Вычисления с заданной точностью в среде mpi

В.В. Горбик, А.В. Панюков

Южно-Уральский государственный университет, Челябинск

Введение

Существенным недостатком вычислений с помощью аппаратных типов данных с плавающей точкой является погрешность. Данную проблему позволяет решить использование производных дробно-рациональных типов данных, в которых числа представляются точно в виде дроби из векторов переменной длины [1]. Однако операции над такими типами данных на порядки увеличивают вычислительную нагрузку и, следовательно, оттесняет круг требовательных к ресурсам задач в ранг невыполнимых (в отношении времени вычислений и требований к памяти для хранения представлений разрастающихся чисел). В этом случае, когда задачи требуют таких объемов расчетов, при которых не предоставляется возможным проводить вычисления точно, и погрешности решения задач стандартными аппаратными типами данных выходят за пределы допустимого, можно использовать типы данных с плавающей точкой увеличенной точности. Эффективные реализации таких производных типов данных, в отличие от дробно-рациональных, сравнимы по скорости вычислений с аппаратным (при аналогичной длине мантиссы), при этом позволяют произвольно динамически увеличить точность.

Вычисления с заданной точностью

В данной работе используются типы данных *mpf* (*multiple precision floating-point*, из открытой библиотеки *gnu mp*) и *mpfi* (из одноименной библиотеки *multiple precision floating-point interval library* [2]). Выбор был сделан на основе того, что:

- код библиотеки *gnu mp* оптимизирован под большое количество архитектур и имеет хорошую производительность;
- обе библиотеки входят в поставку любого *linux* дистрибутива и распространяются по свободной лицензии *lgpl*.

Библиотека *mpfi* основана на *gnu mp* и *mpfr* (*multiple-precision floating-point with* correct rounding *[3]*). Вместо единого числа в *mpfi* используется пара чисел с плавающей точкой заданной длины (имющих тип *mpfr*), представляющих интервал, внутри которого лежит реальное значение.

В отличие от *mpf*, *mpfi* позволяет получить *гарантированные* (за счет использования интервальных вычислений) и точно определенные результаты (за счет использования *правильных* округлений по стандарту *IEEE 754*).

Адаптация типов данных с заданной точностью к mpi

Сложность эффективной адаптации производных типов динамической длины к mpi заключается в том, что в mpi нет стандартных средств для передачи типов:

- располагающихся в памяти непоследовательно,
- изменяющих длину блоков данных в процессе выполнения программы.

На рис. 1 представлены структуры типов mpf_t и mpfi_t (для 64-битных архитектурах). Мантиссы хранятся вне базовой структуры. В *mpf* на мантиссу указывает *_mp_d*. В случае с *mpfi* указатели *_mpfr_d* указывают на вторые элементы выделенных блоков памяти (начало мантиссы), в первых элементах хранится текущая длина мантиссы. Таким образом, данные в памяти лежат вразброс, и на основе стандартного представления mpf и mpfi невозможно определить *mpi* тип данных. Однако, эффективную передачу mpf и mpfi можно осуществить двумя способами:

- неполная сериализация,
- упорядочевание распределения в памяти.

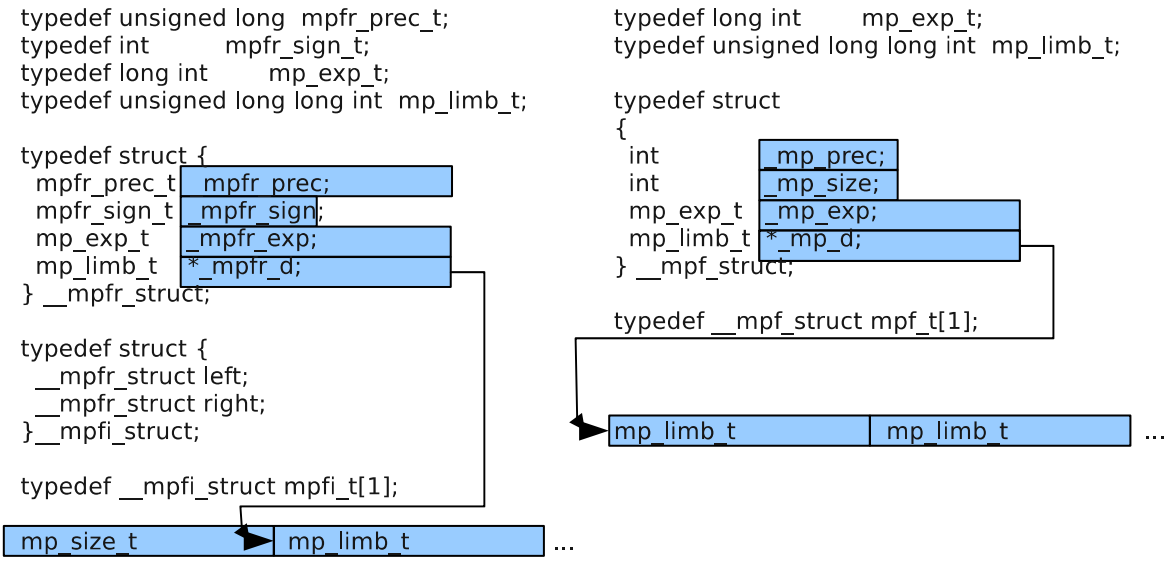

Рисунок 1. Типы данных *mpf* и *mpfi*.

В случае неполной сериализации достаточно последовательно упаковывать необходимые поля, закрашенные на рис. 2.

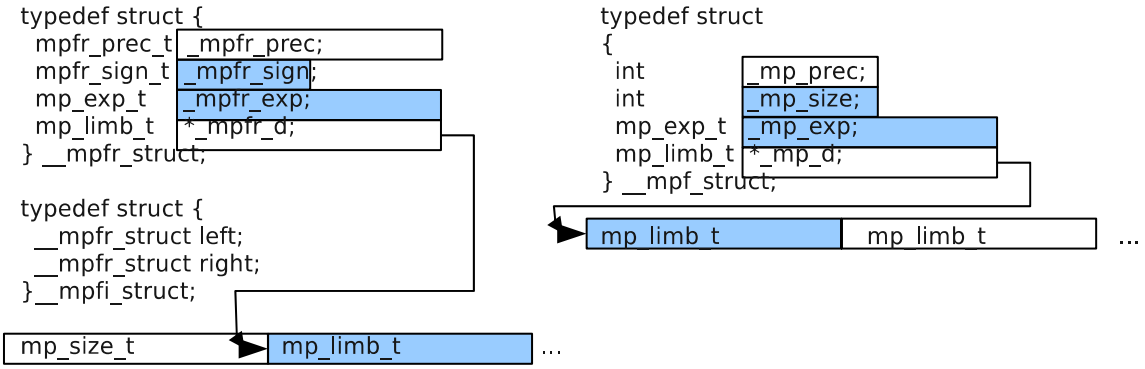

Рисунок 2. Сериализация типов *mpf* и *mpfi*.

В зависимости от алгоритма, если на стороне приемника не известна заданная точность отправителя, в список очевидно следует включить поля *_mp_prec* (*_mpfr_prec*). В некоторых случаях, можно передавать только *_mp_size* задействованных элементов массива мантиссы (в общем случае *_mp_size* совпадает с полной длиной *_mp_prec*).

Данный подход имеет очевидный минус, заключающийся в том, что приходится реализовывать функции передачи *mpi* на основе передачи массива байт (в случае неполной передачи мантиссы — массива байт заранее неизвестной длины). Некой прозрачности можно добиться, переопределив стандартные функции теми, которые будут проверять mpi тип (для *mpf* и *mpfi* можно взять не занятые) и выполнять передачу mpf (mpfi) или же вызывать стандартную функцию.

Упорядочевание распределения в памяти

Данный подход имеет смысл в случае, когда алгоритм оперирует числами заданной точности, которая не изменяется в процессе вычислений. Используя макросы приведенные на рис. 3 можно определить производные от *mpf* и *mpfi* типы *mpfn* и *mpfin*.

```
#define mpfi_type(prec) \
typedef struct \
{ \
__mpfr_struct left; \
__mpfr_struct right; \
mp_limb_t _mp_d_left[L(prec) + 1]; \
mp_limb_t _mp_d_right[L(prec) + 1]; \
} __mpfi_struct##prec; \
typedef __mpfi_struct##prec \ mpfi##prec##_t[1]; \
MPI_Datatype MPI_mpfi##prec;

#define mpf_type(prec) \
```

Рисунок 3. Макросы определения производных типов *mpfi* и *mpf*.

Очевидно, что структуры типов *mpfn* и *mpfin*, а также массивы объектов данных типов будут располагаться в памяти последовательно (не учитывая выравнивание). При этом, все операции, (кроме инициализации и задания точности) доступные для *mpf* (*mpfi*), могут быть без ограничений применены к *mpfn* (*mpfin*). Процедуры инициализации не принципиально отличается от оригинальных. Вместо выделения памяти под мантиссу требуется просто инициализировать указатели *_mp_d* (*left._mpfr_d* и *right._mpfr_d*) соответствующими адресами статически выделенной мантиссы.

Значительный положительный фактор данного подхода заключается в том, что на основе типов *mpfn* и *mpfin* (для каждой заданной точности) можно легко объявить тип данных *mpi*, и далее использовать все *mpi* функции доступные для *mpi*-производных типов данных без ограничений.

На рис. 4 представлены структуры *mpfn* и *mpfin* с точки зрения *mpi* (для 64-битных архитектур).

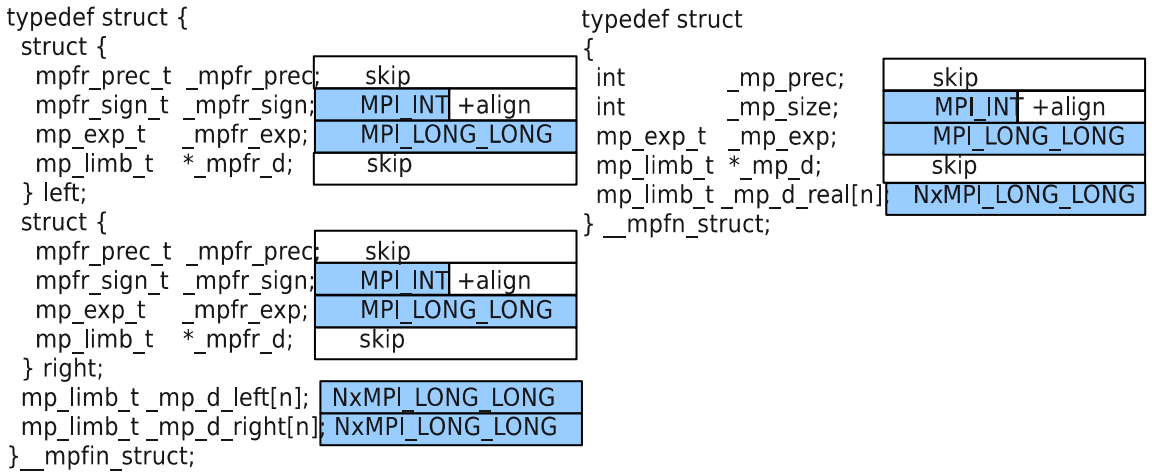

Рисунок 4. Поля включаемые в описание *mpi*-типа.

Если объявить *mpi* тип данных таким образом, как показано на рис. 4, указывая пропускать неокрашенные области, то указатели на мантиссы на стороне приемника переписываться не будут.

Еще один важный вопрос по данному подходу — как упорядочевание влияет на производительность. В таблице 1 приведено сравнение попаданий в кэш исходных и модифицированных типов. Сравнение производилось на процессоре с кэш второго уровня 2 Мб (32 Кб первого уровня) на задаче решения системы уравнений методом

Жордана-Гаусса малой размерности в 30 уравнений с точностью 192 бит. Временные характеристики приведены в разделе вычислительный эксперимент.

Таблица. 1. тесты на кэш-попадания исходных и модифицированных типов

	mpfi	mpfin	mpf	mpfn
I refs:	43366674	41850711	13762859	13601951
I1 misses:	23822	21566	10938	10205
L2i misses:	3456	3447	3236	3228
D refs:	18095665	17450618	5076607	5012224
D1 misses:	84427	65950	46662	38923
L2d misses:	13121	12126	9886	9646
L2 refs:	108249	87516	57600	49128
L2 misses:	16577	15573	13122	12874

На первый взгляд, из обобщенных тестов следует, что показатели кэш-попаданий в модифицированных типах лучше. Однако, при более детальном рассмотрении по конкретным функциям, оказывается, что кэш-попадания лучше для простых функций (сравнение, сложение, вычитание и т.д.), но несколько хуже для действий умножения и деления. С ростом размерности задачи до 3000 уравнений и увеличением точности до 1024-2048 бит наблюдается незначительное превосходство исходных типов.

Вычислительный эксперимент

Вычислительный эксперимент производился на кластере "Скиф-Урал" Южно-Уральского государственного университета. Краткие технические характеристики представлены в таблице 2.

Таблица 2. Технические характеристики вычислительной платформы

Тип процессора (на 1 blade)	2 четырехъядерных Intel Xeon E5472 3.0 GHz
Оперативная память (на 1 blade)	8 GB
Тип системной сети	InfiniBand (20Gbit/s, макс. задержка 2 мкс)
Операционная система	SUSE Linux Enterprise Server 10 x86_64

Для вычисленного эксперимента использовалась параллельная версия алгоритма Жордана-Гаусса, адаптированная для вычислений с типами *mpf* (*mpfn*) и *mpfi* (*mpfin*).

Эффективность распараллеливания представлена на на рис. 5. Стоит отметить, что с увеличением точности (длины мантиссы) эффективность распараллеливания возрастает.

Заключение

В данной статье были предложены методы для эффективного проведения расчетов с плавающей точкой заданной точности (с использованием библиотеки *gnu mp* и *mpfi*) в среде *mpi*. При решении подобных задач положительный эффект от распараллеливания не только в ускорении вычислений, но и возможности решать задачи большей размерности, т.к. достаточно легко достичь границ, когда матрица целиком не будет умещаться в оперативной памяти одного узла.

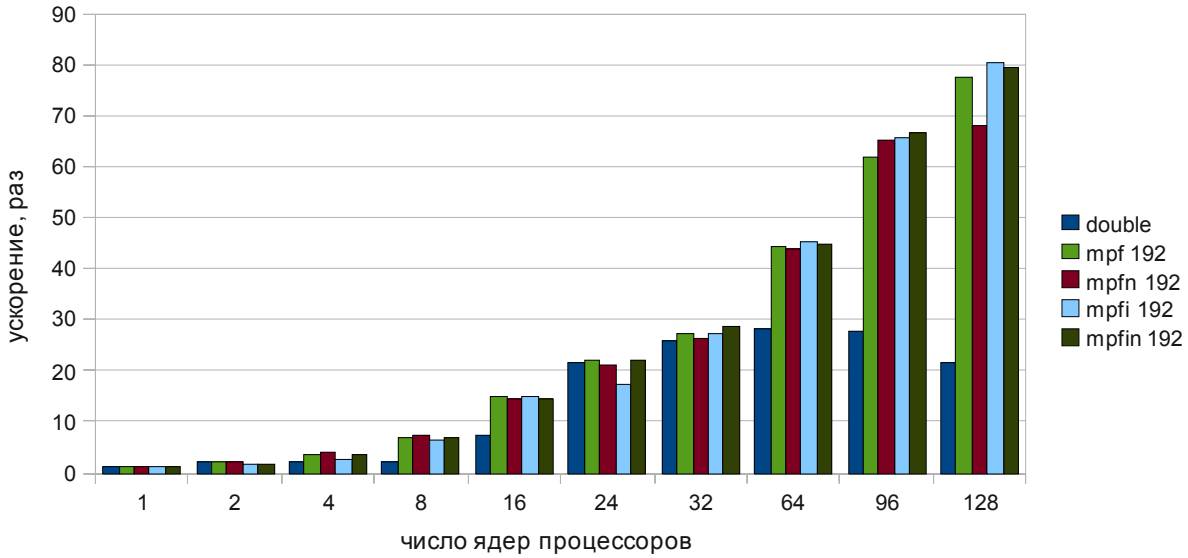

Рисунок 5. Эффективность распараллеливания.

Таблица 3. Результаты численного эксперимента

	double	mpf			mpfn			mpfi	mpfin
	53	64	192	704	64	192	704	192	192
1	43.04	1250.99	1817.42	6203.19	1238.5	1796.03	6219.7	3943.99	3817.11
2	22.06	627.28	913.78	3106.4	617.31	910.5	3132.24	2747.88	2073
4	20.41	315.35	489.06	1561.31	312.38	463.75	1573.92	1464.61	1072.29
8	18.83	171.44	269.83	795.95	166.03	239.94	804.95	634.02	537.36
16	5.98	86.91	123.35	413.09	85.19	124.01	410.43	265.07	261.93
24	1.98	60.12	83.07	325.72	61.5	84.44	277.53	225.7	172.57
32	1.68	47.7	66.41	212.97	47.3	68.72	220.01	144.66	133.1
64	1.53	27.9	40.98	135.55	28.06	40.97	123.79	87.36	84.79
96	1.54	19.85	29.25	92.61	19.88	27.57	84.55	60.14	57.31
128	2.01	16.86	23.41	77.1	18.53	26.31	76.18	49.15	47.9

Литература

1. Горбик В.В. Безошибочное решение задач линейного программирования на многопроцессорных системах // Параллельные вычислительные технологии (ПаВТ'2008), 2008. С. 364-369
2. *mpfi library*. URL: http://perso.ens-lyon.fr/nathalie.revol/software.html
3. *mpfr library*. URL: http://www.mpfr.org/