Параллельные программы библиотеки JINRLIB

Попкова Л.В., Сапожников А.П., Сапожникова Т.Ф.

Лаборатория информационных технологий
Объединенный институт ядерных исследований

GRID 2016

Библиотека JINRLIB

JINRLIB - библиотека программ, предназначенных для решения широкого круга математических и физических задач.

Основу библиотеки составляют программы из библиотеки ДУБНА и CERNLIB. Пополнение библиотеки происходит в виде автономных программ и программных пакетов, создаваемых в основном сотрудниками ЛИТ ОИЯИ и их коллаборантами.

Программы объединяются в библиотеки объектных модулей или существуют в виде самостоятельных пакетов прикладных программ (ППП). В настоящий момент их насчитывается > 60.

Каждая программа идентифицируется уникальным индексом или именем. Для индексирования программ, включаемых в библиотеки объектных модулей, используется классификация, принятая в свое время в библиотеке ДУБНА.

Для идентификации остальных программ используется имя программы и классификация, принятая в издательском отделе ОИЯИ.

Состав библиотеки

Windows:

- GNU Fortran 77
- Compaq Visual Fortran
- Fortran Power Station
- Microsoft Fortran

JINRLIB

- Программы с использованием технологии MPI

Scientific Linux (ЦИВК) - /usr/local/lib:

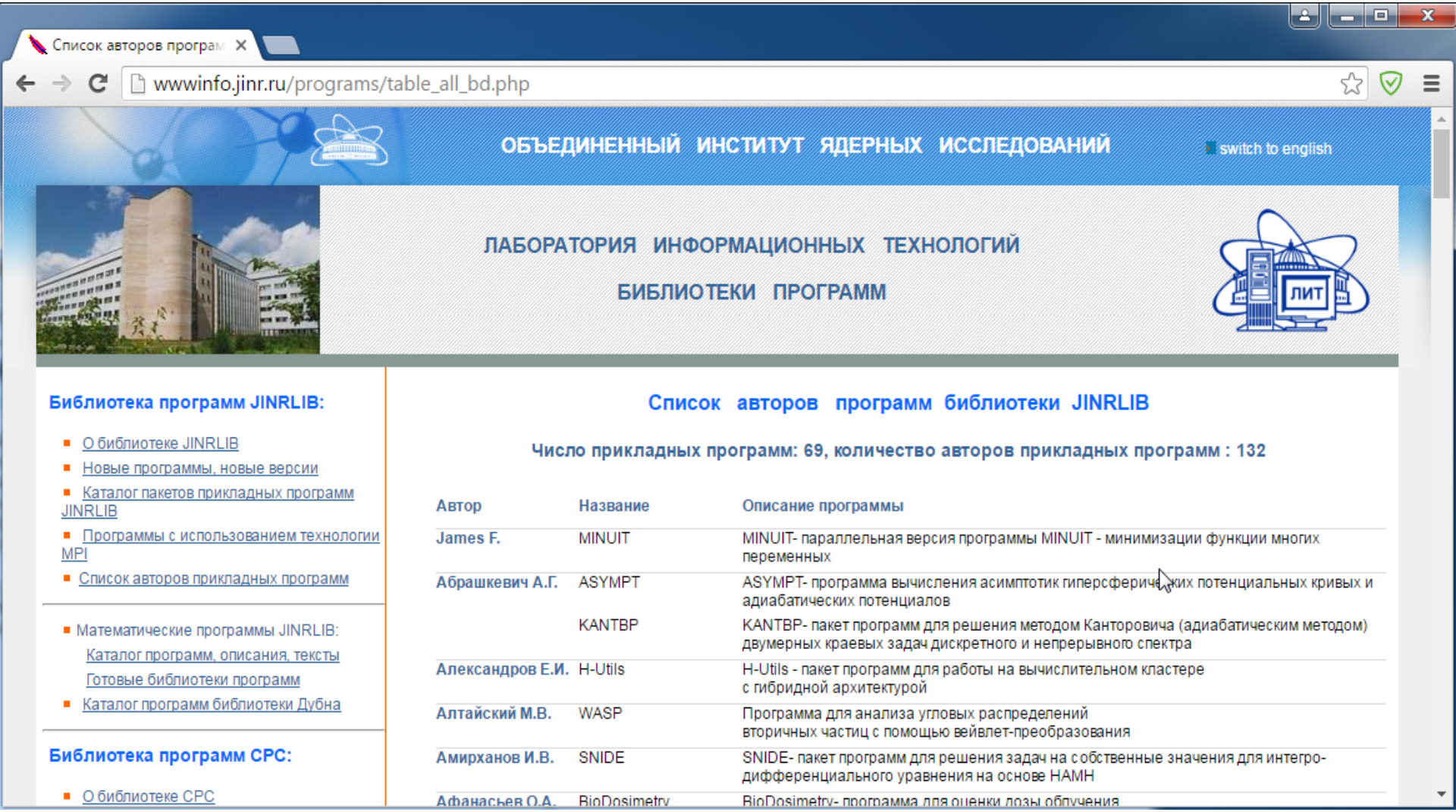
- GNU Fortran 77 (g77) - libjinr.a;
- GNU Fortran 95 (gfortran) - libjinr95.a;
- Intel Fortran (ifort) – libjinri.a.

ППП (>60):

- Теоретическая физика высоких энергий
- Теоретическая физика низких энергий
- Физика тяжелых ионов
- Автоматизация обработки экспериментальных данных
- Вычислительная математика и техника

WWW-сопровождение библиотеки

Специализированный сайт http://www.jinr.ru/programs/jinrlib
обеспечивает электронный доступ к библиотеке, где можно найти каталог, тексты и описания программ и программных пакетов, библиотеки объектных модулей.



Поддержка программирования в среде MPI

В последнее время происходит бурное развитие технологий программирования параллельных вычислений, в частности, MPI. Эта тенденция нашла свое отражение и в библиотеке JINRLIB.

Стратегия распараллеливания MPI программ:

- Библиотечная программа, подготовленная для работы в среде MPI, должна успешно работать при любом количестве NP параллельных процессов, в частности, и при NP=1.
- Возникает единый исходный текст библиотечной программы, пригодный к эксплуатации как на последовательных вычислительных системах, так и на кластерах, состоящих из большого числа процессоров.
- Необходимым условием для обеспечения такой унификации является наличие программ-заглушек, что позволяет использовать библиотечные программы и без MPI.

Эта идея, которая при работе на кластерах использует "настоящий" MPI, а при работе в однопроцессорной конфигурации - его заглушки, была успешно реализована при распараллеливании ряда программ библиотеки.

PFUMILI - параллельная версия программы минимизации нелинейных функционалов **FUMILI (1)**

Автор FUMILI – Игорь Николаевич Силин (1936-2006) был участником и соавтором большей части работ по фазовому анализу, выполненных в Объединенном Институте Ядерных Исследований.

Успехи ОИЯИ в этой области во многом обусловлены созданным им алгоритмом минимизации нелинейных функционалов. Программа И.Н.Силина FUMILI, реализующая этот алгоритм, уже более 40 лет активно используется учеными многих стран.

Опыт распараллеливания вычислительных программ (на примере **PFUMILI**) (2)

Прежде всего нужно найти наиболее времяемкие места распараллеливаемой программы. Для этого используется следующая техника:

- в исходном тексте программы выделяется некоторое количество интервалов;
- в начале и конце каждого такого N-го интервала помещаются засечки времени;
- в самом конце тестовой программы делается печать накопленной статистики.

Для этого использовалась программа Profile из библиотеки JINRLIB, которая подсчитывает количество посещений заданного интервала и суммарное время, затраченное процессом на эти посещения. Была получена следующая статистика:

```
Proc: 0 of  1. Time  1 =    0.02 sec. Ncall=    45  callF=  3340001
Proc: 0 of  1. Time  3 =    0.25 sec. Ncall=     3  callF=  3340001
Proc: 0 of  1. Time  4 =   21.14 sec. Ncall=   45  callF=  3340001
Astime:     21.48  CPU_time:    21.48  for process  0
```

Легко видеть, что наиболее времяемким местом является интервал номер 4 (subroutine SGZ), занимающий львиную долю (98.5%) общего времени теста.

Реализация (PFUMILI 3)

Результат работы SGZ есть сумма по всем заданным экспериментальным точкам (NED). Основная идея распараллеливания - поделить всю работу (NED экспериментальных точек) между всеми mpi_size процессами приблизительно поровну:

```
nn=ned/mpi_size                          ! points for 1 process
n1=1+mpi_rank*nn                         ! 1-st point
n2=n1+nn                                 ! last point
if(mpi_rank.eq.mpi_size-1) n2=ned
```

После чего цикл по всем экспериментальным точкам заменяется на более короткий. Поскольку этот цикл является самым внешним циклом, распределение его работы между процессами-исполнителями следует признать хорошим распараллеливанием.

Кроме того, чтобы поделить вычислительную работу и объединить полученные частные результаты, надо совершить межпроцессные обмены информацией.

Достигнутые результаты (PFUMILI 4)

Запустив тот же тест на исполнение коллективом из NP=2 процессов, мы получили для него следующую статистику:

```
Proc: 0 of  2. Time  1 =    0.02 sec.  Ncall=    45  callF=  1690166
Proc: 0 of  2. Time  3 =    0.33 sec.  Ncall=     3  callF=  1690166
Proc: 0 of  2. Time  4 =   13.16 sec. Ncall=    45  callF=  1690166
Astime:     13.70  CPU_time:    13.55  for process  0

Proc: 1 of  2. Time  1 =    0.02 sec.  Ncall=    45  callF=  1690001
Proc: 1 of  2. Time  3 =    0.33 sec.  Ncall=     3  callF=  1690001
Proc: 1 of  2. Time  4 =   13.12 sec. Ncall=    45  callF=  1690001
Astime:     13.59  CPU_time:    13.50  for process  1
```

Здесь астрономического времени AsTime затрачено в 1.7 раз меньше, чем при NP=1, поскольку 2 процесса работали одновременно, а ускорение "не дотянуло" до идеальных 2 именно из-за необходимости совершать межпроцессные обмены данными.

Параллельная версия программы **MINUIT** – минимизации функций многих переменных (1)

MINUIT – программа минимизации функции многих переменных, написанная в начале 70-х годов прошлого века Фредериком Джеймсом (ЦЕРН) и весьма популярная до сих пор. MINUIT состоит более чем из 60 подпрограмм, общая длина которых более 7000 строк.

Основной этап работы по распараллеливанию – разделение вычислительных операций между отдельными процессами, а также еще ряд достаточно типичных для любой большой вычислительной программы этапов:

- Организация обрамления программы: подпрограммы MN_Start и MN_Finish для захвата и освобождения MPI-ресурсов.
- Проработка набора базовых коммуникационных операций, специфичных именно для MINUIT.
- Подготовка программ-заглушек пакета MPI - для обеспечения возможности эксплуатации этой версии MINUIT без MPI.

Параллельная версия программы **MINUIT** – минимизации функций многих переменных (2)

- Реорганизация операций ввода и вывода:

  Печать, и вообще весь вывод, естественно, должен выполнять только один из P процессов, участвующих в совместной работе, и, очевидно, это должен быть процесс 0. Поэтому все операторы вывода должны предваряться проверкой номера процесса:

  If (MyProc.eq.0) Write(,*) Data*

  Что же касается ввода, то тут чуть сложнее. Ясно, что файлы с входной информацией должны принадлежать кому-то одному, стало быть нулевому процессу. Прочитав из файла, он должен поделиться прочитанным со своими партнерами:

  If (MyProc.eq.0) read(1,) (data(i),i=1,count)*
  call MN_Bcast(data,count) !Propagate for all processes

  Это типичная техника организации ввода начальных данных при распараллеливании программы.

CLEBSCH2 - программа для вычисления простейшей формы коэффициентов Клебша-Гордана

*Real*8 function Clebsch2(k,n)* вычисляет коэффициент Клебша-Гордана

$$C(k,n)=k!*(n-k)!/n!$$

Программа свободна от типичных при вычислении факториалов "в лоб" случаев переполнения при умножении.

В программе реализован вариант – разумно чередовать операции умножения и деления. Присмотревшись к выражению коэффициента, можно увидеть, что он получается равным произведению первых K членов ряда 1,2,3…… N, деленному на произведение K последних членов этого ряда.
Поэтому цикл работы программы можно сократить с N до K и учесть симметрию C(k,n)=C(n-k,n). В результате имеем простейший цикл:

```
c=1.
do i=1, Min(K,N-K)
      c=c*float(i)/float(N-i+1)
enddo
Clebsch=c
```

Программа делает всего не более N/2 умножений и N/2 делений, является коллективной операцией пакета MPI и прямым потомком операции MPI_AllReduce.

PRIMUS – параллельная программа, реализующая классический алгоритм решета Эратосфена для генерации простых чисел

В работе /*/ опубликована программа, реализующая классический генератор простых чисел, то есть целых чисел, нацело делящихся только на 1 и самих себя. Как видно уже из ее названия, программа реализует классический алгоритм Эратосфена, в просторечии именуемый эратосфеновым решетом.

В параллельной программе авторский интерфейс был модифицирован: вместо одного параметра N введен диапазон [N0,N], что позволило легко использовать ее при параллельной работе нескольких процессов в рамках единой задачи.

Для распараллеливания была написана программа Primus, которая по существу является надстройкой над Eratosthenes, позволяющая задействовать заданное количество NP процессов. Распараллеливание в ней свелось к простому делению отрезка [2,N] числовой оси между процессами приблизительно поровну с последующим объединением полученных результатов в памяти главного процесса.

/*/ Lumomir Alexandrov, D. B. Baranov, Plamen Yotov (JINR, BLTP, Dubna, Russia). Polynomial splines interpolating prime series // JINR-P5-2002-228. http://arxiv.org/abs/math/0212246v5

PROFILE - программный инструмент для исследования производительности программ (1)

PROFILE - программный инструмент для исследования производительности программ в определяемых пользователем интервалах. Программа пригодна для использования в традиционных (последовательных) фортранных программах, так и в распараллеленных с использованием технологии MPI.

Измерение времени работы процессора при исполнении исследуемой программы является чрезвычайно полезным мероприятием.
Оно позволяет:

- оценить производительность программы в целом, определяя ее конкурентоспособность в ряду аналогичных программ;
- выявить наиболее времяёмкие места изучаемой программы, чтобы в дальнейшем сосредоточить свои усилия именно на них.

PROFILE - программный инструмент для исследования производительности программ (2)

Предполагается, что пользователь хочет измерить время работы своей программы на N<200 интервалах ее исполнения. Тогда при N=-1 единожды совершается первоначальная засечка времени с помощью стандартной системной программы Seconds. Далее, **в начале и конце** каждого N-го интервала работы исследуемой программы необходимо вызвать Profile(N). Эти засечки делаются с использованием стандартной программы CPU_Time. Таким образом, общее количество временных засечек должно быть четным.

В конце программы следует вызвать Profile(0) для вывода всех сделанных засечек. Результат выглядит примерно так:

```
Time  0 =     5.4 sec.
Time  1 =     6.1 sec.
 . . .
Time  <N-1> = 4.8 sec.

WallTime:      6.5  Total CPU_time:      6.1 sec.
```

IntroMPI - учебные материалы по технологии MPI

Предлагается подборка учебных материалов по технологии параллельного программирования MPI:

- ❑ архив с учебными программами по технологии MPI;
- ❑ статья "Как нам распараллелить программу и запустить ее на кластере HybriLIT", где излагаются основы технологии параллельного программирования MPI, даются некоторые рекомендации для работы на кластере ЛИТ.
- ❑ презентация "Введение в MPI".

Благодарю за внимание!