出しされた RPC を明示的に wait_async 指示文で同期しなかった場合, OpenGR コンパイラ処理系ランタイムにより, プログラム終了時もしくは exit() の明示的な呼び出し時に暗黙に同期される.

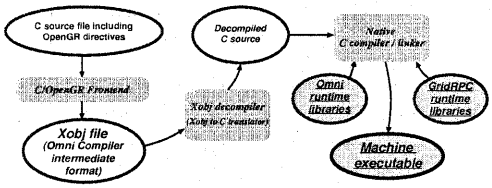

図 4: OpenGR C コンパイラの動作概要

3.6 OpenMP 指示文との組み合わせ

2.3 節 では, サーバスタブサイドの OpenMP との協調動作に関して述べたが, RPC 呼び出し側 (クライアントサイド) においても, 特に for ループ内での並列 RPC 呼び出し時に, OpenMP 指示文と OpenGR 指示文を組み合わせて使うことが有効であると考える. 以下に, OpenMP 指示文と OpenGR 指示文の組合わせの例を示し, その動作を説明する.

```
#pragma omp parallel for
for (i = 0; i < n; i++) {
#pragma ogr call foo
    {
        foo(a, b, c);
    }
}
```

for 文は OpenMP 指示文によりスレッドベースで並列実行される. したがって, foo の RPC 呼び出しは, 各スレッドで行われることになる. 特にサーバ関数スタブへの引数の大きさが巨大な場合や, 伝送路のバンド幅を RPC 呼び出し側のマシンが逐次実行では使いきれない場合などに, RPC 呼び出しそのものを並列実行させてしまうことで逐次実行時の RPC 呼び出しオーバーヘッドを隠蔽する効果が期待できる. この効果は, call 指示文を call_async 指示文に置き換えても, 同様に期待できる.

4. OpenGR コンパイラ開発現状および今後

現状, OpenGR コンパイラは, サーバスタブシッピング機構を除く OpenGR 指示文を解釈する C 言語コンパイラとして稼動し, Ninf-G で直接生成されたサーバスタブモジュールを使用することで RPC 動作が行える環境にある. 図 **4** に OpenGR C コンパイラの動作概要を示す.

OpenGR コンパイラは Omni OpenMP コンパイラを基に作られており, Omni コンパイラのソースコードツリー CVS リポジトリ内に組み込まれている. オリジナルの Omni コンパイラへの変更点は主に,

- C 言語フロントエンドへの OpenGR 指示文パーサおよび RPC 実行用コード生成部の組み込み
- RPC 実行用コードのための, Ninf-G API 呼び出しランタイムライブラリの追加
- コンパイルドライバ ogrcc の追加

である.

今後の開発予定は,

- サーバスタブシッピング機構の実装
- RPC 実行用コード生成部の改良
- 同期 (バリア) セット用指示文の設計実装
- Fortran 用 OpenGR 指示文の設計と Fortran 用 OpenGR 対応フロントエンドの開発

である.

5. 関連研究

指示文を用いて関数やサブルーチンを単位とした並列/分散プログラミングを行なう手法に関する研究は以前より行なわれている. Gross らはタスク並列に実行するべき部分を指定する指示文を HPF に導入することにより, タスク並列性とデータ並列性の両方を引き出すフレームワークを提案している[4),8)]. 指示文で指定された部分における複数のサブルーチン呼び出しをそれぞれ並列に実行する. その際, 各サブルーチンを実行すべきプロセッサやサブルーチンの入力および出力データに関する情報を指示文で指定することにより, サブルーチン間のデータ送受信に関する処理はコンパイラによって自動的に生成される. Foster らも同様なアプローチで Fortran 77 のサブセットにタスク並列処理を記述するための指示文を導入している[2)]. これらの研究はいずれもデータ並列性とタスク並列性を相補的に引き出すことによりプログラムの実行効率の向上を目指しているものであり, 対象としているアプリケーションとして異なる種類の複数の処理をパイプライン的に行なうようなものを想定しており, 我々の対象とするパラメータサーチ系のアプリケーションに対しては適さない.

首藤らは HPF にタスク並列処理のための指示文を追加し Java で実行環境とコンパイラを作成している[9)]. 利用者による RPC プログラミングや関数を単位とした fork-join 型並列処理を行なうための拡張がなされている. また, 依存解析結果に基づいて実行の条件が揃った関数が発火され, 必要なデータが揃っていない処理はロックされる機能を備えており, 非同期呼び出しを行なった際に明示的に待つ必要がない. また, コンパイラの中間表現 (抽象構文木) をシリアライズしてネットワーク越しに送ることにより, 異機種が混在した環境に対応している.

佐藤らは OpenMP の指示文とスレッドセーフな RPC 機構である OmniRPC を組み合わせることにより, タスク並列なプログラムを RPC によりグリッド上で実行するフレームワークを提案している[7)]. OpenMP 指示文を利用して複数の RPC を発行するというアプローチは同じであるが, この組み合わせでは RPC のコードもユーザが記述する必要があるのに対し, OpenGR は RPC を行なう部分を直接記述するのではなく, 指示文で指定するだけで良い.

参考文献

1) Casanova, H. and Dongarra, J. NetSolve: A Network Server for Solving Computational Science Problems, *Supercomputer Applications and High Performance Computing*, **11**, 3 (1997), 212–223, http://icl.cs.utk.edu/netsolve.
2) Foster, I., David R. Kohr, J., Krishnaiyer, R. and Choudhary, A. Double Standards: Bringing Task Parallelism to HPF via the Message Passing Interface, Supercomputing 96 (1996).
3) Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit, *Supercomputer Applications*, **11**, 2 (1997), 115–128, htpp://www.globus.org.
4) Gross, T., O'Hallaron, D. R. and Subhlok, J. Task Parallelism in a High Performance Fortran Framework, *IEEE Parallel and Distributed Technology*, **2**, 3 (1994), 16–26.
5) Matsuoka, S., Nakada, H., Sato, M. and Sekiguchi, S. Design issues of Network Enabled Server Systems for the Grid http://www.eece.unm.edu/~dbader/grid/.
6) Nakada, E., H. Design and implementations of Ninf: towards a global computing infrastructure, FGCS, **15** (1999), 649–658.
7) Sato, M., Hirano, M., Tanaka, Y. and Sekiguchi, S. OmniRPC: a Grid RPC facility for Cluster and Global Computing in OpenMP, Workshop on OpenMP Application and Tools (2001).
8) Subhlok, J. and Yang, B. A New Model for Integrated Nested Task and Data Parallel Programming, 6th SIGPLAN Symposium on Principles and Practice of Parallel Programming (1997).
9) 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, ハイパフォーマンスコンピューティング研究会, 第 97 巻, 情報処理学会 (1997).
10) 田中 良夫, 中田 秀基, 平野 基孝, 佐藤 三久, 関口 智嗣 Globus による Grid RPC システムの実装と評価, 情報処理学会ハイパフォーマンスコンピューティング研究会, **2001**, 77 (2001), 165–170.