Федеральное государственное бюджетное учреждение науки
Институт вычислительной математики и
математической геофизики
Сибирского отделения Российской академии наук

Центр коллективного пользования
«Сибирский суперкомпьютерный центр СО РАН»

Документация для библиотеки PARMONC

М.А. Марченко

email: mam@osmf.sscc.ru

2016

Содержание

1 Методика распределенного численного статистического моделирования 3

2 Распределительный способ получения псевдослучайных чисел 5

3 Библиотека PARMONC 11

3.1 Процедуры для массивно-параллельной вычислительной системы для случая независимого моделирования отдельных реализаций 15

3.2 Процедуры для массивно-параллельной вычислительной системы для случая моделирования отдельных реализации совместно на нескольких процессорах 19

3.3 Процедуры для гибридной вычислительной системы для случая независимого моделирования отдельных реализаций 21

3.4 Процедуры для гибридной вычислительной системы для случая моделирования отдельных реализации совместно на нескольких процессорах . . 21

4 Примеры использования библиотеки PARMONC 22

4.1 Описание задачи 22

4.2 Пример программы на FORTRAN-е 23

4.3 Пример программы на C 26

4.4 Компиляция программы 28

4.5 Запуск задания на кластере 29

4.6 Анализ результатов 29

4.7 Возобновление расчетов после их остановки 30

5 Приложения 33

1 Методика распределенного численного статистического моделирования

1. Под численным статистическим моделированием понимается компьютерная реализация вероятностной модели некоторого объекта с целью оценивания необходимых интегральных характеристик (функционала φ) на основе закона больших чисел [1]:

$$\varphi \approx \mathrm{E}\zeta \approx \bar{\zeta} = \frac{1}{L}\sum_{i=1}^{L}\zeta^{(i)}.$$

Здесь каждая реализация $\zeta^{(i)}$ функционально зависит от некоторого числа независимых случайных величин α, равномерно распределенных в интервале $(0,1)$:

$$\zeta^{(i)} = \zeta^{(i)}(\alpha_1, \alpha_2, \ldots, \alpha_k),\ k = k(i)$$

и ее получение осуществляется при помощи определенного численного алгоритма. Компьютерное моделирование выборочных значений случайной величины α осуществляется с помощью специального алгоритма – **генератора псевдослучайных чисел** [1].

2. Абсолютная **статистическая погрешность** оценки функционала (т.е. погрешность оценки математического ожидания выборочным средним) выражается величиной

$$\varepsilon_{stat}^{(a)} = \gamma\sqrt{\mathrm{D}\zeta/L},$$

где величина γ зависит от требуемого уровня доверия [1]. При приближенной оценке функционалов целесообразно потребовать, чтобы детерминированная погрешность ε_{det} (т.е. погрешность оценки функционала математическим ожиданием) была примерно равна по модулю абсолютной статистической погрешности: $|\varepsilon_{det}| \approx |\varepsilon_{stat}^{(a)}|$.

3. Поскольку среднее машинное время для моделирования L реализаций случайной величины ζ равно $t_1 L$, где t_1 – среднее машинное время для моделирования одной реализации, то из условия равенства детерминированной и абсолютной статистической погрешностей следует, что

$$L \approx C\varepsilon_{det}^{-2}\mathrm{D}\zeta,\ C > 0.$$

Если оцениваемый функционал является малым по модулю

$$|\varphi| \approx |\mathrm{E}\zeta| \approx 0,$$

то целесообразно поставить условие равенства детерминированной и относительной статистической погрешностей:

$$\varepsilon_{det} \approx \varepsilon_{stat}^{(r)} = \gamma\frac{\sqrt{\mathrm{D}\zeta/L}}{|\mathrm{E}\zeta|}.$$

В этом случае

$$L \approx C\varepsilon_{det}^{-2}(\mathrm{E}\zeta)^{-2}\mathrm{D}\zeta,\ C > 0.$$

Из приведенных соотношений следует, что вычислительная трудоемкость алгоритма численного статистического моделирования пропорциональна величине

$$C(\zeta_i) = t_1 \varepsilon_{det}^{-2} \mathrm{D}\zeta,$$

а для случая малых по модулю функционалов – величине

$$C(\zeta_i) = t_1 \varepsilon_{det}^{-2} (\mathrm{E}\zeta)^{-2} \mathrm{D}\zeta.$$

Ключевой проблемой при использовании методов численного статистического моделирования является достаточно большая величина их вычислительной трудоемкости. Понизить трудоемкость алгоритмов статистического моделирования можно либо путем использования их модификаций с меньшей дисперсией (с тем же значением математического ожидания либо с заданным смещением), либо путем уменьшения затрат машинного времени на получение одного выборочного значения случайной оценки, либо сочетанием этих подходов.

4. Вычислительно трудоемкие задачи статистического моделирования характеризуются следующими особенностями (одной или сочетанием нескольких):

- значительной величиной времени моделирования τ одной реализации случайной величины ζ,
- большой величиной дисперсии $\mathrm{D}\zeta$,
- требуемой малой величиной верхней границы погрешности ε,
- большими потребностями в машинной памяти (когда при моделировании используется много оперативной памяти или оценка математического ожидания требует сохранения на диск значительных по объему массивов данных).
- многократными вычислительными экспериментами (параметрический анализ, оценка нескольких функционалов, подбор параметров модели и т.п.).

Для эффективного осреднения случайных оценок в такого рода задачах необходимо применять распределенные вычисления на многопроцессорных компьютерах.

5. Методика распределенного численного статистического моделирования состоит в организации моделирования независимых реализаций и вычисления выборочных средних на отдельных процессорах (процессорах-вычислителях), а также периодической передачи полученных выборочных средних на некоторый выделенный процессор – **центральный процессор-сборщик**, на котором их получают и осредняют по формуле

$$\bar{\zeta}_M = (\sum_{m=0}^{M-1} l_m)^{-1} \sum_{m=0}^{M-1} l_m \bar{\zeta}^{(m)}, \qquad (1)$$

где l_m – объем выборки, соответствующий m-у процессору, $\bar{\zeta}^{(m)}$ – соответствующее m-у процессору выборочное среднее. При вычислениях применяются следующие соображения:

- для каждого процессора псевдослучайные числа выбираются из непересекающихся подпоследовательностей базовой последовательности $\{\alpha_k\}$,
- вычисления можно проводить на разных процессорах в разное время, объемы выборки, полученные на процессорах, могут различаться; при этом допустимо использовать вычислительные процессоры с разной производительностью;
- результаты ранее проведенных вычислений (при условии, что распределение случайной величины не изменяется) можно использовать при осреднении с новыми результатами по формуле (1).

Понятно, что главными критериями осуществимости такой методики является:

- использование эффективно параллелизуемых генераторов псевдослучайных чисел,
- возможность «поместить» данные вычислительной программы для моделирования реализаций в оперативную память каждого процессора,

2 Распределительный способ получения псевдослучайных чисел

1. Для получения базовой последовательности псевдослучайных чисел $\{\alpha_n\}$ предлагается использовать следующий 128-битный конгруэнтный генератор псевдослучайных чисел (метод вычетов) [2, 3]:

$$u_0 = 1,\ u_n \equiv u_{n-1}A \pmod{2^{128}}, \qquad \alpha_n = u_n 2^{-128}, \qquad n = 1, 2, \ldots, \tag{2}$$

где множитель A определяется выражением:

$$A \equiv 5^{100109} \pmod{2^{128}}.$$

В работе [2] было произведено проверка генератора с использованием ряда статистических тестов: для начальных 10^9 чисел были успешно реализованы стандартные статистические тесты. Были также проведены аналитические исследования, связанные с тем, что n-мерные распределения для метода вычетов, вообще говоря, сосредоточены на семействах плоскостей, то есть на многообразиях меньших размерностей [4]. Однако для рассматриваемого генератора, как показывают результаты, полученные в [2], эти многообразия достаточно плотно заполняют соответствующие n-мерные единичные гиперкубы и этим дефектом можно пренебречь.

В [5] приведены результаты тестов n-мерной равномерности для выборок, которые были получены объединением начальных чисел из заданных подпоследовательностей генератора (2): многомерные распределения проверялись на равномерность в n-мерном кубе ($n = 1, 2, \ldots, 8$) по критерию χ^2, объем выборки псевдослучайных чисел равен 10^{10}. Путем решения тестовых задач проверялась также статистическая эквивалентность

генератора (2) и конгруэнтного генератора с параметрами $r = 40, A = 5^{17}$: были успешно проверены гипотезы о равенстве математических ожиданий для случайных оценок ряда функционалов (в том числе таких, для которых известно точное значение), получаемых независимо с использованием указанных генераторов.

2. При реализации методики распределенного статистического моделирования целесообразно определять различные подпоследовательности базовых псевдослучайных чисел для вычислительных экспериментов (конкретных расчетов для выбранной задачи), процессоров, реализаций (отдельных случайных испытаний) и их элементов. Такой способ можно назвать **распределительным**, так как он особенно удобен для распределенных вычислений.

Базовая последовательность $\{\alpha_n\}$ предварительно разделяется на подпоследовательности заданной длины μ, начинающиеся с чисел $\alpha_{k\mu}$, $k = 0, 1, 2, \ldots$. Значение μ должно выбираться из тех соображений, что μ псевдослучайных чисел практически достаточно для реализации соответствующего укрупненного блока. Для метода вычетов начальное значение k-й подпоследовательностей получается по формуле

$$u_{(k+1)\mu} \equiv u_{k\mu} A^{\mu} (\mathrm{mod}\ 2^{128}),\ \alpha_{k\mu} = u_{k\mu} 2^{-128}, k = 0, 1, 2, \ldots. \tag{3}$$

Такая методика распределения псевдослучайных чисел позволяет определять одинаковые подпоследовательности целиком для каждого вычислительного эксперимента, процессора, реализации и ее элементов. Ее использование улучшает корреляционную зависимость между результатами расчетов для разных вариантов задач, что особенно полезно при параметрическом анализе вероятностных моделей.

3. В дальнейшем под **массивно-параллельной вычислительной системой** будем понимать однородную (из одинаковых процессоров или вычислительных ядер) систему с распределенной памятью, где у каждого процессора (вычислительного ядра) – своя оперативная память [6]. Процессоры при этом обычно объединены в вычислительные узлы, причем в самой вычислительной системе вычислительные узлы могут иметь различную производительность. Под **гибридной вычислительной системой** будем понимать систему, состоящую из вычислительных узлов, на каждом из которых расположены основные процессоры (CPU) и многоядерные сопроцессоры (например, Intel Xeon Phi [7]).

При распределении псевдослучайных чисел по подпоследовательностям необходимо учитывать особенности параллельного алгоритма и тип используемой вычислительной системы. А именно, выделим следующие варианты применения распределенного способа:

1. массивно-параллельные вычислительные системы для случая, когда моделирование отдельных реализаций осуществляется независимо на разных процессорах (вычислительных ядрах);

2. массивно-параллельные вычислительные системы для случая, когда моделирование каждой реализации осуществляется совместно на нескольких процессорах;

3. гибридные вычислительные системы для случая, когда моделирование отдельных реализаций осуществляется независимо как на вычислительных ядрах основного процессора, так и на вычислительных ядрах сопроцессора;

4. гибридные вычислительные системы для случая, когда моделирование каждой реализаций осуществляется совместно на вычислительных ядрах основного процессора и на вычислительных ядрах сопроцессора.

В первом и третьем варианте предполагается, что оперативной памяти, доступной отдельному процессору или вычислительному ядру, достаточно для осуществления моделирования отдельной реализации. Во втором и четвертом вариантах для моделирования одной реализации используется вся оперативная память, доступная вычислительным ядрам узла, причем в процессе моделирования происходит обмен данными между ядрами (фактически, между MPI- или OpenMP-процессами, запущенными на ядрах).

Для разных вариантов применения распределительного способа целесообразно использовать иерархическую структуру (т.е. вложенность) подпоследовательностей базовой последовательности генератора, начиная с самой длинной – подпоследовательностей для вычислительных экспериментов (обозначим их длину как n_e).

4. На массивно-параллельных вычислительных системах для случая, когда моделирование отдельных реализаций осуществляется независимо на разных процессорах (вычислительных ядрах), применяется следующая процедура:

- подпоследовательности для вычислительных экспериментов «разрезаются» на подпоследовательности для процессоров (обозначим их длину как n_p);

- подпоследовательности для процессоров «разрезаются» на подпоследовательности для реализаций (обозначим их длину как n_r);

- подпоследовательности для реализаций «разрезаются» на подпоследовательности для элементов реализации (обозначим их длину как n_l).

Ясно, что длина подпоследовательностей для элементов реализации должна быть достаточно большой, чтобы содержащиеся в них псевдослучайные числа в совокупности удовлетворяли необходимым статистическим тестам.

Описанный способ распределения псевдослучайных чисел соответствует особенностям алгоритма и вычислительной системы: сначала выделяются подпоследовательности для вычислительных экспериментов, затем – для процессоров, на которых будут независимо моделироваться реализации. Такой способ назовем **bf-генератором** (bf – сокращение от «Big-Frog») (см. рис. 1).

5. На массивно-параллельных вычислительных системах для случая, когда моделирование каждой реализации осуществляется совместно на нескольких процессорах, применяется следующая процедура:

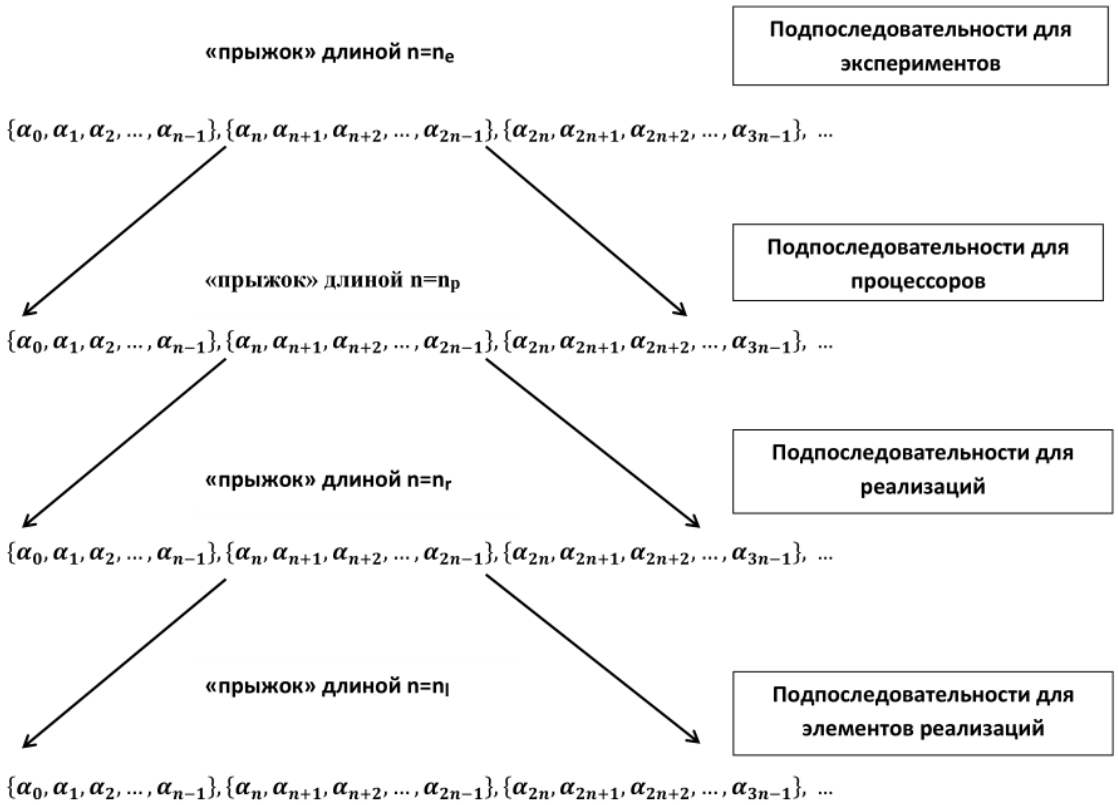


Рис. 1: Вариант распределительного способа для массивно-параллельных вычислительных систем для случая, когда моделирование отдельных реализаций осуществляется независимо на разных процессорах (вычислительных ядрах); во вложенных подпоследовательностях псевдослучайные числа перенумеровываются по новому.

- подпоследовательности для вычислительных экспериментов «разрезаются» на подпоследовательности для вычислительных узлов (обозначим их длину как n_n);
- подпоследовательности для вычислительных узлов «разрезаются» на подпоследовательности для реализаций (обозначим их длину как n_r);
- подпоследовательности для реализаций «разрезаются» на подпоследовательности для вычислительных ядер (обозначим их длину как n_c);
- подпоследовательности для вычислительных ядер «разрезаются» на подпоследовательности для элементов реализации (обозначим их длину как n_l).

Описанный способ распределения псевдослучайных чисел соответствует особенностям алгоритма и вычислительной системы: сначала выделяются подпоследовательности для вычислительных экспериментов, затем – для вычислительных узлов, на которых будут моделироваться реализации совместно на нескольких вычислительных ядрах. Назовем такой способ **bfm-генератором** («bfm» – сокращение от «Big-Frog for Many processors ») (см. рис. 2).

6. На гибридных вычислительных системах для случая, когда моделирование отдельных реализаций осуществляется независимо как на вычис-

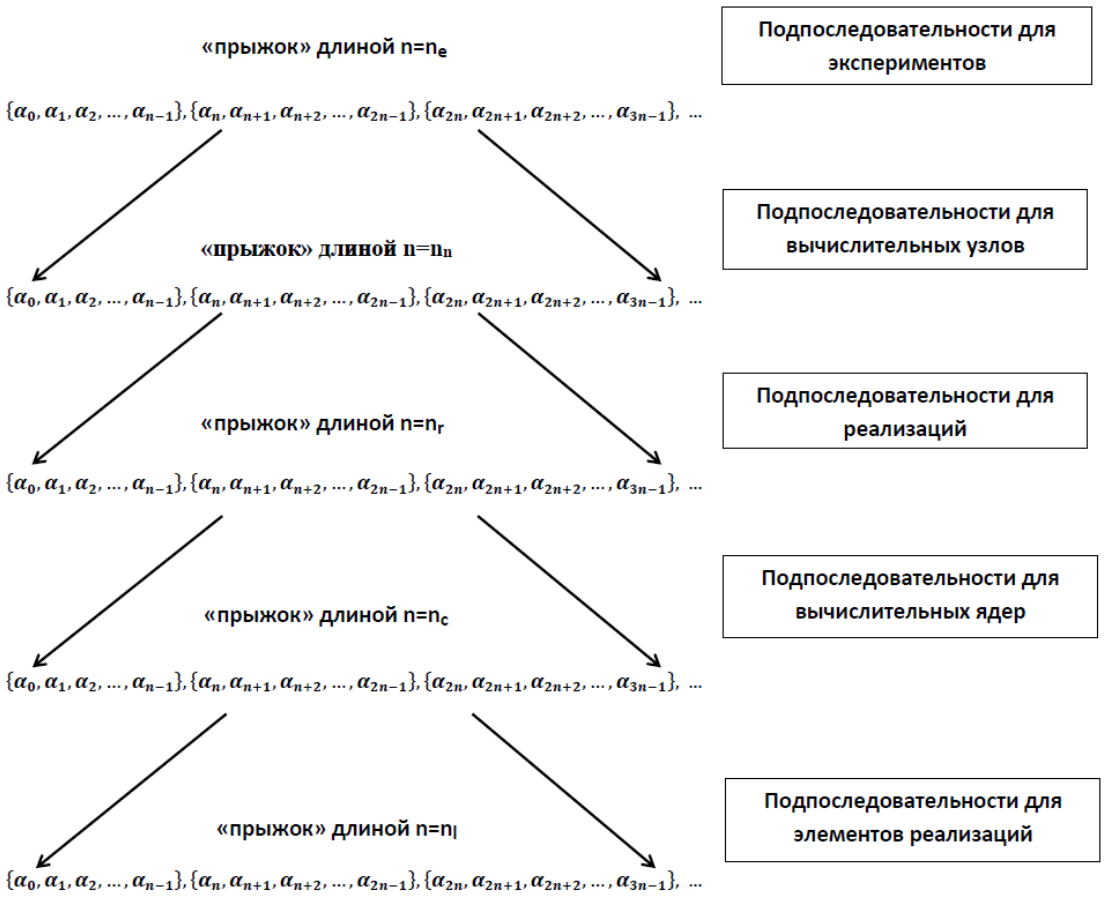


Рис. 2: Вариант распределительного способа для массивно-параллельных вычислительных систем для случая, когда моделирование каждой реализации осуществляется совместно на нескольких процессорах; во вложенных подпоследовательностях псевдослучайные числа перенумеровываются по новому.

лительных ядрах основного процессора, так и на вычислительных ядрах сопроцессора, применяется следующая процедура:

- подпоследовательности для вычислительных экспериментов «разрезаются» на подпоследовательности для вычислительных узлов (обозначим их длину как n_n);
- подпоследовательности для вычислительных узлов «разрезаются» на подпоследовательности для групп вычислительных ядер (обозначим их длину как n_g);
- подпоследовательности для групп вычислительных ядер «разрезаются» на подпоследовательности для вычислительных ядер (обозначим их длину как n_c);
- подпоследовательности вычислительных ядер «разрезаются» на подпоследовательности для реализаций (обозначим их длину как n_r);
- подпоследовательности для реализаций «разрезаются» на подпоследовательности для элементов реализации (обозначим их длину как n_l).

Описанный способ распределения псевдослучайных чисел соответствует особенностям алгоритма и вычислительной системы: сначала выделяются подпоследовательности для

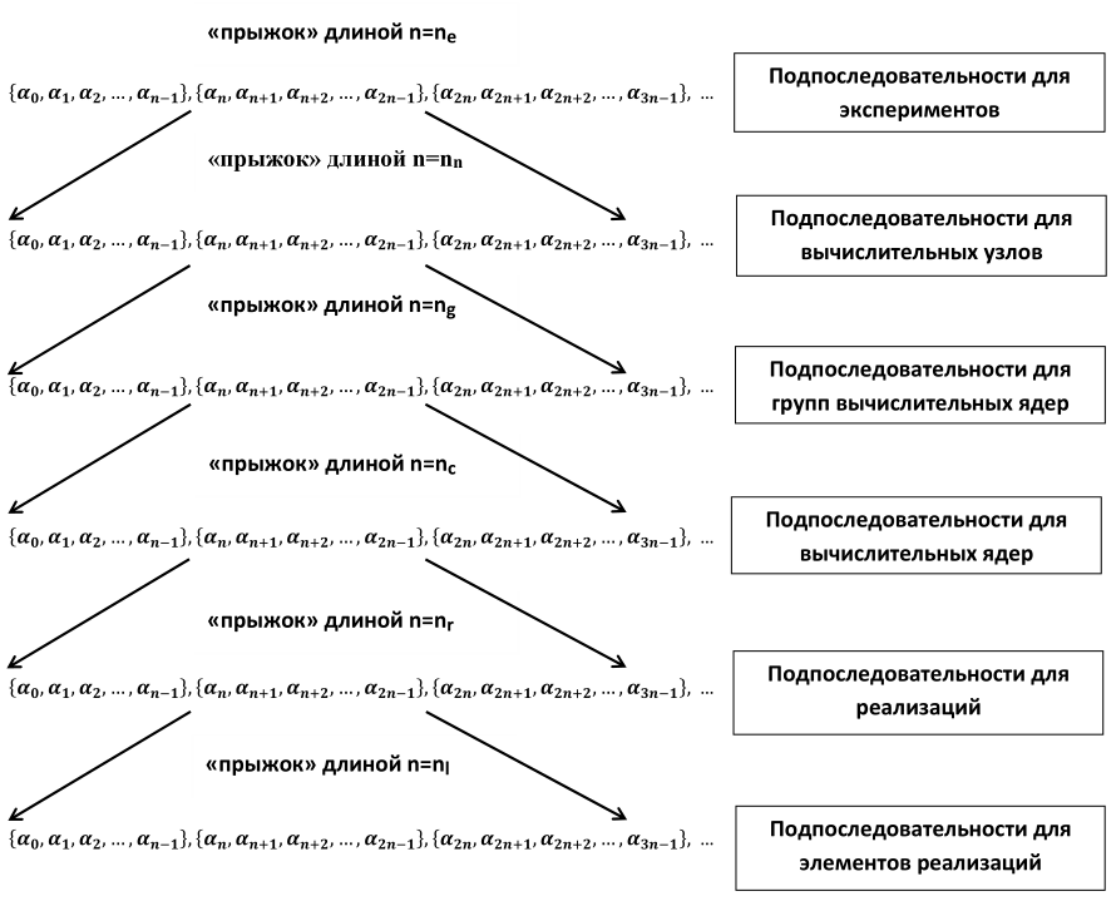


Рис. 3: Вариант распределительного способа для гибридных вычислительных систем для случая, когда моделирование отдельных реализаций осуществляется независимо как на вычислительных ядрах основного процессора, так и на вычислительных ядрах сопроцессора; во вложенных подпоследовательностях псевдослучайные числа перенумеровываются по новому.

вычислительных экспериментов, затем – для вычислительных узлов, на которых выделяются подпоследовательности для групп вычислительных ядер. Под такими группами понимаются группа ядер основного процессора (CPU) и группа ядер сопроцессора. Далее подпоследовательности для групп вычислительных ядер «разрезаются» таким же образом, как и для bf-генератора. Назовем такой способ **bfh-генератором** («bfh» – сокращение от «Big-Frog for Hybrid systems») (см. рис. 3).

7. На гибридных вычислительных системах для случая, когда моделирование каждой реализаций осуществляется совместно на вычислительных ядрах основного процессора и на вычислительных ядрах сопроцессора, применяется следующая процедура:

- подпоследовательности для вычислительных экспериментов «разрезаются» на подпоследовательности для вычислительных узлов (обозначим их длину как n_n);
- подпоследовательности для вычислительных узлов «разрезаются» на подпоследовательности для реализаций (обозначим их длину как n_r);

- подпоследовательности для реализаций «разрезаются» на подпоследовательности для групп вычислительных ядер (обозначим их длину как n_g);
- подпоследовательности для групп вычислительных ядер «разрезаются» на подпоследовательности для вычислительных ядер (обозначим их длину как n_c);
- подпоследовательности для вычислительных ядер «разрезаются» на подпоследовательности для элементов реализации (обозначим их длину как n_l).

Описанный способ распределения псевдослучайных чисел соответствует особенностям алгоритма и вычислительной системы: сначала выделяются подпоследовательности для вычислительных экспериментов, затем – для вычислительных узлов, на которых будут моделироваться реализации совместно на ядрах основного процессора и ядрах сопроцессора. Для моделирования реализаций далее выделяются подпоследовательности для групп вычислительных ядер. Под такими группами понимаются группа ядер основного процессора (CPU) и группа ядер сопроцессора. Далее применяется такая же процедура «разрезания», как для bfm-генератора. Такой способ распределения псевдослучайных чисел назовем **bfhm-генератором** (bfhm – сокращение от «Big-Frog for Hybrid systems and Many processors») (см. рис. 4).

Очевидно, что в отличие от обычного способа распределения случайных чисел «подряд» (то есть в порядке обращения к генератору), описанные выше процедуры обеспечивают меньшеее изменение результатов моделирования при изменении параметров задачи, по сравнении с процедурой выбора псевдослучайных чисел «подряд» из базовой последовательности.

8. В ряда случаев для моделирования элементов реализации полезно использовать также дополнительный быстродействующий генератор, например хорошо проверенный 40-битный генератор [1]:

$$v_0 = \tilde{v},\ v_n \equiv v_{n-1}B \pmod{2^{40}}, \qquad \beta_n = v_n 2^{-40}, \qquad n = 1, 2, \ldots, \tag{4}$$

где $B \equiv 5^{17} \pmod{2^{40}}$. Здесь для определения начального значения $\tilde{v}$ используются старшие 40 бит чисел u_n начальных чисел подпоследовательностей для реализаций генератора (2), т.е. генератор (4) «рандомизируется» с помощью генератора (2) с соответствующим увеличением длины периода. Подчеркнем, что в этом случае распределительный способ использования генератора (2) обеспечивает распределительный способ и для генератора (4) с соответствующим увеличением длины периода. Дополнительный генератор целесообразно использовать при моделировании элементов реализации, распределение которых различно в разных вариантах задачи.

Распределительный способ получения псевдослучайных чисел реализован в библиотеке `PARMONC` в процедурах `rnd128`, `rnd128el` и `rnd40`.

3 Библиотека PARMONC

1. В библиотеке PARMONC [8] реализована методика распределенного численного

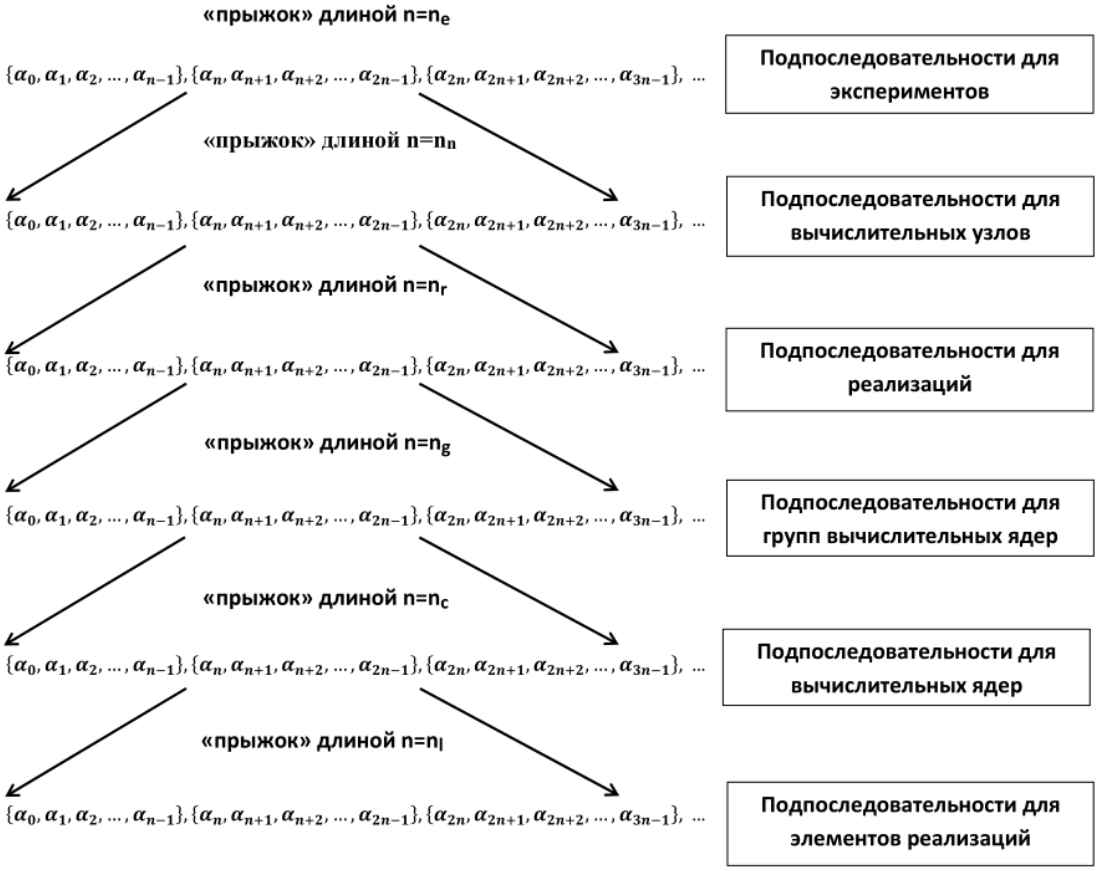


Рис. 4: Вариант распределительного способа для гибридных вычислительных систем для случая, когда моделирование каждой реализаций осуществляется совместно на вычислительных ядрах основного процессора и на вычислительных ядрах сопроцессора (т.е. на всем вычислительном узле); во вложенных подпоследовательностях псевдослучайные числа перенумеровываются по новому.

статистического моделирования. Для программы получено свидетельство о государственной регистрации [9].

2. Для всех вариантов сочетания архитектуры вычислительной системы и способа моделирования отдельных реализаций общими являются функции, реализующие параллельные генераторы псевдослучайных чисел:

- функция `rnd128` – параллельный 128-битный генератор (2); функция не имеет аргументов;
- функция `rnd40` – 40-битный генератор (4); функция не имеет аргументов;
- функция `rnd128el` – 128-битный генератор (2) с выбором номера подпоследовательности для элементов реализации; каждая такая подпоследовательность используется для моделирования конкретного элемента реализации; функция имеет один аргумент: `n` – номер подпоследовательности (0, 1, . . . , 999).

3. Для всех вариантов общей также является вспомогательная программа (для вы-

зова из командной строки) `manaver`, применяемая для осреднения результатов нескольких отдельных расчетов (т.е. сделанных в разных вычислительных экспериментах), сохраненных в разных рабочих каталогах, при решении фиксированной задачи; запускается пользователем по окончании расчетов.

4. Для процедур, в названии которых присутствуют «корни» `parmonc1d, tparmonc1d`,

предполагается, что в пользовательской процедуре реализация случайного объекта - вектор $[\zeta]$; вычисляемые числовые значения:

- $[\bar{\zeta}]$ – вектор, элементы – выборочные средние,
- $[\bar{\varepsilon}]$ – вектор, элементы – соответствующие значения абсолютной погрешности,
- $[\bar{\rho}]$ – вектор, элементы – соответствующие значения относительной погрешности,
- $[\bar{\sigma}^2]$ – вектор, элементы – соответствующие значения выборочной дисперсии.

Кроме того, вычисляются оценки сверху для элементов векторов, содержащих значения абсолютной и относительной погрешности, а также дисперсии.

5. Для процедур, в названии которых присутствует присутствуют «корни» `parmonc2df, tparmonc2df, parmonc2dc, tparmonc2dc`, предполагается, что в пользовательской процедуре реализация случайного объекта - матрица $[\zeta]$; вычисляемые числовые значения:

- $[\bar{\zeta}]$ – матрица, элементы – выборочные средние,
- $[\bar{\varepsilon}]$ – матрица, элементы – соответствующие значения абсолютной погрешности,
- $[\bar{\rho}]$ – матрица, элементы – соответствующие значения относительной погрешности,
- $[\bar{\sigma}^2]$ – матрица, элементы – соответствующие значения выборочной дисперсии.

Кроме того, вычисляются оценки сверху для элементов матриц, содержащих значения абсолютной и относительной погрешности, а также дисперсии.

6. Для процедур, в названии которых присутствуют «корни» `parmonc_corr`,

tparmonc_corr, предполагается, что в каждой пользовательской процедуре реализация случайного объекта - вектор $[\zeta]$; вычисляемые числовые значения:

- $[c\bar{o}v]$ – матрица ковариации сочетаний элементов двух векторов,
- $[\bar{r}]$ – матрица с коэффициентами корреляции сочетаний элементов двух векторов.

Для каждого случайного вектора вычисляемые числовые значения:

- $[\bar{\zeta}]$ – вектор, элементы – выборочные средние,
- $[\bar{\varepsilon}]$ – вектор, элементы – соответствующие значения абсолютной погрешности,
- $[\bar{\rho}]$ – вектор, элементы – соответствующие значения относительной погрешности,
- $[\bar{\sigma}^2]$ – вектор, элементы – соответствующие значения выборочной дисперсии.

Кроме того, вычисляются оценки сверху для элементов векторов, содержащих значения абсолютной и относительной погрешности, а также дисперсии.

7. Для процедур, в названии которых присутствуют «корни»

`parmonc_fnparam, tparmonc_fnparam`, предполагается, что в пользовательской процедуре реализация случайного объекта - вектор $[\zeta]$; вычисляемые числовые значения:

- $[\bar{\zeta}]$ – вектор, элементы – выборочные средние,
- $[\bar{\varepsilon}]$ – вектор, элементы – соответствующие значения абсолютной погрешности,
- $[\bar{\rho}]$ – вектор, элементы – соответствующие значения относительной погрешности,
- $[\bar{\sigma}^2]$ – вектор, элементы – соответствующие значения выборочной дисперсии.

Кроме того, для каждого значения параметра вычисляемые вычисляются оценки сверху для элементов векторов, содержащих значения абсолютной и относительной погрешности, а также дисперсии.

8. Для всех вариантов сочетания архитектуры вычислительной системы и способа

моделирования отдельных реализаций после запуска параллельной программы на кластере в рабочей директории пользователя автоматически создается каталог `/parmonc_data` с вложенными каталогами `/results` и `/service`.

В зависимости от применяемой основной процедуры запуска распределенного статистического моделирования, в каталоге `/parmonc_data/results` будут содержаться разный состав и число файлов с результатами:

1. для процедур, в названии которых присутствуют «корни» `parmonc1d, tparmonc1d, parmonc2df, tparmonc2df, parmonc2dc, tparmonc2dc`, результаты статистического моделирования содержатся в следующих файлах:
   - `func.dat` содержит вектор или матрицу с выборочными средними,
   - `func_ci.dat` содержит вектор или матрицу с выборочными средними вместе с векторами или матрицами со значениями абсолютной и относительной погрешностей,
   - `func_log.dat` содержит информацию о статистическом моделировании.
2. для процедур, в названии которых присутствуют «корни» `parmonc_corr`, `tparmonc_corr`, результаты статистического моделирования содержатся в следующих файлах:
   - `func_covariance.dat, func_corrcoeff.dat` содержит вектор с выборочными средними,
   - `func1.dat, func2.dat` содержит вектор с выборочными средними,
   - `func_ci1.dat, func_ci2.dat` содержит вектор с выборочными средними вместе с векторами со значениями абсолютной и относительной погрешностей,

- `func_log1.dat, func_log2.dat` содержит информацию о статистическом моделировании.

3. для процедур, в названии которых присутствуют «корни» `parmonc_fnparam`, `tparmonc_fnparam`, результаты статистического моделирования сохраняются в отдельных подкаталогах, соответствующих параметрам, для которых производятся вычисления; в каждом таком подкаталоге результаты содержатся в следующих файлах:

   - `func.dat` содержит вектор с выборочными средними,
   - `func_ci.dat` содержит вектор с выборочными средними вместе с векторами со значениями абсолютной и относительной погрешностей,
   - `func_log.dat` содержит информацию о статистическом моделировании.

9. Для всех вариантов в каталоге `/parmonc_data/results/` располагается файл `parmonc_exp.dat`, содержащий информацию о каждом вычислительном эксперименте (число процессоров, время счета, число использованных псевдослучайных чисел, параметры библиотечных процедур).

В следующих разделах описываются библиотечные процедуры для различных сочетаний архитектуры вычислительной системы и способа моделирования отдельных реализаций.

3.1 Процедуры для массивно-параллельной вычислительной системы для случая независимого моделирования отдельных реализаций

В библиотеке используются описанные ниже компоненты, которые сгруппированы по назначению. Здесь приведены названия процедур для языка Fortran, в программах на языке C функции и процедуры вызываются со знаком «_» в конце названия (кроме процедур в п.3).

1. Основные процедуры для запуска распределенного статистического моделирования с целью расчета выборочных средних, дисперсии и оценок погрешности для случая, когда моделируемый случайный объект (отдельная реализация) – вектор.

   Имена процедур: `parmonc1d, tparmonc1d`.

   Набор аргументов:

   ```
   parmonc1d(proc, dim, maxsamp, res, numeseq, perpass, peraver)
   tparmonc1d(proc, dim, maxsamp, res, numseq, perpass, peraver)
   ```

   Описание аргументов:

   - `proc` – имя пользовательской процедуры;

- `dim` – число компонентов в векторе с возвращаемым значением;
- `maxsamp` – максимальный размер выборки на каждом процессоре;
- `res` – флаг нового расчета/возобновления расчетов;
- `numseq` – номер подпоследовательности для экспериментов (подпоследовательности нумеруются, начиная с 0);
- `perpass` – период отправки данных с каждого процессора на центральный процессор-сборщик;
- `peraver` – период получения данных центральным процессором-сборщиком.

В качестве аргумента `proc` используется имя (фактически, адрес) пользовательской процедуры, в которой моделируется и возвращается одна реализация случайного объекта в виде вектора, т.е. одномерного массива. Пользовательская процедура имеет следующие аргументы (Fortran):

```
subroutine proc(dim, sample)
integer*4 dim
real*8 sample(dim)
```

Здесь `sample` – массив, в который возвращается смоделированное значение случайного объекта.

Процедура `parmonc1d` отличаются от `tparmonc1d` единицами измерения аргументов `perpass` и `peraver`, задающих периодичность обмена: для `parmonc1d` – в реализациях, для `tparmonc1d` – в минутах.

2. Основные процедуры для запуска распределенного статистического моделирования с целью расчета выборочных средних, дисперсии и оценок погрешности для случая, когда моделируемый случайный объект (отдельная реализация) – матрица.

   Имена процедур:

   - `parmonc2df, tparmonc2df` – варианты процедур для программ на языке FORTRAN,
   - `parmonc2dc, parmonc2dc` – варианты процедур для программ на языке C.

   Набор аргументов:

```
parmonc2df(proc, dim1, dim2, maxsamp, res, numeseq,
            perpass, peraver)
tparmonc2df(proc, dim1, dim2, maxsamp, res, numeseq,
            perpass, peraver)
parmonc2dc(proc, dim1, dim2, maxsamp, res, numeseq,
           perpass, peraver)
tparmonc2dc(proc, dim1, dim2, maxsamp, res, numeseq,
            perpass, peraver)
```

Описание аргументов:

- `proc` – имя пользовательской процедуры;
- `dim1` – число строк в матрице с возвращаемым значением;
- `dim2` – число столбцов в матрице с возвращаемым значением;
- `maxsamp` – максимальный размер выборки на каждом процессоре;
- `res` – флаг нового расчета/возобновления расчетов;
- `numseq` – номер подпоследовательности для экспериментов (подпоследовательности нумеруются, начиная с 0);
- `perpass` – период отправки данных с каждого процессора на центральный процессор-сборщик;
- `peraver` – период получения данных центральным процессором-сборщиком.

В качестве аргумента `proc` используется имя (фактически, адрес) пользовательской процедуры, в которой моделируется и возвращается одна реализация случайного объекта в виде матрицы, т.е. двухмерного массива. Пользовательская процедура имеет следующие аргументы (Fortran):

```
subroutine proc ( dim1, dim2, sample )
integer*4 dim1, dim2
real*8 sample(dim1, dim2)
```

Здесь `sample` – массив, в который возвращается смоделированное значение случайного объекта.

Процедуры `parmonc2df, tparmonc2dc` отличаются от `tparmonc2df`, `tparmonc2dc` единицами измерения аргументов `perpass` и `peraver`, задающих периодичность обмена: для `parmonc2df, parmonc2dc` – в реализациях, для `tparmonc2df, tparmonc2dc` – в минутах.

3. Основные процедуры для запуска распределенного статистического моделирования с целью расчета ковариаций и коэффициентов корреляции сочетаний компонентов двух моделируемых случайных векторов (а также расчет выборочных средних, дисперсии и оценок погрешности для каждого вектора):

   Имена процедур: `parmonc_corr, tparmonc_corr`.

   Набор аргументов:

```
parmonc_corr(proc1, dim1, proc2, dim2, maxsamp,
             res, numeseq, perpass, peraver)
tparmonc_corr(proc1, dim1, proc2, dim2, maxsamp,
              res, numeseq, perpass, peraver)
```

   Описание аргументов:

- `proc1` – имя первой пользовательской процедуры;
- `dim1` – число компонентов в первом векторе с возвращаемым значением;
- `proc2` – имя второй пользовательской процедуры;
- `dim2` – число компонентов во втором векторе с возвращаемым значением;
- `maxsamp` – максимальный размер выборки на каждом процессоре;
- `res` – флаг нового расчета/возобновления расчетов;
- `numseq` – номер подпоследовательности для экспериментов (подпоследовательности нумеруются, начиная с 0);
- `perpass` – период отправки данных с каждого процессора на центральный процессор-сборщик;
- `peraver` – период получения данных центральным процессором-сборщиком.

В качестве аргументов `proc1, proc2` используются имена (фактически, адреса) пользовательских процедур, в каждой из которых моделируются и возвращаются одна реализация случайного объекта в виде вектора, т.е. одномерного массива. Каждая пользовательская процедура имеет следующие аргументы (Fortran):

```
subroutine proc1(dim1, sample)
integer*4 dim1
real*8 sample(dim1)
```

Здесь `sample` – массив, в который возвращается смоделированное значение случайного объекта.

Процедура `parmonc_corr` отличаются от `tparmonc_corr` единицами измерения аргументов `perpass` и `peraver`, задающих периодичность обмена: для `parmonc_corr` – в реализациях, для `tparmonc_corr` – в минутах.

4. Основная процедура для запуска распределенного статистического моделирования с целью расчета выборочных средних, дисперсии и оценок погрешности для случая, когда моделируемая реализация (вектор) зависит от ряда заданных параметров.

   Имена процедур: `parmonc_fnparam, tparmonc_fnparam`.

   Набор аргументов:

```
parmonc_fnparam(proc, dim, mparam, numrow, numcol,
                res, numeseq, perpass, peraver)
tparmonc_fnparam(proc, dim, mparam, numrow, naumcol,
                res, numeseq, perpass, peraver)
```

   Описание аргументов:

- `proc` – имя пользовательской процедуры;
- `dim` – число компонентов в векторе с возвращаемым значением;
- `mparam` – матрица с параметрами;
- `numrow` – число строк в матрице с параметрами;
- `numcol` – число столбцов в матрице с параметрами;
- `maxsamp` – максимальный размер выборки на каждом процессоре;
- `res` – флаг нового расчета/возобновления расчетов;
- `numseq` – номер подпоследовательности для экспериментов (подпоследовательности нумеруются, начиная с 0);
- `perpass` – период отправки данных с каждого процессора на центральный процессор-сборщик;
- `peraver` – период получения данных центральным процессором-сборщиком.

В качестве аргумента `mparam` используется матрица, содержащая набор значений параметров по строкам, т.е. каждая строка используются в качестве параметра при моделировании в пользовательской процедуре `proc`. Таким образом, число параметров – `numrow`, каждый из них – вектор длины `numcol`.

В качестве аргумента `proc` используется имя (фактически, адрес) пользовательской процедуры, в которой моделируется и возвращается одна реализация случайного объекта в виде вектора, т.е. одномерного массива. Пользовательская процедура имеет следующие аргументы (Fortran):

```
subroutine proc(dim, sample, numcol, vparam)
integer*4 dim, numrow, numcol
real*8 sample(dim), vparam(numcol)
```

Здесь `sample` – массив, в который возвращается смоделированное значение случайного объекта; `vparam` – вектор с конкретным значением параметра.

Процедура `parmonc_fnparam` отличаются от `tparmonc_fnparam` единицами измерения аргументов `perpass` и `peraver`, задающих периодичность обмена: для `parmonc_fnparam` – в реализациях, для
`tparmonc_fnparam` – в минутах.

Вспомогательная программа `genparam` позволяет переопределить множители «прыжков» $A(n_e), A(n_p), A(n_r), A(n_l)$ для параллельного генератора (2).

3.2 Процедуры для массивно-параллельной вычислительной системы для случая моделирования отдельных реализации совместно на нескольких процессорах

Назначение процедур аналогичны варианту библиотеки для массивно-параллельной вычислительной системы для случая, когда моделирование отдельных реализаций осу-

ществляется независимо на разных процессорах (вычислительных ядрах). В названиях компонент прибавляется суффикс «_m», например, `parmonc1d_m`, `parmonc_fnparam_m`.

Важные отличия:

- в состав аргументов всех основных процедур для запуска распределенного статистического моделирования прибавляется один дополнительный аргумент (всегда последний в списке), значение которого – число процессоров (ядер), на которых будет совместно моделироваться отдельная реализация (см ниже);
- в состав аргументов пользовательской процедуры, в которой моделируется одна реализация случайного объекта, включаются два дополнительных входных аргумента: 1) массив, содержащий номера процессоров, на которых совместно моделируется реализация; 2) аргумент, задающий размер такого массива.

В качестве примера приведем описание процедур `parmonc1d_m`, `tparmonc1d_m`. Набор аргументов у них следующий:

```
parmonc1d_m(proc, dim, maxsamp, res, numeseq,
            perpass, peraver, procnum)

tparmonc1d_m(proc, dim, maxsamp, res, numseq,
            perpass, peraver, procnum)
```

Описание аргументов:

- `proc` – имя пользовательской процедуры;
- `dim` – число компонентов в векторе с возвращаемым значением;
- `maxsamp` – максимальный размер выборки на каждом процессоре;
- `res` – флаг нового расчета/возобновления расчетов;
- `numseq` – номер подпоследовательности для экспериментов (подпоследовательности нумеруются, начиная с 0);
- `perpass` – период отправки данных с каждого процессора на центральный процессор-сборщик;
- `peraver` – период получения данных центральным процессором-сборщиком;
- `procnum` – число процессоров (ядер), на которых будет совместно моделироваться отдельная реализация.

В качестве аргумента `proc` используется имя (фактически, адрес) пользовательской процедуры, в которой моделируется и возвращается одна реализация случайного объекта в виде вектора, т.е. одномерного массива. Пользовательская процедура имеет следующие аргументы (Fortran):

```
subroutine proc(dim, sample, procnum, procarray)
integer*4 dim, procnum
real*8 sample(dim)
integer*4 procarray(procnum)
```

Здесь:

- `sample` – массив размерности `dim`, в который возвращается смоделированное значение случайного объекта.
- `procnum` – входной параметр – число процессоров, на которых будет осуществляться совместное моделирование одной реализации; равен значению аргумента `procnum` процедур `parmonc1d_m, tparmonc1d_m`.
- `procarray` – входной параметр с массивом размерности `procnum`, в котором в процедуру передаются номера процессоров, на которых будет осуществляться совместное моделирование одной реализации; значения номеров процессоров в массив подставляются автоматически; в своей процедуре пользователь определяет алгоритм моделирования реализации, используя процедуры MPI и значения номеров процессоров из массива `procarray`.

При вычислении функционалов на процессорах, чьи номера входят в массив `procnum`, следует организовывать пересылку средствами MPI рассчитанных значений на процессор с номером `procarray[0]`. Пересылка с этого процессора рассчитанных значений на центральный процессор-сборщик будет производится средствами библиотеки автоматически.

3.3 Процедуры для гибридной вычислительной системы для случая независимого моделирования отдельных реализаций

Назначение и аргументы процедур аналогичны варианту библиотеки для массивно-параллельной вычислительной системы для случая, когда моделирование отдельных реализаций осуществляется независимо на разных процессорах (вычислительных ядрах). В названиях компонент прибавляется суффикс «_h», например, `parmonc1d_h, parmonc_fnparam_h`.

При этом добавляется процедура `parmonc_coproc_generinith`. Ее назначение – инициализация подпоследовательности генератора на заданного ядра сопроцессора; пользователь добавляет ее в свой код.

3.4 Процедуры для гибридной вычислительной системы для случая моделирования отдельных реализации совместно на нескольких процессорах

Назначение и аргументы аналогичны варианту библиотеки для массивно-параллельной вычислительной системе для случая, когда моделирование каждой реализации осу-

ществляется совместно на нескольких процессорах. В названиях компонент прибавляется суффикс «_hm», например, `parmonc1d_hm, parmonc_fnparam_hm`.

При этом добавляется процедура `parmonc_coproc_generinithm`. Ее назначение – инициализация подпоследовательности генератора на заданного ядра сопроцессора; пользователь добавляет ее в свой код.

4 Примеры использования библиотеки PARMONC

Below we describe typical steps how to: a) write a typical application for stochastic simulation, b) compile it with the use of the PARMONC, c) launch it on the cluster, d) postprocess the results of the simulation, e) analyze results of the simulation, f) resume the simulation after terminating.

The example is provided on the cluster NKS-30T in the directory **/ifs/apps/parmonc/examples/diffusion** (there are different directories for FORTRAN and C examples).

4.1 Описание задачи

We consider a 2-dimensional system of stochastic differential equations (SDEs) over a time interval $[0, 100]$:

$$d\bar{y}(t) = Cdt + Dd\bar{w}(t), \bar{y}(t) = 0$$

where

$$\bar{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}, \bar{w}(t) = \begin{pmatrix} w_1(t) \\ w_2(t) \end{pmatrix}, C = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}, D = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$$

A goal is to evaluate expectations of its components $\mathrm{E}y_1(t),\ \mathrm{E}y_2(t)$ at fixed points $t_i = i \cdot 10^{-1},\ i = 1, \ldots, 1000$. We simulate trajectories of the SDE system using a generalized Euler method with a stepsize $h = 10^{-6}$:

$$\bar{y}^0 = 0, \bar{y}^{(n+1)} = \bar{y}^{(n)} + hC + \sqrt{h}D\bar{\xi}^{(n)}, n = 0, 1, 2, \ldots, 10^8$$

where

$$\bar{y}^{(n)} = \begin{pmatrix} y_1^{(n)} \\ y_2^{(n)} \end{pmatrix}, \bar{\xi}^{(n)} = \begin{pmatrix} \xi_1^{(n)} \\ \xi_2^{(n)} \end{pmatrix}$$

A simulation yields a matrix $[\zeta_{ij}], 1 \le i \le 1000, 1 \le j \le 2$:

$$\zeta_{ij} = y_j^{(n)}, n = i10^5, 1 \le i \le 1000, 1 \le j \le 2.$$

Thus, each element of the matrix gives after averaging

$$\bar{\zeta}_{ij} \approx \mathrm{E}y_j(t_i), t_i = i \cdot 10^{-1},\ i = 1, \ldots, 1000, 1 \le j \le 2.$$

4.2 Пример программы на FORTRAN-е

Below a user-defined program **diffusion.f90** is provided.

Listing 4.2.1 *Main program where* **parmonc** *is called*

```
program diffusion

use parmonc_data
implicit none
integer*4 numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage
external difftraj

numrow = 1000
numcol = 2
maxsampvol = 10**9
resuming = 0
expseqnum = 0
perpass = 10
peraverage = 20

call parmonc2df (difftraj, numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage)

end
```

Note that in this code we set **resuming** $= \mathbf{0}$. We make it because we start a new simulation. Also, we set **expseqnum** $= \mathbf{0}$. Thus, we choose the first (with the number 0) experiments' subsequence.

Listing 4.2.2 *Listing of a subroutine 'difftraj' returning a sample value and other routines where calls to* **rnd128** *are located*

```
subroutine difftraj ( num_row, num_col, sample )

use parmonc_data
implicit none
integer*4 num_row, num_col
real*8 sample(num_row, num_col)
integer i, k, isave
real*8 deltat, y(2), tau, tend

tend = 1.0d2
deltat = 1.0d-6
```

```
isave = 10**5

y(1) = 0.0d0
y(2) = 0.0d0

tau = 0.0d0
i = 0
k = 0

do while (1)
call euler_method (deltat, y)
tau = tau + deltat
i = i + 1
if (tau.gt.tend) then
exit
endif
if ( mod(i, isave) == 0 ) then
k = k + 1
sample(k, 1) = y(1)
sample(k, 2) = y(2)
endif
enddo

return
end
!*********
subroutine euler_method (deltat, y)

use parmonc_data
implicit none
real*8 deltat, y(2)
real*8 d(2, 2)
real*8 c(2), ksi(2)
integer i, j

call normdistr2( ksi )
call convection( c )
call diffusion( d )

do i = 1, 2
y(i) = y(i) + deltat * c(i)
```

```
do j = 1, 2
y(i) = y(i) + dsqrt(deltat) * d(i, j) * ksi(j)
enddo
enddo

end
!*********
subroutine normdistr2 ( ksi )

use parmonc_data
implicit none
real*8 ksi(2)
real*8 r1, r2, r3, e1, pi
real*8, external :: rnd128

pi = 3.14159265358979320
r1 = rnd128()
r2 = rnd128()
e1 = dsqrt( -2.0d0 * dlog(r1) )
r3 = 2.0d0 * pi * r2
ksi(1) = e1 * dsin( r3 )
ksi(2) = e1 * dcos( r3 )

end
!*********
subroutine convection ( c )
implicit none
real*8 c(2)
c(1) = 1.0d0
c(2) = 1.0d0
return
end
!*********
subroutine diffusion ( d )
implicit none
real*8 d(2, 2)
d(1, 1) = 1.0d-2
d(1, 2) = 0.0d0
d(2, 1) = 0.0d0
d(2, 2) = 1.0d-2
end
```

4.3 Пример программы на C

Below a user-defined program **diffusion.c** is provided.

Listing 4.3.1 *Main program where* **parmonc2dc_** *is called*

```
int main()
{
int numrow, numcol, maxsampvol, resuming, expseqnum,
                perpass, peraverage;

numrow = 1000;
numcol = 2;
maxsampvol = pow(10,9);
resuming = 0;
expseqnum = 0;
perpass = 10;
peraverage = 20;

parmonc2dc_ (difftraj, &numrow, &numcol, &maxsampvol, &resuming, &expseqnum,
              &perpass, &peraverage);
}
```

Note that in this code we set **resuming** $= \mathbf{0}$. We make it because we start a new simulation. Also, we set **expseqnum** $= \mathbf{0}$. Thus, we choose the first (with the number 0) experiments' subsequence.

Listing 4.3.2 *Listing of a routine 'difftraj' returning a sample value and other routines where calls to* **rnd128_** *are located*

```
void difftraj (int *numrow, int *numcol, double *sample1d)
{
// assign new variables with values of pointers which define matrix dimensions

int nr=*numrow, nc=*numcol;

// define and initialize a new variable and make type casting for input matrix
// this is a matrix to put sample values in

double (*sample)[nr][nc]=(double (*)[nr][nc])sample1d;

int i, k, isave;
double h, tau, tend, x[2];

tend = 1.0e2;
```

```
h = 1.0e-6;
isave = pow(10,5);

x[0] = 0.0;
x[1] = 0.0;

tau = 0.0;
i = 0;
k = 0;

do
{
Euler_Method (h, x);
tau += h;
i++;
if (tau >= tend) break;
if ( i%isave == 0 )
{
// asigning the output matrix
(*sample)[k][0] = x[0];
(*sample)[k][1] = x[1];
k++;
};
} while(1);
}
//********
void normdistr2d ( double ksi[2] )
{
double r1, r2, r3, e1, pi = 3.1415926535897932;

r1 = rnd128_();
r2 = rnd128_();

e1 = sqrt( -2.0 * log(r1) );
r3 = 2.0 * pi * r2;
ksi[0] = e1 * sin( r3 );
ksi[1] = e1 * cos( r3 );
}
//********
void convection ( double f[2] )
{
f[0] = 0.0;
```

```
f[1] = 0.0;
}
//********
void diffusion ( double (*s)[2] )
{
s[0][0] = 1.0e-2;
s[0][1] = 0.0;
s[1][0] = 0.0;
s[1][1] = 1.0e-2;
}
//********
void Euler_Method (double deltat, double x[2])
{
double f[2], ksi[2], s[2][2];
int i, j;

normdistr2d( ksi );
convection( f );
diffusion( s );

for(i=0;i<2;i++)
{
x[i] += deltat * f[i];
for(j=0;j<2;j++){
x[i] += sqrt(deltat) * s[i][j] * ksi[j];
}
}
}
```

4.4 Компиляция программы

To compile user-defined program and build it using PARMONC, one should launch the following commands:

for programs written in FORTRAN:

```
$ mpiifort -o diffusion -L$PRMCLIB -I$PRMCINC diffusion.f90 -lparmonc
```

for programs written in C:

```
$ mpiicc -o diffusion -L$PRMCLIB -I$PRMCINC diffusion.c -lparmonc
```

Here **diffusion** is the name of the executable file, **diffusion.f90** and **diffusion.c** are the user-defined programs.

4.5 Запуск задания на кластере

To launch a simulation one should start a job on the cluster. To do this, the starting script must be launched. An example of such script (named **parmonc_sh**) is listed below.

Listing 4.5.1. *Example of the starting script*

```
#!/bin/bash
#PBS -V
#PBS -q g6_q
#PBS -r n
#PBS -l nodes=2:ppn=8,cput=04:10:00,walltime=04:15:00
#PBS -N diffusion
#PBS -j oe
date
cd $PBS_O_WORKDIR
pwd
MPD_CON_EXT=‘date‘
##cat $PBS_NODEFILE
mpirun -r ssh  -genv I_MPI_FABRICS shm:ofa -n 16 ./diffusion
date
```

Here we use 2 nodes of the cluster NKS-30T (1 node = 2 processors = 2*4 = 8 cores). Thus, for PARMONC the number of processors equals to 16.

4.6 Анализ результатов

A few lines from the file **func.dat** are given. First line is a file header with names of columns, other lines contain calculated quantities that corresponds to elements of the matrix $[\bar{\zeta}_{ij}]$.

Listing 4.7.1. *Shortened listing of the file* **func.dat**

```
    est001,           est002
    0.20582E-04,    0.14813E-04
    0.26291E-04,    0.30634E-04
    0.30480E-04,    0.18184E-04
    0.32538E-04,    0.22777E-04
...
```

That is, $\mathrm{E}y_1(t_1) \approx 0.20582 \cdot 10^{-4}$, $\mathrm{E}y_2(t_1) \approx 0.14813 \cdot 10^{-4}$; $\mathrm{E}y_1(t_2) \approx 0.26291 \cdot 10^{-4}$, $\mathrm{E}y_2(t_2) \approx 0.30634 \cdot 10^{-4}$, etc.

A few lines from the file **func_ci.dat** are given. First line is a file header with names of columns, other lines contain calculated quantities (only the first bunch of columns is given).

Each bunch of columns with names

```
est<k>, abserr<k>, relerr<k>, var<k>, where <k> is either 1 or 2
```

corresponds to a bunch of k-th columns of the matrices $[\bar{\zeta}_{ij}], [\bar{\varepsilon}_{ij}], [\bar{\rho}_{ij}], [\bar{\sigma}^2_{ij}]$, respectively.

Listing 4.7.2. *Shortened listing of the file* **func_ci.dat**. *Right bunch of columns is cut off*

```
    est001,         abserr001,        relerr001,         var001, ...
    0.20582E-04,    0.58925E-04,    0.28628E+03,    0.99575E-05, ...
    0.26291E-04,    0.83174E-04,    0.31635E+03,    0.19839E-04, ...
    0.30480E-04,    0.10182E-03,    0.33407E+03,    0.29735E-04, ...
 ...
```

That is, $\bar{\zeta}_{11} \approx 0.20582 \cdot 10^{-4}, \bar{\varepsilon}_{11} \approx 0.58925 \cdot 10^{-4}, \bar{\rho}_{11} \approx 286.3\%, \bar{\sigma}^2_{11} \approx 0.995751 \cdot 10^{-5}$, etc. Thus, $\mathrm{E}y_1(t_1) \approx 0.20582 \cdot 10^{-4} \pm 0.58925 \cdot 10^{-4}$, etc.

Below a listing of the file **func_log.dat** is given.

Listing 4.7.3. *Listing of the file* **func_log.dat**

```
    0.11100E+05      total sample volume
    0.10877E+02      mean computational time per sample in sec.
    0.28586E-02      absolute error upper bound
    0.16671E+05      relative error upper bound in %
    0.10078E-01      variance upper bound
```

That is, in total 11100 sample values were simulated, a mean computational time consumed to simulate a sample on a processor is 10.88 sec., the absolute error upper bound is $\bar{\varepsilon}_{max} = 0.28586 \cdot 10^{-2}$, the relative error upper bound is $\bar{\rho}_{max} = 16671\%$, the variance upper bound is $\bar{\sigma}^2_{max} = 0.10078 \cdot 10^{-1}$.

When the job is started, the contents of the file **parmonc_exp.dat** is as follows:

Listing 4.7.4. *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

This data represents main parameters characterizing stochastic experiment.

4.7 Возобновление расчетов после их остановки

To resume simulation previously performed and terminated it is necessary to set the arguments of **parmonc** as described below.

Listing 4.8.1 *Setting arguments of the* **parmonc** *in order to resume simulation*

```
program diffusion

use parmonc_data

implicit none
integer*4 numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage
external fsample

numrow = 1000
numcol = 2
maxsampvol = 10**9
resuming = 1
expseqnum = 1
perpass = 10
peraverage = 20

call parmonc2df (fsample, numrow, numcol, maxsampvol, resuming, expseqnum, &
 perpass, peraverage)

end
```

We changed the argument **resuming** from 0 to 1 (doing it we allow resuming of the simulation). Also, we changed the argument **expseqnum**. The argument **expseqnum** representing the number of experiments' sequence, when we want to perform different stochastic experiment we increase the value of **expseqnum** by 1 in comparison with the previous experiment.

Arguments **maxsampvol, perpass, peraverage** may be changed at the discretion of the user.

Note, that it is important **not to change arguments 'numrow' and 'numcol'**, otherwise the result of computing and averaging may be unpredictable.

When the job is started, the contents of the file **parmonc_exp.dat** becomes as follows:

Listing 4.8.2 *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 14:34:43 2011:
     number of processors =         32
```

```
     resuming = yes
     number of experiment‘s subsequence =      1
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

If we perform another two iterations the contents of the file becomes the following:

Listing 4.7.3 *Listing of the file* **parmonc_exp.dat**

```
Started Wed Apr 20 12:12:52 2011:
     number of processors =         16
     resuming = no
     number of experiment‘s subsequence =      0
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 14:34:43 2011:
     number of processors =         32
     resuming = yes
     number of experiment‘s subsequence =      1
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 15:05:56 2011:
     number of processors =          8
     resuming = yes
     number of experiment‘s subsequence =      2
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
Started Wed Apr 20 17:49:30 2011:
     number of processors =          8
     resuming = yes
     number of experiment‘s subsequence =      3
     passing data period =         10
     averaging period =         20
     max.sample volume =      1000000000
```

5 Приложения

1. Свидетельство о государственной регистрации программы для ЭВМ.



РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016616687

PARMONC - библиотека для реализации технологии распределённого численного статистического моделирования на массивно-параллельных вычислительных системах

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (RU)***

Автор: ***Марченко Михаил Александрович (RU)***

Заявка № **2016611600**

Дата поступления **26 февраля 2016 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***17 июня 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

2. Диплом лауреата конкурса прикладных разработок и исследований в области компьютерных технологий



Список литературы

[1] Михайлов Г. А., Войтишек А. В. Численное статистическое моделирование. Методы Монте-Карло. М.: Издательский центр «Академия», 2006. 368 с.

[2] Dyadkin I. G., Hamilton K. G. A study of 128-bit multipliers for congruential pseudorandom number generators // Comput. Phys. Comm. 2000. Vol. 125, no. 1–3. P. 239–258.

[3] Marchenko M. A. Parallel Pseudorandom Number Generator for Large-scale Monte Carlo Simulations // Lecture Notes in Computer Science. 2007. Vol. 4671. P. 276–282.

[4] Marsaglia G. Random numbers fall mainly in the planes // Proc. Natl. Acad. Sci. 1968. Vol. 61. P. 25–28.

[5] Марченко М. А., Михайлов Г. А. Распределенные вычисления по методу Монте-Карло // Автоматика и телемеханика. 2007. № 5. С. 157–170.

[6] Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002.

[7] Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High-Performance Programming. Elsevier, 2013. 432 p.

[8] Marchenko M. A. PARMONC – A Software Library for Massively Parallel Stochastic Simulation // Lecture Notes in Computer Science. 2011. Vol. 6873. P. 302–315.

[9] Марченко М. А. PARMONC – библиотека для реализации технологии распределенного численного статистического моделирования на массивно-параллельных вычислительных системах. Свидетельство о государственной регистрации программы для ЭВМ № 2016616687, дата государственной регистрации в Реестре программ для ЭВМ 17.06.2016 г.